

axiomTM



The 30 Year Horizon

<i>Manuel Bronstein</i>	<i>William Burge</i>	<i>Timothy Daly</i>
<i>James Davenport</i>	<i>Michael Dewar</i>	<i>Martin Dunstan</i>
<i>Albrecht Fortenbacher</i>	<i>Patrizia Gianni</i>	<i>Johannes Grabmeier</i>
<i>Jocelyn Guidry</i>	<i>Richard Jenks</i>	<i>Larry Lambe</i>
<i>Michael Monagan</i>	<i>Scott Morrison</i>	<i>William Sit</i>
<i>Jonathan Steinbach</i>	<i>Robert Sutor</i>	<i>Barry Trager</i>
<i>Stephen Watt</i>	<i>Jim Wen</i>	<i>Clifton Williamson</i>

Volume 9: Axiom Compiler

Portions Copyright (c) 2005 Timothy Daly

The Blue Bayou image Copyright (c) 2004 Jocelyn Guidry

Portions Copyright (c) 2004 Martin Dunstan

Portions Copyright (c) 1991-2002,
The Numerical Algorithms Group Ltd.
All rights reserved.

This book and the Axiom software is licensed as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga	Roy Adler	Richard Anderson
George Andrews	Henry Baker	Stephen Balzac
Yuriy Baransky	David R. Barton	Gerald Baumgartner
Gilbert Baumsлаг	Fred Blair	Vladimir Bondarenko
Mark Botch	Alexandre Bouyer	Peter A. Broadbery
Martin Brock	Manuel Bronstein	Florian Bundschuh
William Burge	Quentin Carpent	Bob Caviness
Bruce Char	Cheekai Chin	David V. Chudnovsky
Gregory V. Chudnovsky	Josh Cohen	Christophe Conil
Don Coppersmith	George Corliss	Robert Corless
Gary Cornell	Meino Cramer	Claire Di Crescenzo
Timothy Daly Sr.	Timothy Daly Jr.	James H. Davenport
Jean Della Dora	Gabriel Dos Reis	Michael Dewar
Claire DiCrescendo	Sam Dooley	Lionel Ducos
Martin Dunstan	Brian Dupee	Dominique Duval
Robert Edwards	Heow Eide-Goodman	Lars Erickson
Richard Fateman	Bertfried Fauser	Stuart Feldman
Brian Ford	Albrecht Fortenbacher	George Frances
Constantine Frangos	Timothy Freeman	Korrinn Fu
Marc Gaetano	Rudiger Gebauer	Kathy Gerber
Patricia Gianni	Holger Gollan	Teresa Gomez-Diaz
Laureano Gonzalez-Vega	Stephen Gortler	Johannes Grabmeier
Matt Grayson	James Griesmer	Vladimir Grinberg
Oswald Gschnitzer	Jocelyn Guidry	Steve Hague
Vilya Harvey	Satoshi Hamaguchi	Martin Hassner
Ralf Hemmecke	Henderson	Antoine Hersen
Pietro Iglio	Richard Jenks	Kai Kaminski
Grant Keady	Tony Kennedy	Paul Kosinski
Klaus Kusche	Bernhard Kutzler	Larry Lambe
Frederic Lehouby	Michel Levaud	Howard Levy
Rudiger Loos	Michael Lucks	Richard Luczak
Camm Maguire	Bob McElrath	Michael McGettrick
Ian Meikle	David Mentre	Victor S. Miller
Gerard Milmeister	Mohammed Mobarak	H. Michael Moeller
Michael Monagan	Marc Moreno-Maza	Scott Morrison
Mark Murray	William Naylor	C. Andrew Neff
John Nelder	Godfrey Nolan	Arthur Norman
Jinzhong Niu	Michael O'Connor	Kostas Oikonomou
Julian A. Padget	Bill Page	Jaap Weel
Susan Pelzel	Michel Petitot	Didier Pinchon
Claude Quitte	Norman Ramsey	Michael Richardson
Renaud Rioboo	Jean Rivlin	Nicolas Robidoux
Simon Robinson	Michael Rothstein	Martin Rubey
Philip Santas	Alfred Scheerhorn	William Schelter
Gerhard Schneider	Martin Schoenert	Marshall Schor
Fritz Schwarz	Nick Simicich	William Sit
Elena Smirnova	Jonathan Steinbach	Christine Sundaresan
Robert Sutor	Moss E. Sweedler	Eugene Surowitz
James Thatcher	Baldir Thomas	Mike Thomas
Dylan Thurston	Barry Trager	Themos T. Tsikas
Gregory Vanuxem	Bernhard Wall	Stephen Watt
Juergen Weiss	M. Weller	Mark Wegman
James Wen	Thorsten Werther	Michael Wester
John M. Wiley	Berhard Will	Clifton J. Williamson
Stephen Wilson	Shmuel Winograd	Robert Wisbauer
Sandra Wityak	Waldemar Wiwianka	Knut Wolf
Clifford Yapp	David Yun	Richard Zippel
Evelyn Zoernack	Bruno Zuercher	Dan Zwillinger

Contents

0.1	Makefile	1
1	Overview	3
1.1	The Input	4
1.2	The Output, the EQ.nrlib directory	8
1.3	The code.lsp and EQ.lsp files	9
1.4	The code.o file	23
1.5	The info file	23
1.6	The EQ.fn file	26
1.7	The index.kaf file	31
1.7.1	The index offset byte	33
1.7.2	The “loadTimeStuff”	33
1.7.3	The “compilerInfo”	35
1.7.4	The “constructorForm”	42
1.7.5	The “constructorKind”	42
1.7.6	The “constructorModemap”	42
1.7.7	The “constructorCategory”	44
1.7.8	The “sourceFile”	45
1.7.9	The “modemaps”	45
1.7.10	The “operationAlist”	47
1.7.11	The “superDomain”	49
1.7.12	The “signaturesAndLocals”	49
1.7.13	The “attributes”	49
1.7.14	The “predicates”	49
1.7.15	The “abbreviation”	50
1.7.16	The “parents”	50
1.7.17	The “ancestors”	51
1.7.18	The “documentation”	51
1.7.19	The “slotInfo”	53
1.7.20	The “index”	55
2	Compiler top level	57
2.1	Global Data Structures	57
2.2	Pratt Parsing	57
2.3)compile	58

2.3.1	Spad compiler	61
2.4	Operator Precedence Table Initialization	62
2.4.1	LED and NUD Tables	62
2.5	Gliph Table	65
2.5.1	Rename Token Table	65
2.5.2	Generic function table	66
2.6	Giant steps, Baby steps	66
3	The Parser	67
3.1	EQ.spad	67
3.2	preparse	71
3.2.1	defvar \$index	72
3.2.2	defvar \$linelist	72
3.2.3	defvar \$echolinestack	72
3.2.4	defvar \$preparse-last-line	72
3.3	Parsing routines	72
3.3.1	defun initialize-preparse	73
3.3.2	defun preparse	76
3.3.3	defun Build the lines from the input for piles	81
3.3.4	defun parsepiles	84
3.3.5	defun add-parens-and-semis-to-line	84
3.3.6	defun preparseReadLine	85
3.3.7	defun skip-ifblock	86
3.3.8	defun preparseReadLine1	87
3.4	I/O Handling	88
3.4.1	defun preparse-echo	88
3.4.2	Parsing stack	88
3.4.3	defstruct \$stack	88
3.4.4	defun stack-load	89
3.4.5	defun stack-clear	89
3.4.6	defmacro stack-/empty	89
3.4.7	defun stack-push	89
3.4.8	defun stack-pop	90
3.4.9	Parsing token	90
3.4.10	defstruct \$token	90
3.4.11	defvar \$prior-token	90
3.4.12	defvar \$nonblank	91
3.4.13	defvar \$current-token	91
3.4.14	defvar \$next-token	91
3.4.15	defvar \$valid-tokens	91
3.4.16	defun token-install	92
3.4.17	defun token-print	92
3.4.18	Parsing reduction	92
3.4.19	defstruct \$reduction	92

4	Parse Transformers	93
4.1	Direct called parse routines	93
4.1.1	defun parseTransform	93
4.1.2	defun parseTran	93
4.1.3	defun parseAtom	94
4.1.4	defun parseTranList	95
4.1.5	defplist parseConstruct	95
4.1.6	defun parseConstruct	95
4.2	Indirect called parse routines	96
4.2.1	defplist parseAnd	97
4.2.2	defun parseAnd	97
4.2.3	defplist parseAtSign	97
4.2.4	defun parseAtSign	98
4.2.5	defun parseType	98
4.2.6	defplist parseCategory	98
4.2.7	defun parseCategory	99
4.2.8	defun parseDropAssertions	99
4.2.9	defplist parseCoerce	99
4.2.10	defun parseCoerce	100
4.2.11	defplist parseColon	100
4.2.12	defun parseColon	100
4.2.13	defplist parseDEF	101
4.2.14	defun parseDEF	101
4.2.15	defun parseLhs	102
4.2.16	defun transIs	102
4.2.17	defun transIs1	102
4.2.18	defun isListConstructor	103
4.2.19	defplist parseDollarGreaterthan	104
4.2.20	defun parseDollarGreaterthan	104
4.2.21	defplist parseDollarGreaterEqual	104
4.2.22	defun parseDollarGreaterEqual	104
4.2.23	defun parseDollarLessEqual	105
4.2.24	defplist parseDollarNotEqual	105
4.2.25	defun parseDollarNotEqual	105
4.2.26	defplist parseEquivalence	106
4.2.27	defun parseEquivalence	106
4.2.28	defplist parseExit	106
4.2.29	defun parseExit	107
4.2.30	defplist parseGreaterEqual	107
4.2.31	defun parseGreaterEqual	107
4.2.32	defplist parseGreaterthan	108
4.2.33	defun parseGreaterthan	108
4.2.34	defplist parseHas	108
4.2.35	defun parseHas	108
4.2.36	defun parseHasRhs	110
4.2.37	defun loadIfNecessary	111

4.2.38	defun loadLibIfNecessary	111
4.2.39	defun updateCategoryFrameForConstructor	112
4.2.40	defun convertOpAlist2compilerInfo	112
4.2.41	defun updateCategoryFrameForCategory	113
4.2.42	defplist parseIf	113
4.2.43	defun parseIf	114
4.2.44	defun parseIf,ifTran	114
4.2.45	defplist parseImplies	116
4.2.46	defun parseImplies	116
4.2.47	defplist parseIn	117
4.2.48	defun parseIn	117
4.2.49	defplist parseInBy	118
4.2.50	defun parseInBy	118
4.2.51	defplist parseIs	119
4.2.52	defun parseIs	119
4.2.53	defplist parseIsnt	119
4.2.54	defun parseIsnt	120
4.2.55	defplist parseJoin	120
4.2.56	defun parseJoin	120
4.2.57	defplist parseLeave	121
4.2.58	defun parseLeave	121
4.2.59	defplist parseLessEqual	121
4.2.60	defun parseLessEqual	122
4.2.61	defplist parseLET	122
4.2.62	defun parseLET	122
4.2.63	defplist parseLETD	123
4.2.64	defun parseLETD	123
4.2.65	defplist parseMDEF	123
4.2.66	defun parseMDEF	123
4.2.67	defplist parseNot	124
4.2.68	defun parseNot	124
4.2.69	defun parseNot	124
4.2.70	defplist parseNotEqual	125
4.2.71	defun parseNotEqual	125
4.2.72	defplist parseOr	125
4.2.73	defun parseOr	125
4.2.74	defplist parsePretend	126
4.2.75	defun parsePretend	126
4.2.76	defplist parseReturn	127
4.2.77	defun parseReturn	127
4.2.78	defplist parseSegment	127
4.2.79	defun parseSegment	128
4.2.80	defplist parseSeq	128
4.2.81	defun parseSeq	128
4.2.82	defplist parseVCONS	129
4.2.83	defun parseVCONS	129

4.2.84	defplist parseWhere	129
4.2.85	defun parseWhere	129
5	Compile Transformers	131
5.0.86	defvar \$NoValueMode	131
5.0.87	defvar \$EmptyMode	131
5.1	Routines for handling forms	131
5.2	Functions which handle == statements	133
5.2.1	defun compDefineAddSignature	133
5.2.2	defun hasFullSignature	134
5.2.3	defun addEmptyCapsuleIfNecessary	134
5.2.4	defun getTargetFromRhs	135
5.2.5	defun giveFormalParametersValues	135
5.2.6	defun macroExpandInPlace	136
5.2.7	defun macroExpand	136
5.2.8	defun macroExpandList	137
5.2.9	defun compDefineCategory1	137
5.2.10	defun makeCategoryPredicates	138
5.2.11	defun mkCategoryPackage	139
5.2.12	defun mkEvalableCategoryForm	140
5.2.13	defun compDefineCategory2	142
5.2.14	defun compile	145
5.2.15	defun encodeFunctionName	148
5.2.16	defun mkRepetitionAssoc	149
5.2.17	defun splitEncodedFunctionName	149
5.2.18	defun encodeItem	150
5.2.19	defun getCaps	150
5.2.20	defun constructMacro	151
5.2.21	defun spadCompileOrSetq	151
5.2.22	defun compileConstructor	153
5.2.23	defun compileConstructor1	153
5.2.24	defun putInLocalDomainReferences	154
5.2.25	defun NRTputInTail	154
5.2.26	defun NRTputInHead	155
5.2.27	defun getArgumentModeOrMoan	156
5.2.28	defun augLisplibModemapsFromCategory	157
5.2.29	defun mkAlistOfExplicitCategoryOps	158
5.2.30	defun flattenSignatureList	159
5.2.31	defun interactiveModemapForm	160
5.2.32	defun replaceVars	161
5.2.33	defun fixUpPredicate	161
5.2.34	defun orderPredicateItems	162
5.2.35	defun signatureTran	163
5.2.36	defun orderPredTran	163
5.2.37	defun isDomainSubst	166
5.2.38	defun moveORsOutside	167

5.2.39	defun substVars	168
5.2.40	defun modemapPattern	169
5.2.41	defun evalAndRwriteLispForm	169
5.2.42	defun rwriteLispForm	170
5.2.43	defun mkConstructor	170
5.2.44	defun compDefineCategory	170
5.2.45	defun compDefineLisplib	171
5.2.46	defun unloadOneConstructor	173
5.2.47	defun compileDocumentation	174
5.2.48	defun lisplibDoRename	174
5.2.49	defun initializeLisplib	175
5.2.50	defun writeLib1	176
5.2.51	defun finalizeLisplib	176
5.2.52	defun getConstructorOpsAndAtts	178
5.2.53	defun getCategoryOpsAndAtts	178
5.2.54	defun getSlotFromCategoryForm	179
5.2.55	defun transformOperationAlist	179
5.2.56	defun getFunctorOpsAndAtts	181
5.2.57	defun getSlotFromFunctor	181
5.2.58	defun compMakeCategoryObject	182
5.2.59	defun mergeSignatureAndLocalVarAlists	182
5.2.60	defun lisplibWrite	182
5.2.61	defun compDefineFunctor	183
5.2.62	defun compDefineFunctor1	183
5.2.63	defun isCategoryPackageName	190
5.2.64	defun NRTgetLookupFunction	191
5.2.65	defun NRTgetLocalIndex	192
5.2.66	defun augmentLisplibModemapsFromFunctor	193
5.2.67	defun allLASSOCs	194
5.2.68	defun formal2Pattern	194
5.2.69	defun mkDatabasePred	195
5.2.70	defun disallowNilAttribute	195
5.2.71	defun compFunctorBody	195
5.2.72	defun bootStrapError	196
5.2.73	defun reportOnFunctorCompilation	197
5.2.74	defun displayMissingFunctions	197
5.2.75	defun makeFunctorArgumentParameters	198
5.2.76	defun genDomainViewList0	200
5.2.77	defun genDomainViewList	201
5.2.78	defun genDomainView	201
5.2.79	defun genDomainOps	202
5.2.80	defun mkOpVec	203
5.2.81	defun AssocBarGensym	204
5.2.82	defun compDefWhereClause	204
5.2.83	defun orderByDependency	207
5.3	Code optimization routines	208

5.3.1	defun optimizeFunctionDef	208
5.3.2	defun optimize	209
5.3.3	defun optXLAMCond	210
5.3.4	defun optCONDtail	211
5.3.5	defvar \$BasicPredicates	211
5.3.6	defun optPredicateIfTrue	211
5.3.7	defun optIF2COND	212
5.3.8	defun subrname	212
5.3.9	Special case optimizers	213
5.3.10	defplist optCall	213
5.3.11	defun Optimize “call” expressions	214
5.3.12	defun optPackageCall	215
5.3.13	defun optCallSpecially	215
5.3.14	defun optSpecialCall	216
5.3.15	defun compileTimeBindingOf	217
5.3.16	defun optCallEval	218
5.3.17	defplist optSEQ	218
5.3.18	defun optSEQ	218
5.3.19	defplist optEQ	220
5.3.20	defun optEQ	220
5.3.21	defplist optMINUS	220
5.3.22	defun optMINUS	221
5.3.23	defplist optQSMINUS	221
5.3.24	defun optQSMINUS	221
5.3.25	defplist opt-	222
5.3.26	defun opt-	222
5.3.27	defplist optLESSP	222
5.3.28	defun optLESSP	222
5.3.29	defplist optSPADCALL	223
5.3.30	defun optSPADCALL	223
5.3.31	defplist optSuchthat	224
5.3.32	defun optSuchthat	224
5.3.33	defplist optCatch	224
5.3.34	defun optCatch	225
5.3.35	defplist optCond	226
5.3.36	defun optCond	227
5.3.37	defun EqualBarGensym	228
5.3.38	defplist optMkRecord	229
5.3.39	defun optMkRecord	229
5.3.40	defplist optRECORDELT	230
5.3.41	defun optRECORDELT	230
5.3.42	defplist optSETRECORDELT	230
5.3.43	defun optSETRECORDELT	231
5.3.44	defplist optRECORDCOPY	231
5.3.45	defun optRECORDCOPY	231
5.4	Functions to manipulate modemaps	232

5.4.1	defun addDomain	232
5.4.2	defun unknownTypeError	233
5.4.3	defun isFunctor	233
5.4.4	defun getDomainsInScope	234
5.4.5	defun putDomainsInScope	235
5.4.6	defun isSuperDomain	235
5.4.7	defun addNewDomain	236
5.4.8	defun augModemapsFromDomain	236
5.4.9	defun augModemapsFromDomain1	237
5.4.10	defun substituteCategoryArguments	237
5.4.11	defun addConstructorModemaps	238
5.4.12	defun getModemap	239
5.4.13	defun compApplyModemap	239
5.4.14	defun compMapCond	240
5.4.15	defun compMapCond'	241
5.4.16	defun compMapCond''	241
5.4.17	defun compMapCondFun	243
5.4.18	defun getUniqueSignature	243
5.4.19	defun getUniqueModemap	243
5.4.20	defun getModemapList	244
5.4.21	defun getModemapListFromDomain	244
5.4.22	defun domainMember	244
5.4.23	defun augModemapsFromCategory	245
5.4.24	defun addEltModemap	245
5.4.25	defun mkNewModemapList	246
5.4.26	defun insertModemap	247
5.4.27	defun mergeModemap	248
5.4.28	defun TruthP	249
5.4.29	defun evalAndSub	249
5.4.30	defun getOperationAlist	250
5.4.31	defvar \$FormalMapVariableList	250
5.4.32	defun substNames	251
5.4.33	defun augModemapsFromCategoryRep	251
5.5	Maintaining Modemaps	253
5.5.1	defun addModemapKnown	253
5.5.2	defun addModemap	253
5.5.3	defun addModemap0	254
5.5.4	defun addModemap1	254
5.6	Indirect called comp routines	255
5.6.1	defplist compAdd plist	255
5.6.2	defun compAdd	255
5.6.3	defun compTuple2Record	258
5.6.4	defplist compCapsule plist	258
5.6.5	defun compCapsule	258
5.6.6	defun compCapsuleInner	259
5.6.7	defun processFunctor	259

5.6.8	defun compCapsuleItems	260
5.6.9	defun compSingleCapsuleItem	261
5.6.10	defun doIt	261
5.6.11	defun doItIf	265
5.6.12	defun isMacro	267
5.6.13	defplist compCase plist	267
5.6.14	defun compCase	268
5.6.15	defun compCase1	268
5.6.16	defplist compCat plist	269
5.6.17	defplist compCat plist	269
5.6.18	defplist compCat plist	269
5.6.19	defun compCat	270
5.6.20	defplist compCategory plist	270
5.6.21	defun compCategory	270
5.6.22	defun compCategoryItem	271
5.6.23	defun mkExplicitCategoryFunction	273
5.6.24	defun mustInstantiate	274
5.6.25	defun wrapDomainSub	274
5.6.26	defplist compColon plist	274
5.6.27	defun compColon	275
5.6.28	defun makeCategoryForm	278
5.6.29	defplist compCons plist	278
5.6.30	defun compCons	278
5.6.31	defun compCons1	279
5.6.32	defplist compConstruct plist	280
5.6.33	defun compConstruct	280
5.6.34	defplist compConstructorCategory plist	281
5.6.35	defplist compConstructorCategory plist	281
5.6.36	defplist compConstructorCategory plist	281
5.6.37	defplist compConstructorCategory plist	281
5.6.38	defun compConstructorCategory	282
5.6.39	defplist compDefine plist	282
5.6.40	defun compDefine	282
5.6.41	defun compDefine1	283
5.6.42	defun getAbbreviation	285
5.6.43	defun mkAbbrev	286
5.6.44	defun addSuffix	286
5.6.45	defun alistSize	286
5.6.46	defun getSignatureFromMode	287
5.6.47	defun compInternalFunction	287
5.6.48	defun compDefineCapsuleFunction	288
5.6.49	defun compileCases	291
5.6.50	defun getSpecialCaseAssoc	293
5.6.51	defun addArgumentConditions	293
5.6.52	defun compArgumentConditions	294
5.6.53	defun stripOffSubdomainConditions	295

5.6.54	defun stripOffArgumentConditions	296
5.6.55	defun getSignature	296
5.6.56	defun checkAndDeclare	298
5.6.57	defun hasSigInTargetCategory	298
5.6.58	defun getArgumentMode	299
5.6.59	defplist compElt plist	300
5.6.60	defun compElt	300
5.6.61	defplist compExit plist	301
5.6.62	defun compExit	301
5.6.63	defplist compHas plist	302
5.6.64	defun compHas	302
5.6.65	defun compHasFormat	303
5.6.66	defun mkList	304
5.6.67	defplist compIf plist	304
5.6.68	defun compIf	304
5.6.69	defun compFromIf	305
5.6.70	defun canReturn	306
5.6.71	defun compBoolean	308
5.6.72	defun getSuccessEnvironment	308
5.6.73	defun getInverseEnvironment	310
5.6.74	defun getUnionMode	311
5.6.75	defun isUnionMode	311
5.6.76	defplist compImport plist	312
5.6.77	defun compImport	312
5.6.78	defplist compIs plist	312
5.6.79	defun compIs	312
5.6.80	defplist compJoin plist	313
5.6.81	defun compJoin	313
5.6.82	defun compForMode	315
5.6.83	defplist compLambda plist	315
5.6.84	defun compLambda	315
5.6.85	defplist compLeave plist	316
5.6.86	defun compLeave	317
5.6.87	defplist compMacro plist	317
5.6.88	defun compMacro	317
5.6.89	defplist compPretend plist	318
5.6.90	defun compPretend	318
5.6.91	defplist compQuote plist	319
5.6.92	defun compQuote	320
5.6.93	defplist compReduce plist	320
5.6.94	defun compReduce	320
5.6.95	defun compReduce1	320
5.6.96	defplist compRepeatOrCollect plist	322
5.6.97	defplist compRepeatOrCollect plist	322
5.6.98	defun compRepeatOrCollect	323
5.6.99	defplist compReturn plist	325

5.6.100	defun compReturn	325
5.6.101	defplist compSeq plist	326
5.6.102	defun compSeq	326
5.6.103	defun compSeq1	326
5.6.104	defun replaceExitEtc	327
5.6.105	defun convertOrCroak	328
5.6.106	defun compSeqItem	328
5.6.107	defplist compSetq plist	329
5.6.108	defplist compSetq plist	329
5.6.109	defun compSetq	329
5.6.110	defun compSetq1	329
5.6.111	defun uncons	330
5.6.112	defun setqMultiple	330
5.6.113	defun setqMultipleExplicit	333
5.6.114	defun setqSetelt	334
5.6.115	defun setqSingle	334
5.6.116	defun NRTassocIndex	336
5.6.117	defun assignError	336
5.6.118	defun outputComp	337
5.6.119	defun maxSuperType	338
5.6.120	defun isDomainForm	338
5.6.121	defun isDomainConstructorForm	339
5.6.122	defplist compString plist	339
5.6.123	defun compString	339
5.6.124	defplist compSubDomain plist	340
5.6.125	defun compSubDomain	340
5.6.126	defun compSubDomain1	341
5.6.127	defun lispize	342
5.6.128	defplist compSubsetCategory plist	342
5.6.129	defun compSubsetCategory	342
5.6.130	defplist compSuchthat plist	343
5.6.131	defun compSuchthat	343
5.6.132	defplist compVector plist	343
5.6.133	defun compVector	344
5.6.134	defplist compWhere plist	344
5.6.135	defun compWhere	345
5.7	Functions for coercion	346
5.7.1	defun coerce	346
5.7.2	defun coerceEasy	346
5.7.3	defun coerceSubset	347
5.7.4	defun coerceHard	348
5.7.5	defun coerceExtraHard	349
5.7.6	defun hasType	350
5.7.7	defun coerceable	350
5.7.8	defun coerceExit	351
5.7.9	defplist compAtSign plist	351

5.7.10	defun compAtSign	352
5.7.11	defplist compCoerce plist	352
5.7.12	defun compCoerce	352
5.7.13	defun compCoerce1	353
5.7.14	defun coerceByModemap	354
5.7.15	defun autoCoerceByModemap	354
5.7.16	defun resolve	356
5.7.17	defun mkUnion	356
5.7.18	defun This orders Unions	357
5.7.19	defun modeEqualSubst	357
5.7.20	compilerDoitWithScreenedLisplib	358
6	Post Transformers	359
6.1	Direct called postparse routines	359
6.1.1	defun postTransform	359
6.1.2	defun postTran	360
6.1.3	defun postOp	361
6.1.4	defun postAtom	361
6.1.5	defun postTranList	362
6.1.6	defun postScriptsForm	362
6.1.7	defun postTranScripts	362
6.1.8	defun postTransformCheck	363
6.1.9	defun postcheck	363
6.1.10	defun postError	364
6.1.11	defun postForm	364
6.2	Indirect called postparse routines	365
6.2.1	defplist postAdd plist	366
6.2.2	defun postAdd	366
6.2.3	defun postCapsule	367
6.2.4	defun postBlockItemList	367
6.2.5	defun postBlockItem	368
6.2.6	defplist postAtSign plist	368
6.2.7	defun postAtSign	369
6.2.8	defun postType	369
6.2.9	defplist postBigFloat plist	369
6.2.10	defun postBigFloat	370
6.2.11	defplist postBlock plist	370
6.2.12	defun postBlock	370
6.2.13	defplist postCategory plist	371
6.2.14	defun postCategory	371
6.2.15	defun postCollect,finish	372
6.2.16	defun postMakeCons	372
6.2.17	defplist postCollect plist	373
6.2.18	defun postCollect	373
6.2.19	defun postIteratorList	374
6.2.20	defplist postColon plist	374

6.2.21	defun postColon	375
6.2.22	defplist postColonColon plist	375
6.2.23	defun postColonColon	375
6.2.24	defplist postComma plist	376
6.2.25	defun postComma	376
6.2.26	defun comma2Tuple	376
6.2.27	defun postFlatten	376
6.2.28	defplist postConstruct plist	377
6.2.29	defun postConstruct	377
6.2.30	defun postTranSegment	378
6.2.31	defplist postDef plist	378
6.2.32	defun postDef	378
6.2.33	defun postDefArgs	380
6.2.34	defplist postExit plist	381
6.2.35	defun postExit	381
6.2.36	defplist postIf plist	381
6.2.37	defun postIf	381
6.2.38	defplist postin plist	382
6.2.39	defun postin	382
6.2.40	defun postInSeq	382
6.2.41	defplist postIn plist	383
6.2.42	defun postIn	383
6.2.43	defplist postJoin plist	383
6.2.44	defun postJoin	384
6.2.45	defplist postMapping plist	384
6.2.46	defun postMapping	384
6.2.47	defplist postMDef plist	385
6.2.48	defun postMDef	385
6.2.49	defplist postPretend plist	386
6.2.50	defun postPretend	386
6.2.51	defplist postQUOTE plist	387
6.2.52	defun postQUOTE	387
6.2.53	defplist postReduce plist	387
6.2.54	defun postReduce	387
6.2.55	defplist postRepeat plist	388
6.2.56	defun postRepeat	388
6.2.57	defplist postScripts plist	388
6.2.58	defun postScripts	389
6.2.59	defplist postSemiColon plist	389
6.2.60	defun postSemiColon	389
6.2.61	defun postFlattenLeft	389
6.2.62	defplist postSignature plist	390
6.2.63	defun postSignature	390
6.2.64	defun removeSuperfluousMapping	391
6.2.65	defun killColons	391
6.2.66	defplist postSlash plist	391

6.2.67	defun postSlash	391
6.2.68	defplist postTuple plist	392
6.2.69	defun postTuple	392
6.2.70	defplist postTupleCollect plist	392
6.2.71	defun postTupleCollect	393
6.2.72	defplist postWhere plist	393
6.2.73	defun postWhere	393
6.2.74	defplist postWith plist	394
6.2.75	defun postWith	394
6.3	Support routines	394
6.3.1	defun setDefOp	394
6.3.2	defun aplTran	395
6.3.3	defun aplTran1	395
6.3.4	defun aplTranList	397
6.3.5	defun hasAplExtension	397
6.3.6	defun deepestExpression	398
6.3.7	defun containsBang	398
6.3.8	defun getScriptName	399
6.3.9	defun decodeScripts	399
7	DEF forms	401
7.0.10	defvar \$defstack	401
7.0.11	defvar \$is-spill	401
7.0.12	defvar \$is-spill-list	401
7.0.13	defvar \$vl	402
7.0.14	defvar \$is-gensymlist	402
7.0.15	defvar \$initial-gensym	402
7.0.16	defvar \$is-eqlist	402
7.0.17	defun hackforis	402
7.0.18	defun hackforis1	403
7.0.19	defun unTuple	403
7.0.20	defun errhuh	403
8	PARSE forms	405
8.1	The original meta specification	405
8.2	The PARSE code	410
8.2.1	defvar \$tmptok	410
8.2.2	defvar \$tok	410
8.2.3	defvar \$ParseMode	411
8.2.4	defvar \$definition-name	411
8.2.5	defvar \$lablasoc	411
8.2.6	defun PARSE-NewExpr	411
8.2.7	defun PARSE-Command	412
8.2.8	defun PARSE-SpecialKeyWord	412
8.2.9	defun PARSE-SpecialCommand	413
8.2.10	defun PARSE-TokenCommandTail	413

8.2.11	defun PARSE-TokenOption	414
8.2.12	defun PARSE-TokenList	414
8.2.13	defun PARSE-CommandTail	415
8.2.14	defun PARSE-PrimaryOrQM	415
8.2.15	defun PARSE-Option	416
8.2.16	defun PARSE-Statement	416
8.2.17	defun PARSE-InfixWith	417
8.2.18	defun PARSE-With	417
8.2.19	defun PARSE-Category	417
8.2.20	defun PARSE-Expression	419
8.2.21	defun PARSE-Import	419
8.2.22	defun PARSE-Expr	420
8.2.23	defun PARSE-LedPart	420
8.2.24	defun PARSE-NudPart	420
8.2.25	defun PARSE-Operation	421
8.2.26	defun PARSE-leftBindingPowerOf	421
8.2.27	defun PARSE-rightBindingPowerOf	422
8.2.28	defun PARSE-getSemanticForm	422
8.2.29	defun PARSE-Prefix	422
8.2.30	defun PARSE-Infix	423
8.2.31	defun PARSE-TokTail	424
8.2.32	defun PARSE-Qualification	424
8.2.33	defun PARSE-Reduction	425
8.2.34	defun PARSE-ReductionOp	425
8.2.35	defun PARSE-Form	425
8.2.36	defun PARSE-Application	426
8.2.37	defun PARSE-Label	427
8.2.38	defun PARSE-Selector	427
8.2.39	defun PARSE-PrimaryNoFloat	428
8.2.40	defun PARSE-Primary	428
8.2.41	defun PARSE-Primary1	428
8.2.42	defun PARSE-Float	429
8.2.43	defun PARSE-FloatBase	430
8.2.44	defun PARSE-FloatBasePart	430
8.2.45	defun PARSE-FloatExponent	431
8.2.46	defun PARSE-Enclosure	432
8.2.47	defun PARSE-IntegerTok	432
8.2.48	defun PARSE-FormalParameter	433
8.2.49	defun PARSE-FormalParameterTok	433
8.2.50	defun PARSE-Quad	433
8.2.51	defun PARSE-String	433
8.2.52	defun PARSE-VarForm	434
8.2.53	defun PARSE-Scripts	434
8.2.54	defun PARSE-ScriptItem	435
8.2.55	defun PARSE-Name	435
8.2.56	defun PARSE-Data	436

8.2.57	defun PARSE-Sexpr	436
8.2.58	defun PARSE-Sexpr1	436
8.2.59	defun PARSE-NBGlyphTok	437
8.2.60	defun PARSE-GlyphTok	438
8.2.61	defun PARSE-AnyId	438
8.2.62	defun PARSE-Sequence	439
8.2.63	defun PARSE-Sequence1	439
8.2.64	defun PARSE-OpenBracket	440
8.2.65	defun PARSE-OpenBrace	440
8.2.66	defun PARSE-IteratorTail	441
8.2.67	defun PARSE-Iterator	441
8.2.68	The PARSE implicit routines	442
8.2.69	defun PARSE-Suffix	442
8.2.70	defun PARSE-SemiColon	443
8.2.71	defun PARSE-Return	443
8.2.72	defun PARSE-Exit	443
8.2.73	defun PARSE-Leave	444
8.2.74	defun PARSE-Seg	444
8.2.75	defun PARSE-Conditional	445
8.2.76	defun PARSE-ElseClause	445
8.2.77	defun PARSE-Loop	446
8.2.78	defun PARSE-LabelExpr	446
8.2.79	defun PARSE-FloatTok	447
8.3	The PARSE support routines	447
8.3.1	String grabbing	448
8.3.2	defun match-string	448
8.3.3	defun skip-blanks	448
8.3.4	defun token-lookahead-type	449
8.3.5	defun match-advance-string	449
8.3.6	defun initial-substring-p	450
8.3.7	defun quote-if-string	450
8.3.8	defun escape-keywords	451
8.3.9	defun isTokenDelimiter	451
8.3.10	defun underscore	452
8.3.11	Token Handling	452
8.3.12	defun getToken	452
8.3.13	defun unget-tokens	452
8.3.14	defun match-current-token	453
8.3.15	defun match-token	454
8.3.16	defun match-next-token	454
8.3.17	defun current-symbol	454
8.3.18	defun make-symbol-of	454
8.3.19	defun current-token	455
8.3.20	defun try-get-token	455
8.3.21	defun next-token	456
8.3.22	defun advance-token	456

8.3.23	defvar \$XTokenReader	457
8.3.24	defun get-token	457
8.3.25	Character handling	457
8.3.26	defun current-char	457
8.3.27	defun next-char	457
8.3.28	defun char-eq	458
8.3.29	defun char-ne	458
8.3.30	Error handling	458
8.3.31	defvar \$meta-error-handler	458
8.3.32	defun meta-syntax-error	459
8.3.33	Floating Point Support	459
8.3.34	defun floatexpid	459
8.3.35	Dollar Translation	459
8.3.36	defun dollarTran	459
8.3.37	Applying metagrammatical elements of a production (e.g., Star). . . .	460
8.3.38	defmacro Bang	460
8.3.39	defmacro must	460
8.3.40	defun action	461
8.3.41	defun optional	461
8.3.42	defmacro star	461
8.3.43	Stacking and retrieving reductions of rules.	462
8.3.44	defvar \$reduce-stack	462
8.3.45	defmacro reduce-stack-clear	462
8.3.46	defun push-reduction	462
9	Comment handlers	463
9.0.47	defun recordSignatureDocumentation	463
9.0.48	defun recordAttributeDocumentation	463
9.0.49	defun recordDocumentation	464
9.0.50	defun recordHeaderDocumentation	464
9.0.51	defun collectComBlock	465
9.0.52	defun collectAndDeleteAssoc	465
9.0.53	defun finalizeDocumentation	466
9.1	Transformation of ++ comments	468
9.1.1	defun transDocList	468
9.1.2	defun transDoc	469
9.1.3	defun transformAndRecheckComments	470
9.1.4	defun checkRewrite	471
9.1.5	defun checkRecordHash	472
9.1.6	defun checkGetParse	475
9.1.7	defun removeBackslashes	476
9.1.8	defun checkTexht	476
9.1.9	defun checkDecorateForHt	477
9.1.10	defun checkDocError1	478
9.1.11	defun checkDocError	478
9.1.12	defun checkDocMessage	479

9.1.13	defun whoOwns	480
9.1.14	defun checkComments	480
9.1.15	defun checkSplit2Words	482
9.1.16	defun checkAddPeriod	483
9.1.17	defun checkBalance	483
9.1.18	defun checkBeginEnd	484
9.1.19	defun checkSayBracket	486
9.1.20	defun checkArguments	486
9.1.21	defun checkHTargs	486
9.1.22	defun checkLookForLeftBrace	487
9.1.23	defun checkLookForRightBrace	487
9.1.24	defun checkTransformFirsts	488
9.1.25	defun checkSkipBlanks	491
9.1.26	defun checkSkipIdentifierToken	491
9.1.27	defun checkAlphabetic	491
9.1.28	defun checkSkipToken	492
9.1.29	defun checkSkipOpToken	492
9.1.30	defun getMatchingRightPren	492
9.1.31	defun checkGetMargin	493
9.1.32	defun firstNonBlankPosition	493
9.1.33	defun checkIeEg	494
9.1.34	defun checkIeEgfun	494
9.1.35	defun checkSplitBrace	495
9.1.36	defun checkSplitBackslash	496
9.1.37	defun checkSplitPunctuation	497
9.1.38	defun checkSplitOn	498
9.1.39	defun checkNumOfArgs	499
9.1.40	defun checkRemoveComments	500
9.1.41	defun checkTrimCommented	500
9.1.42	defun htcharPosition	501
9.1.43	defun checkAddMacros	501
9.1.44	defun checkIndentedLines	502
9.1.45	defun newString2Words	503
9.1.46	defun newWordFrom	503
9.1.47	defun checkGetArgs	504
9.1.48	defun checkAddSpaceSegments	505
9.1.49	defun checkTrim	506
9.1.50	defun checkExtract	507
9.1.51	defun checkFixCommonProblem	508
9.1.52	defun checkDecorate	508
9.1.53	defun hasNoVowels	511
9.1.54	defun checkAddBackSlashes	511
9.1.55	defun checkAddSpaces	512

10 Utility Functions	515
10.0.56 defun translablel	515
10.0.57 defun translablel1	515
10.0.58 defun displayPreCompilationErrors	516
10.0.59 defun bumperrorcount	517
10.0.60 defun parseTranCheckForRecord	517
10.0.61 defun new2OldLisp	518
10.0.62 defun makeSimplePredicateOrNil	518
10.0.63 defun parse-spadstring	518
10.0.64 defun parse-string	519
10.0.65 defun parse-identifier	519
10.0.66 defun parse-number	520
10.0.67 defun parse-keyword	520
10.0.68 defun parse-argument-designator	520
10.0.69 defun print-package	521
10.0.70 defun checkWarning	521
10.0.71 defun tuple2List	521
10.0.72 defmacro pop-stack-1	522
10.0.73 defmacro pop-stack-2	523
10.0.74 defmacro pop-stack-3	523
10.0.75 defmacro pop-stack-4	523
10.0.76 defmacro nth-stack	524
10.0.77 defun Pop-Reduction	524
10.0.78 defun addclose	524
10.0.79 defun blankp	525
10.0.80 defun drop	525
10.0.81 defun escaped	525
10.0.82 defvar \$comblocklist	525
10.0.83 defun fincomblock	526
10.0.84 defun indent-pos	526
10.0.85 defun infixtok	527
10.0.86 defun is-console	527
10.0.87 defun next-tab-loc	527
10.0.88 defun nonblankloc	528
10.0.89 defun parseprint	528
10.0.90 defun skip-to-endif	528
11 The Compiler	529
11.1 Compiling EQ.spad	529
11.1.1 The top level compiler command	532
11.1.2 The Spad compiler top level function	534
11.1.3 defun compilerDoit	538
11.1.4 defun /RQ,LIB	539
11.1.5 defun /rf-1	540
11.1.6 defun spad	549
11.1.7 defun Interpreter interface to the compiler	550

11.1.8 defun print-defun	553
11.1.9 defun def-rename	553
11.1.10 defun def-rename1	554
11.1.11 defun compTopLevel	554
11.1.12 defun compOrCroak	556
11.1.13 defun compOrCroak1	556
11.1.14 defun comp	557
11.1.15 defun compNoStacking	558
11.1.16 defun compNoStacking1	558
11.1.17 defun comp2	559
11.1.18 defun comp3	560
11.1.19 defun applyMapping	562
11.1.20 defun compApply	563
11.1.21 defun compTypeOf	564
11.1.22 defun compColonInside	564
11.1.23 defun compAtom	565
11.1.24 defun compAtomWithModemap	567
11.1.25 defun transImplementation	567
11.1.26 defun convert	568
11.1.27 defun primitiveType	568
11.1.28 defun compSymbol	569
11.1.29 defun compList	570
11.1.30 defun compExpression	571
11.1.31 defun compForm	571
11.1.32 defun compForm1	571
11.1.33 defun compToApply	573
11.1.34 defun compApplication	574
11.1.35 defun getFormModemaps	575
11.1.36 defun eltModemapFilter	577
11.1.37 defun seteltModemapFilter	577
11.1.38 defun compExpressionList	578
11.1.39 defun compForm2	579
11.1.40 defun compForm3	580
11.1.41 defun compFocompFormWithModemap	581
11.1.42 defun substituteIntoFunctorModemap	583
11.1.43 defun compFormPartiallyBottomUp	584
11.1.44 defun compFormMatch	584
11.1.45 defun compUniquely	584
11.1.46 defun compArgumentsAndTryAgain	585
11.1.47 defun compWithMappingMode	586
11.1.48 defun compWithMappingModel	586
11.1.49 defun extractCodeAndConstructTriple	591
11.1.50 defun hasFormalMapVariable	592
11.1.51 defun argsToSig	592
11.1.52 defun compMakeDeclaration	593
11.1.53 defun modifyModeStack	593

11.1.54 defun Create a list of unbound symbols	594
11.1.55 defun compOrCroak1,compactify	595
11.1.56 defun Compiler/Interpreter interface	595
11.1.57 defun compileSpadLispCmd	596
11.1.58 defun recompile-lib-file-if-necessary	597
11.1.59 defun spad-fixed-arg	598
11.1.60 defun compile-lib-file	598
11.1.61 defun compileFileQuietly	598
11.1.62 defvar \$byConstructors	599
11.1.63 defvar \$constructorsSeen	599
12 Level 1	601
12.0.64 defvar \$current-fragment	601
12.0.65 defun read-a-line	601
13 Level 0	603
13.1 Line Handling	603
13.1.1 Line Buffer	603
13.1.2 defstruct \$line	603
13.1.3 defvar \$current-line	604
13.1.4 defmacro line-clear	604
13.1.5 defun line-print	604
13.1.6 defun line-at-end-p	604
13.1.7 defun line-past-end-p	605
13.1.8 defun line-next-char	605
13.1.9 defun line-advance-char	605
13.1.10 defun line-current-segment	606
13.1.11 defun line-new-line	606
13.1.12 defun next-line	606
13.1.13 defun Advance-Char	607
13.1.14 defun storeblanks	607
13.1.15 defun initial-substring	607
13.1.16 defun get-a-line	608
14 The Chunks	609
15 Index	627

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

0.1 Makefile

This book is actually a literate program[2] and contains executable source code. In particular, the Makefile for this book is part of the source of the book and is included below. Axiom uses the “noweb” literate programming system by Norman Ramsey[6].

Chapter 1

Overview

The Spad language is a mathematically oriented language intended for writing computational mathematics. It derives its logical structure from abstract algebra. It features ideas that are still not available in general purpose programming languages, such as selecting overloaded procedures based on the return type as well as the types of the arguments.

The Spad language is heavily influenced by Barbara Liskov's work. It features encapsulation (aka objects), inheritance, and overloading. It has categories which are defined by the exports. Categories are parameterized functors that take arguments which define their behavior.

More details on the language and its high level concepts is available in the Programmers Guide, Volume 3.

The Spad compiler accepts the Spad language and generates a set of files used by the interpreter, detailed in Volume 5.

The compiler does not produce stand-alone executable code. It assumes that it will run inside the interpreter and that the code it generates will be loaded into the interpreter.

Some of the routines are common to both the compiler and the interpreter. Where this happens we have favored the interpreter volume (Volume 5) as the official source location. In each case we will make reference to that volume and the code in it. Thus, the compiler volume should be considered as an extension of the interpreter document.

This volume will go into painful detail of every aspect of compiling Spad code. We will start by defining the input to, and output from the compiler so we know what we are trying to achieve.

Next we will look at the top level data structures used by the compiler. Unfortunately, the compiler uses a large number of "global variables" to pass information and alter control flow. Some of these are used by many routines and some of these are very local to a small subset or a recursion. We will cover the minor ones as they arise.

Next we examine the Pratt parser idea and the Led and Nud concepts, which is used to drive the low level parsing.

Following that we journey deep into the code, trying our best not to get lost in the details. The code is introduced based on “motivation” rather than in strict execution order or related concept order. We do this to try to make the compiler a “readable novel” rather than a mud-march through the code. The goal is to keep the reader’s interest while trying to be exact. Sometimes this will require detours to discuss subtopics.

“Motivating” a piece of software is a not-very-well established form of narrative writing so we assume your forgiveness if we get it wrong. Worse yet, some of the pieces of the system are “legacy”, in that they are no longer used and should be removed. Other parts of the system may have very weak descriptions because we simply do not understand them either. Since this is a living document and the code for the system is actually the code you are reading we will expand parts as we go.

1.1 The Input

```
)abbrev domain EQ Equation
--FOR THE BENEFIT OF LIBAXO GENERATION
++ Author: Stephen M. Watt, enhancements by Johannes Grabmeier
++ Date Created: April 1985
++ Date Last Updated: June 3, 1991; September 2, 1992
++ Basic Operations: =
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ Equations as mathematical objects. All properties of the basis domain,
++ e.g. being an abelian group are carried over the equation domain, by
++ performing the structural operations on the left and on the
++ right hand side.
-- The interpreter translates "=" to "equation". Otherwise, it will
-- find a modemap for "=" in the domain of the arguments.
```

```
Equation(S: Type): public == private where
  Ex ==> OutputForm
  public ==> Type with
    "=": (S, S) -> $
      ++ a=b creates an equation.
    equation: (S, S) -> $
      ++ equation(a,b) creates an equation.
    swap: $ -> $
      ++ swap(eq) interchanges left and right hand side of equation eq.
    lhs: $ -> S
      ++ lhs(eqn) returns the left hand side of equation eqn.
    rhs: $ -> S
      ++ rhs(eqn) returns the right hand side of equation eqn.
```

```

map: (S -> S, $) -> $
  ++ map(f,eqn) constructs a new equation by applying f to both
  ++ sides of eqn.
if S has InnerEvalable(Symbol,S) then
  InnerEvalable(Symbol,S)
if S has SetCategory then
  SetCategory
  CoercibleTo Boolean
  if S has Evalable(S) then
    eval: ($, $) -> $
      ++ eval(eqn, x=f) replaces x by f in equation eqn.
    eval: ($, List $) -> $
      ++ eval(eqn, [x1=v1, ... xn=vn]) replaces xi by vi in equation eqn.
if S has AbelianSemiGroup then
  AbelianSemiGroup
  "+": (S, $) -> $
    ++ x+eqn produces a new equation by adding x to both sides of
    ++ equation eqn.
  "+": ($, S) -> $
    ++ eqn+x produces a new equation by adding x to both sides of
    ++ equation eqn.
if S has AbelianGroup then
  AbelianGroup
  leftZero : $ -> $
    ++ leftZero(eq) subtracts the left hand side.
  rightZero : $ -> $
    ++ rightZero(eq) subtracts the right hand side.
  "-": (S, $) -> $
    ++ x-eqn produces a new equation by subtracting both sides of
    ++ equation eqn from x.
  "-": ($, S) -> $
    ++ eqn-x produces a new equation by subtracting x from
    ++ both sides of equation eqn.
if S has SemiGroup then
  SemiGroup
  "*": (S, $) -> $
    ++ x*eqn produces a new equation by multiplying both sides of
    ++ equation eqn by x.
  "*": ($, S) -> $
    ++ eqn*x produces a new equation by multiplying both sides of
    ++ equation eqn by x.
if S has Monoid then
  Monoid
  leftOne : $ -> Union($,"failed")
    ++ leftOne(eq) divides by the left hand side, if possible.
  rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side, if possible.
if S has Group then
  Group
  leftOne : $ -> Union($,"failed")

```

```

    ++ leftOne(eq) divides by the left hand side.
    rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side.
if S has Ring then
  Ring
  BiModule(S,S)
if S has CommutativeRing then
  Module(S)
  --Algebra(S)
if S has IntegralDomain then
  factorAndSplit : $ -> List $
    ++ factorAndSplit(eq) make the right hand side 0 and
    ++ factors the new left hand side. Each factor is equated
    ++ to 0 and put into the resulting list without repetitions.
if S has PartialDifferentialRing(Symbol) then
  PartialDifferentialRing(Symbol)
if S has Field then
  VectorSpace(S)
  "/" : ($, $) -> $
    ++ e1/e2 produces a new equation by dividing the left and right
    ++ hand sides of equations e1 and e2.
  inv : $ -> $
    ++ inv(x) returns the multiplicative inverse of x.
if S has ExpressionSpace then
  subst : ($, $) -> $
    ++ subst(eq1,eq2) substitutes eq2 into both sides of eq1
    ++ the lhs of eq2 should be a kernel

private ==> add
  Rep := Record(lhs: S, rhs: S)
  eq1,eq2: $
  s : S
  if S has IntegralDomain then
    factorAndSplit eq ==
      (S has factor : S -> Factored S) =>
        eq0 := rightZero eq
        [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
        [eq]
  l:S = r:S      == [1, r]
  equation(l, r) == [1, r]    -- hack! See comment above.
  lhs eqn        == eqn.lhs
  rhs eqn        == eqn.rhs
  swap eqn       == [rhs eqn, lhs eqn]
  map(fn, eqn)   == equation(fn(eqn.lhs), fn(eqn.rhs))

if S has InnerEvalable(Symbol,S) then
  s:Symbol
  ls:List Symbol
  x:S
  lx:List S

```

```

eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x)
eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) = eval(eqn.rhs,ls,lx)
if S has Evaluable(S) then
  eval(eqn1:$, eqn2:$):$ ==
    eval(eqn1.lhs, eqn2 pretend Equation S) =
      eval(eqn1.rhs, eqn2 pretend Equation S)
  eval(eqn1:$, leqn2:List $):$ ==
    eval(eqn1.lhs, leqn2 pretend List Equation S) =
      eval(eqn1.rhs, leqn2 pretend List Equation S)
if S has SetCategory then
  eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and
    (eq1.rhs = eq2.rhs)@Boolean
  coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex
  coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs
if S has AbelianSemiGroup then
  eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs
  s + eq2 == [s,s] + eq2
  eq1 + s == eq1 + [s,s]
if S has AbelianGroup then
  - eq == (- lhs eq) = (-rhs eq)
  s - eq2 == [s,s] - eq2
  eq1 - s == eq1 - [s,s]
  leftZero eq == 0 = rhs eq - lhs eq
  rightZero eq == lhs eq - rhs eq = 0
  0 == equation(0$S,0$S)
  eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs
if S has SemiGroup then
  eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  eqn:$ * 1:S == eqn.lhs * 1 = eqn.rhs * 1
  -- We have to be a bit careful here: raising to a +ve integer is OK
  -- (since it's the equivalent of repeated multiplication)
  -- but other powers may cause contradictions
  -- Watch what else you add here! JHD 2/Aug 1990
if S has Monoid then
  1 == equation(1$S,1$S)
  recip eq ==
    (lh := recip lhs eq) case "failed" => "failed"
    (rh := recip rhs eq) case "failed" => "failed"
    [lh :: S, rh :: S]
  leftOne eq ==
    (re := recip lhs eq) case "failed" => "failed"
    1 = rhs eq * re
  rightOne eq ==
    (re := recip rhs eq) case "failed" => "failed"
    lhs eq * re = 1
if S has Group then
  inv eq == [inv lhs eq, inv rhs eq]
  leftOne eq == 1 = rhs eq * inv rhs eq

```

```

    rightOne eq == lhs eq * inv rhs eq = 1
if S has Ring then
    characteristic() == characteristic()$S
    i:Integer * eq:$ == (i::S) * eq
if S has IntegralDomain then
    factorAndSplit eq ==
        (S has factor : S -> Factored S) =>
            eq0 := rightZero eq
            [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
        (S has Polynomial Integer) =>
            eq0 := rightZero eq
            MF ==> MultivariateFactorize(Symbol, IndexedExponents Symbol, _
                Integer, Polynomial Integer)
            p : Polynomial Integer := (lhs eq0) pretend Polynomial Integer
            [equation((rcf.factor) pretend S,0) for rcf in factors factor(p)$MF]
            [eq]
if S has PartialDifferentialRing(Symbol) then
    differentiate(eq:$, sym:Symbol):$ ==
        [differentiate(lhs eq, sym), differentiate(rhs eq, sym)]
if S has Field then
    dimension() == 2 :: CardinalNumber
    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs
    inv eq == [inv lhs eq, inv rhs eq]
if S has ExpressionSpace then
    subst(eq1,eq2) ==
        eq3 := eq2 pretend Equation S
        [subst(lhs eq1,eq3),subst(rhs eq1,eq3)]

```

1.2 The Output, the EQ.nrlib directory

The Spad compiler generates several files in a directory named after the input abbreviation. The input file contains an abbreviation line:

```
)abbrev domain EQ Equation
```

for each category, domain, or package. The abbreviation line has 3 parts.

- one of “category”, “domain”, or “package”
- the abbreviation for this domain (8 Uppercase Characters maximum)
- the name of this domain

Since the abbreviation for the Equation domain is EQ, the compiler will put all of its output into a subdirectory called “EQ.nrlib”. The “nrlib” is a port of a very old VMLisp file format, simulated with directories.

For the EQ input file, the compiler will create the following output files, each of which we will explain in detail below.

```
/research/test/int/algebra/EQ.nrlib:
used 216 available 4992900
drwxr-xr-x    2 root root  4096 2010-12-09 11:20 .
drwxr-xr-x 1259 root root 73728 2010-12-09 11:43 ..
-rw-r--r--    1 root root 19228 2010-12-09 11:20 code.lsp
-rw-r--r--    1 root root 34074 2010-12-09 11:20 code.o
-rw-r--r--    1 root root 13543 2010-12-09 11:20 EQ.fn
-rw-r--r--    1 root root 19228 2010-12-09 11:20 EQ.lsp
-rw-r--r--    1 root root 36148 2010-12-09 11:20 index.kaf
-rw-r--r--    1 root root  6236 2010-12-09 11:20 info
```

1.3 The code.lsp and EQ.lsp files

```
(/VERSIONCHECK 2)
```

```
(DEFUN |EQ;factorAndSplit;$L;1| (|eq| $)
  (PROG (|eq0| #:G1403 |rcf| #:G1404)
    (RETURN
      (SEQ (COND
        ((|HasSignature| (QREFELT $ 6)
          (LIST '|factor|
            (LIST (LIST '|Factored|
              (|devaluate| (QREFELT $ 6)))
              (|devaluate| (QREFELT $ 6))))))
          (SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
            |EQ;factorAndSplit;$L;1|)
            (EXIT (PROGN
              (LETT #:G1403 NIL |EQ;factorAndSplit;$L;1|)
              (SEQ (LETT |rcf| NIL
                |EQ;factorAndSplit;$L;1|)
                (LETT #:G1404
                  (SPADCALL
                    (SPADCALL
                      (SPADCALL |eq0| (QREFELT $ 9))
                      (QREFELT $ 11))
                      (QREFELT $ 15))
                    |EQ;factorAndSplit;$L;1|)
                  G190
                  (COND
                    ((OR (ATOM #:G1404)
                      (PROGN
                        (LETT |rcf| (CAR #:G1404)
                          |EQ;factorAndSplit;$L;1|)
                          NIL))
                      (GO G191))))
                (GO G191))))
              (GO G191))))
          (GO G191))))
      (GO G191))))
```

```

(SEQ (EXIT
      (LETT #:G1403
            (CONS
              (SPADCALL (QCAR |rcf|)
                        (|spadConstant| $ 16)
                        (QREFELT $ 17))
              #:G1403)
      |EQ;factorAndSplit;$L;1|)))
(LETT #:G1404 (CDR #:G1404)
      |EQ;factorAndSplit;$L;1|)
(GO G190) G191
(EXIT (NREVERSEO #:G1403))))))
('T (LIST |eq|))))))

(PUT (QUOTE |EQ;=;2S$;2|) (QUOTE |SPADreplace|) (QUOTE CONS))

(DEFUN |EQ;=;2S$;2| (|l| |r| $) (CONS |l| |r|))

(PUT (QUOTE |EQ;equation;2S$;3|) (QUOTE |SPADreplace|) (QUOTE CONS))

(DEFUN |EQ;equation;2S$;3| (|l| |r| $) (CONS |l| |r|))

(PUT (QUOTE |EQ;lhs;$S;4|) (QUOTE |SPADreplace|) (QUOTE QCAR))

(DEFUN |EQ;lhs;$S;4| (|eqn| $) (QCAR |eqn|))

(PUT (QUOTE |EQ;rhs;$S;5|) (QUOTE |SPADreplace|) (QUOTE QCDR))

(DEFUN |EQ;rhs;$S;5| (|eqn| $) (QCDR |eqn|))

(DEFUN |EQ;swap;2$;6| (|eqn| $) (CONS (SPADCALL |eqn| (QREFELT $ 21))
  (SPADCALL |eqn| (QREFELT $ 9))))

(DEFUN |EQ;map;M2$;7| (|fn| |eqn| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |fn|)
    (SPADCALL (QCDR |eqn|) |fn|)
    (QREFELT $ 17)))

(DEFUN |EQ;eval;$SS$;8| (|eqn| |s| |x| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |s| |x| (QREFELT $ 26))
    (SPADCALL (QCDR |eqn|) |s| |x| (QREFELT $ 26))
    (QREFELT $ 20)))

(DEFUN |EQ;eval;$LL$;9| (|eqn| |ls| |lx| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |ls| |lx| (QREFELT $ 30))
    (SPADCALL (QCDR |eqn|) |ls| |lx| (QREFELT $ 30))
    (QREFELT $ 20)))

```

```

(DEFUN |EQ;eval;3$;10| (|eqn1| |eqn2| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn1|) |eqn2| (QREFELT $ 33))
    (SPADCALL (QCDR |eqn1|) |eqn2| (QREFELT $ 33))
    (QREFELT $ 20)))

(DEFUN |EQ;eval;$L$;11| (|eqn1| |leqn2| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn1|) |leqn2| (QREFELT $ 36))
    (SPADCALL (QCDR |eqn1|) |leqn2| (QREFELT $ 36))
    (QREFELT $ 20)))

(DEFUN |EQ;=;2$B;12| (|eq1| |eq2| $)
  (COND
    ((SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 39))
      (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 39)))
    ((QUOTE T) (QUOTE NIL))))

(DEFUN |EQ;coerce;$Of;13| (|eqn| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) (QREFELT $ 42))
    (SPADCALL (QCDR |eqn|) (QREFELT $ 42))
    (QREFELT $ 43)))

(DEFUN |EQ;coerce;$B;14| (|eqn| $)
  (SPADCALL (QCAR |eqn|) (QCDR |eqn|) (QREFELT $ 39)))

(DEFUN |EQ;+;3$;15| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 46))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 46))
    (QREFELT $ 20)))

(DEFUN |EQ;+;S2$;16| (|s| |eq2| $)
  (SPADCALL (CONS |s| |s|) |eq2| (QREFELT $ 47)))

(DEFUN |EQ;+;$S$;17| (|eq1| |s| $)
  (SPADCALL |eq1| (CONS |s| |s|) (QREFELT $ 47)))

(DEFUN |EQ;-;2$;18| (|eq| $)
  (SPADCALL
    (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 50))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 50))
    (QREFELT $ 20)))

(DEFUN |EQ;-;S2$;19| (|s| |eq2| $)
  (SPADCALL (CONS |s| |s|) |eq2| (QREFELT $ 52)))

(DEFUN |EQ;-;$S$;20| (|eq1| |s| $)

```

```

(SPADCALL |eq1| (CONS |s| |s|) (QREFELT $ 52)))

(DEFUN |EQ;leftZero;2$;21| (|eq| $)
  (SPADCALL
    (|spadConstant| $ 16)
    (SPADCALL
      (SPADCALL |eq| (QREFELT $ 21))
      (SPADCALL |eq| (QREFELT $ 9))
      (QREFELT $ 56))
    (QREFELT $ 20)))

(DEFUN |EQ;rightZero;2$;22| (|eq| $)
  (SPADCALL
    (SPADCALL
      (SPADCALL |eq| (QREFELT $ 9))
      (SPADCALL |eq| (QREFELT $ 21))
      (QREFELT $ 56))
    (|spadConstant| $ 16)
    (QREFELT $ 20)))

(DEFUN |EQ;Zero;$;23| ($)
  (SPADCALL (|spadConstant| $ 16) (|spadConstant| $ 16) (QREFELT $ 17)))

(DEFUN |EQ;-;3$;24| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 56))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 56))
    (QREFELT $ 20)))

(DEFUN |EQ;*;3$;25| (|eq1| |eq2| $)
  (SPADCALL
    (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 58))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;S2$;26| (|l| |eqn| $)
  (SPADCALL
    (SPADCALL |l| (QCAR |eqn|) (QREFELT $ 58))
    (SPADCALL |l| (QCDR |eqn|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;S2$;27| (|l| |eqn| $)
  (SPADCALL
    (SPADCALL |l| (QCAR |eqn|) (QREFELT $ 58))
    (SPADCALL |l| (QCDR |eqn|) (QREFELT $ 58))
    (QREFELT $ 20)))

(DEFUN |EQ;*;$S$;28| (|eqn| |l| $)
  (SPADCALL
    (SPADCALL (QCAR |eqn|) |l| (QREFELT $ 58))

```

```

(SPADCALL (QCDR |eqn|) |l| (QREFELT $ 58))
(QREFELT $ 20)))

(DEFUN |EQ;One;$;29| ($)
  (SPADCALL (|spadConstant| $ 62) (|spadConstant| $ 62) (QREFELT $ 17)))

(DEFUN |EQ;recip;$U;30| (|eq| $)
  (PROG (|lh| |rh|)
    (RETURN
      (SEQ
        (LETT |lh|
          (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 65))
          |EQ;recip;$U;30|)
        (EXIT
          (COND
            ((QEQCAR |lh| 1) (CONS 1 "failed")))
            ('T
              (SEQ
                (LETT |rh|
                  (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 65))
                  |EQ;recip;$U;30|)
                (EXIT
                  (COND
                    ((QEQCAR |rh| 1) (CONS 1 "failed")))
                    ('T
                      (CONS 0
                        (CONS (QCDR |lh|) (QCDR |rh|))))))))))))))

(DEFUN |EQ;leftOne;$U;31| (|eq| $)
  (PROG (|re|)
    (RETURN
      (SEQ
        (LETT |re|
          (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 65))
          |EQ;leftOne;$U;31|)
        (EXIT
          (COND
            ((QEQCAR |re| 1) (CONS 1 "failed")))
            ('T
              (CONS 0
                (SPADCALL
                  (|spadConstant| $ 62)
                  (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QCDR |re|) (QREFELT $ 58))
                  (QREFELT $ 20))))))))))

(DEFUN |EQ;rightOne;$U;32| (|eq| $)
  (PROG (|re|)
    (RETURN

```

```

(SEQ
  (LETT |re|
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 65))
    |EQ;rightOne;$U;32|)
  (EXIT
    (COND
      ((QEQCAR |re| 1) (CONS 1 "failed"))
      ('T
        (CONS 0
          (SPADCALL
            (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QCDR |re|) (QREFELT $ 58))
            (|spadConstant| $ 62)
            (QREFELT $ 20))))))))))

(DEFUN |EQ;inv;2$;33| (|eq| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 69))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 69))))

(DEFUN |EQ;leftOne;$U;34| (|eq| $)
  (CONS 0
    (SPADCALL (|spadConstant| $ 62)
      (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
        (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
          (QREFELT $ 69))
          (QREFELT $ 58))
        (QREFELT $ 20))))))

(DEFUN |EQ;rightOne;$U;35| (|eq| $)
  (CONS 0
    (SPADCALL
      (SPADCALL (SPADCALL |eq| (QREFELT $ 9))
        (SPADCALL (SPADCALL |eq| (QREFELT $ 21))
          (QREFELT $ 69))
          (QREFELT $ 58))
        (|spadConstant| $ 62) (QREFELT $ 20))))))

(DEFUN |EQ;characteristic;Nni;36| ($) (SPADCALL (QREFELT $ 72)))

(DEFUN |EQ;*;I2$;37| (|i| |eq| $)
  (SPADCALL (SPADCALL |i| (QREFELT $ 75)) |eq| (QREFELT $ 60)))

(DEFUN |EQ;factorAndSplit;$L;38| (|eq| $)
  (PROG (#:G1488 #:G1489 |eq0| |p| #:G1490 |rcf| #:G1491)
    (RETURN
      (SEQ (COND
        ((|HasSignature| (QREFELT $ 6)
          (LIST ' |factor|
            (LIST (LIST ' |Factored|
              (|devaluate| (QREFELT $ 6))))

```

```

(|devaluate| (QREFELT $ 6))))))
(SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
      |EQ;factorAndSplit;$L;38|)
(EXIT (PROGN
      (LETT #:G1488 NIL |EQ;factorAndSplit;$L;38|)
      (SEQ (LETT |rcf| NIL
                |EQ;factorAndSplit;$L;38|)
            (LETT #:G1489
                  (SPADCALL
                    (SPADCALL
                      (SPADCALL |eq0| (QREFELT $ 9))
                      (QREFELT $ 11))
                      (QREFELT $ 15))
                    |EQ;factorAndSplit;$L;38|)
            G190
            (COND
              ((OR (ATOM #:G1489)
                   (PROGN
                     (LETT |rcf| (CAR #:G1489)
                           |EQ;factorAndSplit;$L;38|)
                     NIL))
               (GO G191))))
      (SEQ (EXIT
            (LETT #:G1488
                  (CONS
                    (SPADCALL (QCAR |rcf|)
                              (|spadConstant| $ 16)
                              (QREFELT $ 17))
                    #:G1488)
                    |EQ;factorAndSplit;$L;38|)))
            (LETT #:G1489 (CDR #:G1489)
                  |EQ;factorAndSplit;$L;38|)
            (GO G190) G191
            (EXIT (NREVERSEO #:G1488))))))
((EQUAL (QREFELT $ 6) (|Polynomial| (|Integer|)))
 (SEQ (LETT |eq0| (SPADCALL |eq| (QREFELT $ 8))
      |EQ;factorAndSplit;$L;38|)
      (LETT |p| (SPADCALL |eq0| (QREFELT $ 9))
            |EQ;factorAndSplit;$L;38|)
      (EXIT (PROGN
            (LETT #:G1490 NIL |EQ;factorAndSplit;$L;38|)
            (SEQ (LETT |rcf| NIL
                      |EQ;factorAndSplit;$L;38|)
                  (LETT #:G1491
                        (SPADCALL
                          (SPADCALL |p| (QREFELT $ 80))
                          (QREFELT $ 83))
                        |EQ;factorAndSplit;$L;38|)
                  G190
                  (COND

```

```

((OR (ATOM #:G1491)
  (PROGN
    (LETT |rcf| (CAR #:G1491)
      |EQ;factorAndSplit;$L;38|)
      NIL))
  (GO G191)))
(SEQ (EXIT
  (LETT #:G1490
    (CONS
      (SPADCALL (QCAR |rcf|)
        (|spadConstant| $ 16)
        (QREFELT $ 17))
      #:G1490)
    |EQ;factorAndSplit;$L;38|)))
  (LETT #:G1491 (CDR #:G1491)
    |EQ;factorAndSplit;$L;38|)
  (GO G190) G191
  (EXIT (NREVERSEO #:G1490))))))
('T (LIST |eq|))))))

(DEFUN |EQ;differentiate;$S$;39| (|eq| |sym| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) |sym| (QREFELT $ 84))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) |sym| (QREFELT $ 84))))

(DEFUN |EQ;dimension;Cn;40| ($) (SPADCALL 2 (QREFELT $ 87)))

(DEFUN |EQ;/;3$;41| (|eq1| |eq2| $)
  (SPADCALL (SPADCALL (QCAR |eq1|) (QCAR |eq2|) (QREFELT $ 89))
    (SPADCALL (QCDR |eq1|) (QCDR |eq2|) (QREFELT $ 89))
    (QREFELT $ 20)))

(DEFUN |EQ;inv;2$;42| (|eq| $)
  (CONS (SPADCALL (SPADCALL |eq| (QREFELT $ 9)) (QREFELT $ 69))
    (SPADCALL (SPADCALL |eq| (QREFELT $ 21)) (QREFELT $ 69))))

(DEFUN |EQ;subst;3$;43| (|eq1| |eq2| $)
  (PROG (|eq3|)
    (RETURN
      (SEQ (LETT |eq3| |eq2| |EQ;subst;3$;43|)
        (EXIT (CONS (SPADCALL (SPADCALL |eq1| (QREFELT $ 9)) |eq3|
          (QREFELT $ 92))
          (SPADCALL (SPADCALL |eq1| (QREFELT $ 21)) |eq3|
            (QREFELT $ 92)))))))

(DEFUN |Equation| (#:G1503)
  (PROG ()
    (RETURN
      (PROG (#:G1504)
        (RETURN

```

```

(COND
  ((LETT #:G1504
    (|lassocShiftWithFunction|
      (LIST (|devaluate| #:G1503))
      (HGET |$ConstructorCache| '|Equation|)
      '|domainEqualList|)
    |Equation|)
  (|CDRwithIncrement| #:G1504))
('T
  (UNWIND-PROTECT
    (PROG1 (|Equation;| #:G1503)
      (LETT #:G1504 T |Equation|))
    (COND
      ((NOT #:G1504) (HREM |$ConstructorCache| '|Equation|)))))))))

(DEFUN |Equation;| (|#1|)
  (PROG (DV$1 |dv$| $ #:G1502 #:G1501 #:G1500 #:G1499 #:G1498 |pv$|)
    (RETURN
      (PROGN
        (LETT DV$1 (|devaluate| |#1|) |Equation|)
        (LETT |dv$| (LIST '|Equation| DV$1) |Equation|)
        (LETT $ (make-array 98) |Equation|)
        (QSETREFV $ 0 |dv$|)
        (QSETREFV $ 3
          (LETT |pv$|
            (|buildPredVector| 0 0
              (LIST (|HasCategory| |#1| '|(Field|)')
                (|HasCategory| |#1| '|(SetCategory|)')
                (|HasCategory| |#1| '|(Ring|)')
                (|HasCategory| |#1|
                  '|(PartialDifferentialRing| (|Symbol|))')
                (OR (|HasCategory| |#1|
                  '|(PartialDifferentialRing|
                    (|Symbol|))')
                  (|HasCategory| |#1| '|(Ring|)'))
                (|HasCategory| |#1| '|(Group|)')
                (|HasCategory| |#1|
                  (LIST '|InnerEvalable| '|(Symbol|)
                    (|devaluate| |#1|))')
                (AND (|HasCategory| |#1|
                  (LIST '|Evalable|
                    (|devaluate| |#1|))')
                  (|HasCategory| |#1| '|(SetCategory|)'))
                (|HasCategory| |#1| '|(IntegralDomain|)')
                (|HasCategory| |#1| '|(ExpressionSpace|)')
                (OR (|HasCategory| |#1| '|(Field|)')
                  (|HasCategory| |#1| '|(Group|)'))
                (OR (|HasCategory| |#1| '|(Group|)')
                  (|HasCategory| |#1| '|(Ring|)'))
                (LETT #:G1502

```

```

(|HasCategory| |#1|
  '(|CommutativeRing|))
|Equation|)
(OR #:G1502 (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR #:G1502
  (|HasCategory| |#1| '(|Field|)))
(LETT #:G1501
  (|HasCategory| |#1| '(|Monoid|))
  |Equation|)
(OR (|HasCategory| |#1| '(|Group|))
  #:G1501)
(LETT #:G1500
  (|HasCategory| |#1| '(|SemiGroup|))
  |Equation|)
(OR (|HasCategory| |#1| '(|Group|)) #:G1501
  #:G1500)
(LETT #:G1499
  (|HasCategory| |#1|
    '(|AbelianGroup|))
  |Equation|)
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR #:G1499 #:G1501)
(LETT #:G1498
  (|HasCategory| |#1|
    '(|AbelianSemiGroup|))
  |Equation|)
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1498 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Ring|)))
(OR (|HasCategory| |#1|
  '(|PartialDifferentialRing|
    (|Symbol|)))
  #:G1499 #:G1498 #:G1502
  (|HasCategory| |#1| '(|Field|))
  (|HasCategory| |#1| '(|Group|)) #:G1501
  (|HasCategory| |#1| '(|Ring|)) #:G1500
  (|HasCategory| |#1| '(|SetCategory|))))
|Equation|))
(|haddProp| |$ConstructorCache| '|Equation| (LIST DV$1)
  (CONS 1 $))
(|stuffDomainSlots| $)

```

```

(QSETREFV $ 6 |#1|)
(QSETREFV $ 7 (|Record| (|:| |lhs| |#1|) (|:| |rhs| |#1|)))
(COND
  ((|testBitVector| |pv$| 9)
    (QSETREFV $ 19
      (CONS (|dispatchFunction| |EQ;factorAndSplit;$L;1|) $))))
(COND
  ((|testBitVector| |pv$| 7)
    (PROGN
      (QSETREFV $ 27
        (CONS (|dispatchFunction| |EQ;eval;$SS$;8|) $))
      (QSETREFV $ 31
        (CONS (|dispatchFunction| |EQ;eval;$LL$;9|) $))))))
(COND
  ((|HasCategory| |#1| (LIST ' |Evalable| (|devaluate| |#1|)))
    (PROGN
      (QSETREFV $ 34
        (CONS (|dispatchFunction| |EQ;eval;3$;10|) $))
      (QSETREFV $ 37
        (CONS (|dispatchFunction| |EQ;eval;$L$;11|) $))))))
(COND
  ((|testBitVector| |pv$| 2)
    (PROGN
      (QSETREFV $ 40
        (CONS (|dispatchFunction| |EQ;=;2$B;12|) $))
      (QSETREFV $ 44
        (CONS (|dispatchFunction| |EQ;coerce;$0f;13|) $))
      (QSETREFV $ 45
        (CONS (|dispatchFunction| |EQ;coerce;$B;14|) $))))))
(COND
  ((|testBitVector| |pv$| 23)
    (PROGN
      (QSETREFV $ 47 (CONS (|dispatchFunction| |EQ;+;3$;15|) $))
      (QSETREFV $ 48
        (CONS (|dispatchFunction| |EQ;+;S2$;16|) $))
      (QSETREFV $ 49
        (CONS (|dispatchFunction| |EQ;+;$S$;17|) $))))))
(COND
  ((|testBitVector| |pv$| 20)
    (PROGN
      (QSETREFV $ 51 (CONS (|dispatchFunction| |EQ;-;2$;18|) $))
      (QSETREFV $ 53
        (CONS (|dispatchFunction| |EQ;-;S2$;19|) $))
      (QSETREFV $ 54
        (CONS (|dispatchFunction| |EQ;-;$S$;20|) $))
      (QSETREFV $ 57
        (CONS (|dispatchFunction| |EQ;leftZero;2$;21|) $))
      (QSETREFV $ 8
        (CONS (|dispatchFunction| |EQ;rightZero;2$;22|) $))
      (QSETREFV $ 55

```

```

(CONS IDENTITY
  (FUNCALL (|dispatchFunction| |EQ;Zero;$;23|) $))
(QSETREFV $ 52 (CONS (|dispatchFunction| |EQ;-;3$;24|) $))))
(COND
  ((|testBitVector| |pv$| 18)
    (PROGN
      (QSETREFV $ 59 (CONS (|dispatchFunction| |EQ;*;3$;25|) $))
      (QSETREFV $ 60
        (CONS (|dispatchFunction| |EQ;*;S2$;26|) $))
      (QSETREFV $ 60
        (CONS (|dispatchFunction| |EQ;*;S2$;27|) $))
      (QSETREFV $ 61
        (CONS (|dispatchFunction| |EQ;*;S$;28|) $))))))
(COND
  ((|testBitVector| |pv$| 16)
    (PROGN
      (QSETREFV $ 63
        (CONS IDENTITY
          (FUNCALL (|dispatchFunction| |EQ;One;$;29|) $)))
      (QSETREFV $ 66
        (CONS (|dispatchFunction| |EQ;recip;$U;30|) $))
      (QSETREFV $ 67
        (CONS (|dispatchFunction| |EQ;leftOne;$U;31|) $))
      (QSETREFV $ 68
        (CONS (|dispatchFunction| |EQ;rightOne;$U;32|) $))))))
(COND
  ((|testBitVector| |pv$| 6)
    (PROGN
      (QSETREFV $ 70
        (CONS (|dispatchFunction| |EQ;inv;2$;33|) $))
      (QSETREFV $ 67
        (CONS (|dispatchFunction| |EQ;leftOne;$U;34|) $))
      (QSETREFV $ 68
        (CONS (|dispatchFunction| |EQ;rightOne;$U;35|) $))))))
(COND
  ((|testBitVector| |pv$| 3)
    (PROGN
      (QSETREFV $ 73
        (CONS (|dispatchFunction| |EQ;characteristic;Nni;36|)
          $))
      (QSETREFV $ 76
        (CONS (|dispatchFunction| |EQ;*;I2$;37|) $))))))
(COND
  ((|testBitVector| |pv$| 9)
    (QSETREFV $ 19
      (CONS (|dispatchFunction| |EQ;factorAndSplit;$L;38|) $))))
(COND
  ((|testBitVector| |pv$| 4)
    (QSETREFV $ 85
      (CONS (|dispatchFunction| |EQ;differentiate;S$;39|) $))))

```

```

(COND
  ((|testBitVector| |pv$| 1)
   (PROGN
    (QSETREFV $ 88
     (CONS (|dispatchFunction| |EQ;dimension;Cn;40|) $))
    (QSETREFV $ 90 (CONS (|dispatchFunction| |EQ;/;3$;41|) $))
    (QSETREFV $ 70
     (CONS (|dispatchFunction| |EQ;inv;2$;42|) $))))))
(COND
  ((|testBitVector| |pv$| 10)
   (QSETREFV $ 93
    (CONS (|dispatchFunction| |EQ;subst;3$;43|) $))))
$))))

(setf (get 'Equation| 'infovec|)
  (LIST '#(NIL NIL NIL NIL NIL NIL (|local| |#1|) 'Rep|
    (0 . |rightZero|) |EQ;lhs;$S;4| (|Factored| $)
    (5 . |factor|)
    (|Record| (|:| |factor| 6) (|:| |exponent| 74))
    (|List| 12) (|Factored| 6) (10 . |factors|) (15 . |Zero|)
    |EQ;equation;2S$;3| (|List| $) (19 . |factorAndSplit|)
    |EQ;=;2S$;2| |EQ;rhs;$S;5| |EQ;swap;2$;6| (|Mapping| 6 6)
    |EQ;map;M2$;7| (|Symbol|) (24 . |eval|) (31 . |eval|)
    (|List| 25) (|List| 6) (38 . |eval|) (45 . |eval|)
    (|Equation| 6) (52 . |eval|) (58 . |eval|) (|List| 32)
    (64 . |eval|) (70 . |eval|) (|Boolean|) (76 . =) (82 . =)
    (|OutputForm|) (88 . |coerce|) (93 . =) (99 . |coerce|)
    (104 . |coerce|) (109 . +) (115 . +) (121 . +) (127 . +)
    (133 . -) (138 . -) (143 . -) (149 . -) (155 . -)
    (161 . |Zero|) (165 . -) (171 . |leftZero|) (176 . *)
    (182 . *) (188 . *) (194 . *) (200 . |One|) (204 . |One|)
    (|Union| $ "failed") (208 . |recip|) (213 . |recip|)
    (218 . |leftOne|) (223 . |rightOne|) (228 . |inv|)
    (233 . |inv|) (|NonNegativeInteger|)
    (238 . |characteristic|) (242 . |characteristic|)
    (|Integer|) (246 . |coerce|) (251 . *) (|Factored| 78)
    (|Polynomial| 74)
    (|MultivariateFactorize| 25 (|IndexedExponents| 25) 74 78)
    (257 . |factor|)
    (|Record| (|:| |factor| 78) (|:| |exponent| 74))
    (|List| 81) (262 . |factors|) (267 . |differentiate|)
    (273 . |differentiate|) (|CardinalNumber|)
    (279 . |coerce|) (284 . |dimension|) (288 . /) (294 . /)
    (|Equation| $) (300 . |subst|) (306 . |subst|)
    (|PositiveInteger|) (|List| 71) (|SingleInteger|)
    (|String|))
  '#(~= 312 |zero?| 318 |swap| 323 |subtractIfCan| 328 |subst|
    334 |sample| 340 |rightZero| 344 |rightOne| 349 |rhs| 354
    |recip| 359 |one?| 364 |map| 369 |lhs| 375 |leftZero| 380
    |leftOne| 385 |latex| 390 |inv| 395 |hash| 400

```

```

|factorAndSplit| 405 |eval| 410 |equation| 436 |dimension|
442 |differentiate| 446 |conjugate| 472 |commutator| 478
|coerce| 484 |characteristic| 499 ^ 503 |Zero| 521 |One|
525 D 529 = 555 / 567 - 579 + 602 ** 620 * 638)
'((|unitsKnown| . 12) (|rightUnitary| . 3)
(|leftUnitary| . 3))
(CONS (|makeByteWordVec2| 25
      '(1 15 4 14 5 14 3 5 3 21 21 6 21 17 24 19 25 0 2
        25 2 7))
(CONS '#(|VectorSpace&| |Module&|
        |PartialDifferentialRing&| NIL |Ring&| NIL NIL
        NIL NIL |AbelianGroup&| NIL |Group&|
        |AbelianMonoid&| |Monoid&| |AbelianSemiGroup&|
        |SemiGroup&| |SetCategory&| NIL NIL
        |BasicType&| NIL |InnerEvalable&|)
(CONS '#(|VectorSpace| 6) (|Module| 6)
      (|PartialDifferentialRing| 25)
      (|BiModule| 6 6) (|Ring|)
      (|LeftModule| 6) (|RightModule| 6)
      (|Rng|) (|LeftModule| $$)
      (|AbelianGroup|)
      (|CancellationAbelianMonoid|) (|Group|)
      (|AbelianMonoid|) (|Monoid|)
      (|AbelianSemiGroup|) (|SemiGroup|)
      (|SetCategory|) (|Type|)
      (|CoercibleTo| 41) (|BasicType|)
      (|CoercibleTo| 38)
      (|InnerEvalable| 25 6))
(|makeByteWordVec2| 97
  '(1 0 0 0 8 1 6 10 0 11 1 14 13 0 15 0
    6 0 16 1 0 18 0 19 3 6 0 0 25 6 26 3
    0 0 0 25 6 27 3 6 0 0 28 29 30 3 0 0
    0 28 29 31 2 6 0 0 32 33 2 0 0 0 0 34
    2 6 0 0 35 36 2 0 0 0 18 37 2 6 38 0
    0 39 2 0 38 0 0 40 1 6 41 0 42 2 41 0
    0 0 43 1 0 41 0 44 1 0 38 0 45 2 6 0
    0 0 46 2 0 0 0 0 47 2 0 0 6 0 48 2 0
    0 0 6 49 1 6 0 0 50 1 0 0 0 51 2 0 0
    0 0 52 2 0 0 6 0 53 2 0 0 0 6 54 0 0
    0 55 2 6 0 0 0 56 1 0 0 0 57 2 6 0 0
    0 58 2 0 0 0 0 59 2 0 0 6 0 60 2 0 0
    0 6 61 0 6 0 62 0 0 0 63 1 6 64 0 65
    1 0 64 0 66 1 0 64 0 67 1 0 64 0 68 1
    6 0 0 69 1 0 0 0 70 0 6 71 72 0 0 71
    73 1 6 0 74 75 2 0 0 74 0 76 1 79 77
    78 80 1 77 82 0 83 2 6 0 0 25 84 2 0
    0 0 25 85 1 86 0 71 87 0 0 86 88 2 6
    0 0 0 89 2 0 0 0 0 90 2 6 0 0 91 92 2
    0 0 0 0 93 2 2 38 0 0 1 1 20 38 0 1 1
    0 0 0 22 2 20 64 0 0 1 2 10 0 0 0 93

```

```

0 22 0 1 1 20 0 0 8 1 16 64 0 68 1 0
6 0 21 1 16 64 0 66 1 16 38 0 1 2 0 0
23 0 24 1 0 6 0 9 1 20 0 0 57 1 16 64
0 67 1 2 97 0 1 1 11 0 0 70 1 2 96 0
1 1 9 18 0 19 2 8 0 0 0 34 2 8 0 0 18
37 3 7 0 0 25 6 27 3 7 0 0 28 29 31 2
0 0 6 6 17 0 1 86 88 2 4 0 0 28 1 2 4
0 0 25 85 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 6 0 0 0 1 2 6 0 0 0 1 1 3 0 74
1 1 2 41 0 44 1 2 38 0 45 0 3 71 73 2
6 0 0 74 1 2 16 0 0 71 1 2 18 0 0 94
1 0 20 0 55 0 16 0 63 2 4 0 0 28 1 2
4 0 0 25 1 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 2 38 0 0 40 2 0 0 6 6 20 2 11
0 0 0 90 2 1 0 0 6 1 1 20 0 0 51 2 20
0 0 0 52 2 20 0 6 0 53 2 20 0 0 6 54
2 23 0 0 0 47 2 23 0 6 0 48 2 23 0 0
6 49 2 6 0 0 74 1 2 16 0 0 71 1 2 18
0 0 94 1 2 20 0 71 0 1 2 20 0 74 0 76
2 23 0 94 0 1 2 18 0 0 0 59 2 18 0 0
6 61 2 18 0 6 0 60))))))
'|lookupComplete|))

```

1.4 The code.o file

The Spad compiler translates the Spad language into Common Lisp. It eventually invokes the Common Lisp “compile-file” command to output files in binary. Depending on the lisp system this filename can vary (e.g “code.fasl”). The details of how these are used depends on the Common Lisp in use.

By default, Axiom uses Gnu Common Lisp (GCL), which generates “.o” files.

1.5 The info file

```

(((* (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (* S S S))
($ (= $ S S)))
(($ $ $ S) (|arguments| (|l| . S) (|eqn| . $)) (S (* S S S))
($ (= $ S S)))
(($ #0=(|Integer|) $) (|arguments| (|i| . #0#) (|eq| . $))
(S (|coerce| S (|Integer|))) ($ (* $ $ S $)))
(($ S $) (|arguments| (|l| . S) (|eqn| . $)) (S (* S S S))
($ (= $ S S)))
(+ (($ $ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (+ S S S))
($ (= $ S S)))
(($ $ $ S) (|arguments| (|s| . S) (|eq1| . $)) ($ (+ $ $ $ $)))
(($ S $) (|arguments| (|s| . S) (|eq2| . $)) ($ (+ $ $ $ $))))

```

```

(- (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (- S S S))
  ($ (= $ S S)))
  (($ $ $) (|arguments| (|s| . S) (|eq1| . $)) ($ (- $ $ $)))
  (($ $) (|arguments| (|eq| . $)) (S (- S S))
    ($ (|rhs| S $) (|lhs| S $) (= $ S S)))
  (($ S $) (|arguments| (|s| . S) (|eq2| . $)) ($ (- $ $ $)))
(/ (($ $ $) (|arguments| (|eq2| . $) (|eq1| . $)) (S (/ S S S))
  ($ (= $ S S)))
(= (($ S S) (|arguments| (|r| . S) (|l| . S)))
  (((|Boolean|) $) ((|Boolean|) (|false| (|Boolean|))))
  (|locals| (#:G1393 |Boolean|))
  (|arguments| (|eq2| . $) (|eq1| . $)) (S (= (|Boolean|) S S)))
(|One| (($) (S (|One| S)) ($ (|equation| $ S S))))
(|Zero| (($) (S (|Zero| S)) ($ (|equation| $ S S))))
(|characteristic|
  (((|NonNegativeInteger|))
  (S (|characteristic| (|NonNegativeInteger|)))))
(|coerce|
  (((|Boolean|) $) (|arguments| (|eqn| . $))
  (S (= (|Boolean|) S S)))
  (((|OutputForm|) $)
  ((|OutputForm|) (= (|OutputForm|) (|OutputForm|) (|OutputForm|)))
  (|arguments| (|eqn| . $)) (S (|coerce| (|OutputForm|) S))))
(|constructor|
  (NIL (|locals|
    (|Rep| |Join| (|SetCategory|)
    (CATEGORY |domain|
      (SIGNATURE |construct|
        ((|Record| (|:| |lhs| S) (|:| |rhs| S)) S
        S))
      (SIGNATURE |coerce|
        ((|OutputForm|)
        (|Record| (|:| |lhs| S) (|:| |rhs| S))))
      (SIGNATURE |elt|
        (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
        "lhs"))
      (SIGNATURE |elt|
        (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
        "rhs"))
      (SIGNATURE |setelt|
        (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
        "lhs" S))
      (SIGNATURE |setelt|
        (S (|Record| (|:| |lhs| S) (|:| |rhs| S))
        "rhs" S))
      (SIGNATURE |copy|
        ((|Record| (|:| |lhs| S) (|:| |rhs| S))
        (|Record| (|:| |lhs| S) (|:| |rhs| S)))))))))
(|differentiate|
  (($ $ #1=(|Symbol|)) (|arguments| (|sym| . #1#) (|eq| . $))

```

```

(S (|differentiate| S S (|Symbol|))) ($ (|rhs| S $) (|lhs| S $)))
(|dimension|
  ((#2=(|CardinalNumber|))
    (#2# (|coerce| (|CardinalNumber|) (|NonNegativeInteger|))))
(|equation| (($ S S) (|arguments| (|r| . S) (|l| . S))))
(|eval| (($ $ $) (|arguments| (|eqn2| . $) (|eqn1| . $))
  (S (|eval| S S (|Equation| S))) ($ (= $ S S)))
  (($ $ #3=(|List| $))
    (|arguments| (|eqn2| . #3#) (|eqn1| . $))
    (S (|eval| S S (|List| (|Equation| S)))) ($ (= $ S S)))
  (($ $ #4=(|List| #5=(|Symbol|)) #6=(|List| S))
    (|arguments| (|lx| . #6#) (|ls| . #4#) (|eqn| . $))
    (S (|eval| S S (|List| (|Symbol|)) (|List| S)))
    ($ (= $ S S)))
  (($ $ #5# S) (|arguments| (|lx| . S) (|ls| . #5#) (|eqn| . $))
    (S (|eval| S S (|Symbol| S)) ($ (= $ S S))))
(|factorAndSplit|
  ((|List| $) $)
  ((|MultivariateFactorize| (|Symbol|)
    (|IndexedExponents| (|Symbol|)) (|Integer|)
    (|Polynomial| (|Integer|)))
    (|factor| (|Factored| (|Polynomial| (|Integer|)))
      (|Polynomial| (|Integer|)))
  ((|Factored| S)
    (|factors|
      (|List| (|Record| (|:| |factor| S)
        (|:| |exponent| (|Integer|))))
      (|Factored| S)))
  ((|Factored| (|Polynomial| (|Integer|)))
    (|factors|
      (|List| (|Record| (|:| |factor| (|Polynomial| (|Integer|)))
        (|:| |exponent| (|Integer|))))
      (|Factored| (|Polynomial| (|Integer|))))
  (|locals| (|p| |Polynomial| (|Integer|)) (|eq0| . $))
  (|arguments| (|eq| . $))
  (S (|factor| (|Factored| S) S) (|Zero| S))
  ($ (|rightZero| $ $) (|lhs| S $) (|equation| $ S S)))
(|inv| (($ $) (|arguments| (|eq| . $)) (S (|inv| S S))
  ($ (|rhs| S $) (|lhs| S $))))
(|leftOne|
  ((|Union| $ "failed") $) (|locals| (|re| |Union| S "failed"))
  (|arguments| (|eq| . $))
  (S (|recip| (|Union| S "failed") S) (|inv| S S) (|One| S)
    (* S S S))
  ($ (|rhs| S $) (|lhs| S $) (|One| $) (= $ S S)))
(|leftZero|
  (($ $) (|arguments| (|eq| . $)) (S (|Zero| S) (- S S S))
  ($ (|rhs| S $) (|lhs| S $) (|Zero| $) (= $ S S)))
(|lhs| (($ $) (|arguments| (|eqn| . $))))
(|map| (($ #7=(|Mapping| S S) $)

```

```

      (|arguments| (|fn| . #7#) (|eqn| . $)) ($ (|equation| $ S S)))
(|recip| (((|Union| $ "failed") $)
  (|locals| (|rh| |Union| S "failed")
    (|lh| |Union| S "failed")))
  (|arguments| (|eq| . $))
  (S (|recip| (|Union| S "failed") S))
  ($ (|rhs| S $) (|lhs| S $))))
(|rhs| ((S $) (|arguments| (|eqn| . $))))
(|rightOne|
  (((|Union| $ "failed") $) (|locals| (|re| |Union| S "failed")
    (|arguments| (|eq| . $))
    (S (|recip| (|Union| S "failed") S) (|inv| S S) (|One| S)
      (* S S S))
    ($ (|rhs| S $) (|lhs| S $) (= $ S S))))))
(|rightZero|
  (($ $) (|arguments| (|eq| . $)) (S (|Zero| S) (- S S S))
    ($ (|rhs| S $) (|lhs| S $) (= $ S S))))
(|subst| (($ $ $) (|locals| (|eq3| |Equation| S))
  (|arguments| (|eq2| . $) (|eq1| . $))
  (S (|subst| S S (|Equation| S))
    ($ (|rhs| S $) (|lhs| S $))))
(|swap| (($ $) (|arguments| (|eqn| . $)) ($ (|rhs| S $) (|lhs| S $)))))

```

1.6 The EQ.fn file

```

(in-package 'compiler)(init-fn)
(ADD-FN-DATA '(
#S(FN NAME BOOT::|EQ;*;S2$;26| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightOne;$U;32| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (BOOT::|spadConstant| VMLISP:QCDR CONS VMLISP:QCAR EQL
    BOOT::|EQQCAR COND VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL BOOT::|LETT VMLISP:SEQ RETURN)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QCDR VMLISP:QCAR BOOT::|EQQCAR COND
    VMLISP:EXIT VMLISP:QREFELT BOOT:SPADCALL BOOT::|LETT
    VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;lhs;$S;4| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CAR VMLISP:QCAR) RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT
  NIL MACROS (VMLISP:QCAR))
#S(FN NAME BOOT::|EQ;+;3$;15| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT

```

```

        BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;dimension;Cn;40| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightZero;2$;22| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES
    (BOOT::|spadConstant| CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;coerce;$Of;13| DEF DEFUN VALUE-TYPE T FUN-VALUES
    NIL CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;One;$;29| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
    (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;inv;2$;42| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;$S$;20| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;=;2$B;12| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL COND)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL COND))
#S(FN NAME BOOT::|EQ;/;3$;41| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
     BOOT:SPADCALL)
    RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;recip;$U;30| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
    CALLEES
    (VMLISP:QCDR LIST* CONS VMLISP:QCAR EQL BOOT::QEQCAR COND
     VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
     BOOT::LETT VMLISP:SEQ RETURN)
    RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
    (VMLISP:QCDR VMLISP:QCAR BOOT::QEQCAR COND VMLISP:EXIT

```

```

        VMLISP:QREFELT BOOT:SPADCALL BOOT::LETT VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;-;3$;24| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$L$;11| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftZero;2$;21| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;*;S2$;27| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;*;I2$;37| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE
  NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;3$;10| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$SS$;8| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES
  (VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;factorAndSplit;$L;38| DEF DEFUN VALUE-TYPE T
  FUN-VALUES NIL CALLEES
  (BOOT:|Integer| BOOT:|Polynomial| EQUAL BOOT:NREVERSEO
    BOOT::|spadConstant| VMLISP:QCAR CONS ATOM VMLISP:EXIT CDR
    CAR BOOT:SPADCALL BOOT::LETT BOOT::|devaluate| LIST SVREF
    VMLISP:QREFELT BOOT::|HasSignature| COND VMLISP:SEQ RETURN)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QCAR VMLISP:EXIT BOOT:SPADCALL
    BOOT::LETT VMLISP:QREFELT COND VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;differentiate;$S$;39| DEF DEFUN VALUE-TYPE T

```

```

FUN-VALUES NIL CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS) RETURN-TYPE NIL
ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;eval;$LL$;9| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T T) NO-EMIT NIL MACROS
(VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftOne;$U;34| DEF DEFUN VALUE-TYPE T FUN-VALUES
NIL CALLEES
(CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
CONS)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;map;M2$;7| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;S2$;19| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;equation;2S$;3| DEF DEFUN VALUE-TYPE T FUN-VALUES
NIL CALLEES (CONS) RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL
MACROS NIL)
#S(FN NAME BOOT::|EQ;+;$S$;17| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;factorAndSplit;$L;1| DEF DEFUN VALUE-TYPE T
FUN-VALUES NIL CALLEES
(BOOT:NREVERSEO BOOT::|spadConstant| VMLISP:QCAR CONS ATOM
VMLISP:EXIT CDR CAR BOOT:SPADCALL BOOT::LETT
BOOT::|devaluate| LIST SVREF VMLISP:QREFELT
BOOT::|HasSignature| COND VMLISP:SEQ RETURN)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(BOOT::|spadConstant| VMLISP:QCAR VMLISP:EXIT BOOT:SPADCALL
BOOT::LETT VMLISP:QREFELT COND VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;*;3$;25| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;Zero;$;23| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES

```

```

(CDR BOOT::|spadConstant| CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
(BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;characteristic;Nni;36| DEF DEFUN VALUE-TYPE T
FUN-VALUES NIL CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE NIL
ARG-TYPES (T) NO-EMIT NIL MACROS (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;leftOne;$U;31| DEF DEFUN VALUE-TYPE T FUN-VALUES
NIL CALLEES
(VMLISP:QCDR BOOT::|spadConstant| CONS VMLISP:QCAR EQL
BOOT::|QEQCAR COND VMLISP:EXIT CDR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL BOOT::LETT VMLISP:SEQ RETURN)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QCDR BOOT::|spadConstant| VMLISP:QCAR BOOT::|QEQCAR COND
VMLISP:EXIT VMLISP:QREFELT BOOT:SPADCALL BOOT::LETT
VMLISP:SEQ RETURN))
#S(FN NAME BOOT::|EQ;swap;2$;6| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;-;2$;18| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL) RETURN-TYPE
NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;subst;3$;43| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS VMLISP:EXIT
BOOT::LETT VMLISP:SEQ RETURN)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL VMLISP:EXIT BOOT::LETT VMLISP:SEQ
RETURN))
#S(FN NAME BOOT::|EQ;=;2S$;2| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CONS) RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL
MACROS NIL)
#S(FN NAME BOOT::|EQ;*;$S$;28| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(VMLISP:QCDR CDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;+;S2$;16| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES (CDR CONS CAR SVREF VMLISP:QREFELT BOOT:SPADCALL)
RETURN-TYPE NIL ARG-TYPES (T T T) NO-EMIT NIL MACROS
(VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|Equation;| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
CALLEES
(BOOT::|EQ;One;$;29| BOOT::|EQ;Zero;$;23|
BOOT::|dispatchFunction| BOOT::|testBitVector| COND
BOOT::|Record0| BOOT::|Record| BOOT::|stuffDomainSlots| CONS
BOOT::|haddProp| BOOT::|HasCategory| BOOT::|buildPredVector|

```

```

SYSTEM:SVSET SETF VMLISP:QSETREFV LIST
BOOT::|devaluate| BOOT::LETT RETURN)
RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
(BOOT::|dispatchFunction| COND BOOT::|Record| SETF
  VMLISP:QSETREFV BOOT::LETT RETURN))
#S(FN NAME BOOT::|EQ;coerce;$B;14| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (CDR VMLISP:QCDR VMLISP:QCAR CAR SVREF VMLISP:QREFELT
    BOOT:SPADCALL)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (VMLISP:QCDR VMLISP:QCAR VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rhs;$S;5| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR VMLISP:QCDR) RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT
  NIL MACROS (VMLISP:QCDR))
#S(FN NAME OTHER-FORM DEF NIL VALUE-TYPE NIL FUN-VALUES NIL CALLEES NIL
  RETURN-TYPE NIL ARG-TYPES NIL NO-EMIT NIL MACROS NIL)
#S(FN NAME BOOT::|EQ;inv;2$;33| DEF DEFUN VALUE-TYPE T FUN-VALUES NIL
  CALLEES (CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL CONS)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|EQ;rightOne;$U;35| DEF DEFUN VALUE-TYPE T FUN-VALUES
  NIL CALLEES
  (BOOT::|spadConstant| CDR CAR SVREF VMLISP:QREFELT BOOT:SPADCALL
    CONS)
  RETURN-TYPE NIL ARG-TYPES (T T) NO-EMIT NIL MACROS
  (BOOT::|spadConstant| VMLISP:QREFELT BOOT:SPADCALL))
#S(FN NAME BOOT::|Equation| DEF DEFUN VALUE-TYPE T FUN-VALUES
  (SINGLE-VALUE) CALLEES
  (REMHASH VMLISP:HREM BOOT::|Equation;| PROG1
    BOOT::|CDRwithIncrement| GETHASH VMLISP:HGET
    BOOT::|devaluate| LIST BOOT::|lassocShiftWithFunction|
    BOOT::LETT COND RETURN)
  RETURN-TYPE NIL ARG-TYPES (T) NO-EMIT NIL MACROS
  (VMLISP:HREM PROG1 VMLISP:HGET BOOT::LETT COND RETURN)) )

```

1.7 The index.kaf file

Each constructor (e.g. EQ) had one library directory (e.g. EQ.nrlib). This directory contained a random access file called the index.kaf file. These files contain runtime information such as the operationAlist and the ConstructorModemap. At system build time we merge all of these .nrlib/index.kaf files into one database, INTERP.daase. Requests to get information from this database are cached so that multiple references do not cause additional disk i/o.

Before getting into the contents, we need to understand the format of an index.kaf file. The kaf file is a random access file, originally used as a database. In the current system we make a pass to combine these files at build time to construct the various daase files.

This is just a file of lisp objects, one after another, in (read) format.

A kaf file starts with an integer, in this case, 35695. This integer gives the byte offset to the index. Due to the way the file is constructed, the index is at the end of the file. To read a kaf file, first read the integer, then seek to that location in the file, and do a (read). This will return the index, in this case:

```
((("slot1Info" 0 32444)
  ("documentation" 0 29640)
  ("ancestors" 0 28691)
  ("parents" 0 28077)
  ("abbreviation" 0 28074)
  ("predicates" 0 25442)
  ("attributes" 0 25304)
  ("signaturesAndLocals" 0 23933)
  ("superDomain" 0 NIL)
  ("operationAlist" 0 20053)
  ("modemaps" 0 17216)
  ("sourceFile" 0 17179)
  ("constructorCategory" 0 15220)
  ("constructorModemap" 0 13215)
  ("constructorKind" 0 13206)
  ("constructorForm" 0 13191)
  ("compilerInfo" 0 4433)
  ("loadTimeStuff" 0 20))
```

This is a list of triples. The first item in each triple is a string that is used as a lookup key (e.g. “operationAlist”). The second element is no longer used. The third element is the byte offset from the beginning of the file.

So to read the “operationAlist” from this file you would:

1. open the index.kaf file
2. (read) the integer
3. (seek) to the integer offset from the beginning of the file
4. (read) the index of triples
5. find the keyword (e.g. “operationAlist”) triple
6. select the third element, an integer
7. (seek) to the integer offset from the beginning of the file
8. (read) the “operationAlist”

Note that the information below has been reformatted to fit this document. In order to save space the index.kaf file is does not use prettyprint since it is normally only read by machine.

1.7.1 The index offset byte

35695

1.7.2 The “loadTimeStuff”

```
(setf (get '|Equation| '|infovec|)
  (LIST '#(NIL NIL NIL NIL NIL NIL (|local| |#1|) '|Rep|
    (0 . |rightZero|) |EQ;lhs;$S;4| (|Factored| $)
    (5 . |factor|)
    (|Record| (|:| |factor| 6) (|:| |exponent| 74))
    (|List| 12) (|Factored| 6) (10 . |factors|) (15 . |Zero|)
    |EQ;equation;2S$;3| (|List| $) (19 . |factorAndSplit|)
    |EQ;=;2S$;2| |EQ;rhs;$S;5| |EQ;swap;2$;6| (|Mapping| 6 6)
    |EQ;map;M2$;7| (|Symbol|) (24 . |eval|) (31 . |eval|)
    (|List| 25) (|List| 6) (38 . |eval|) (45 . |eval|)
    (|Equation| 6) (52 . |eval|) (58 . |eval|) (|List| 32)
    (64 . |eval|) (70 . |eval|) (|Boolean|) (76 . =) (82 . =)
    (|OutputForm|) (88 . |coerce|) (93 . =) (99 . |coerce|)
    (104 . |coerce|) (109 . +) (115 . +) (121 . +) (127 . +)
    (133 . -) (138 . -) (143 . -) (149 . -) (155 . -)
    (161 . |Zero|) (165 . -) (171 . |leftZero|) (176 . *)
    (182 . *) (188 . *) (194 . *) (200 . |One|) (204 . |One|)
    (|Union| $ '"failed") (208 . |recip|) (213 . |recip|)
    (218 . |leftOne|) (223 . |rightOne|) (228 . |inv|)
    (233 . |inv|) (|NonNegativeInteger|)
    (238 . |characteristic|) (242 . |characteristic|)
    (|Integer|) (246 . |coerce|) (251 . *) (|Factored| 78)
    (|Polynomial| 74)
    (|MultivariateFactorize| 25 (|IndexedExponents| 25) 74 78)
    (257 . |factor|)
    (|Record| (|:| |factor| 78) (|:| |exponent| 74))
    (|List| 81) (262 . |factors|) (267 . |differentiate|)
    (273 . |differentiate|) (|CardinalNumber|)
    (279 . |coerce|) (284 . |dimension|) (288 . /) (294 . /)
    (|Equation| $) (300 . |subst|) (306 . |subst|)
    (|PositiveInteger|) (|List| 71) (|SingleInteger|)
    (|String|))
  '#(~= 312 |zero?| 318 |swap| 323 |subtractIfCan| 328 |subst|
    334 |sample| 340 |rightZero| 344 |rightOne| 349 |rhs| 354
    |recip| 359 |one?| 364 |map| 369 |lhs| 375 |leftZero| 380
    |leftOne| 385 |latex| 390 |inv| 395 |hash| 400
    |factorAndSplit| 405 |eval| 410 |equation| 436 |dimension|
    442 |differentiate| 446 |conjugate| 472 |commutator| 478
    |coerce| 484 |characteristic| 499 ^ 503 |Zero| 521 |One|
    525 D 529 = 555 / 567 - 579 + 602 ** 620 * 638)
  '((|unitsKnown| . 12) (|rightUnitary| . 3)
    (|leftUnitary| . 3))
  (CONS (|makeByteWordVec2| 25
```

```

'(1 15 4 14 5 14 3 5 3 21 21 6 21 17 24 19 25 0 2
  25 2 7))
(CONS '#(|VectorSpace&| |Module&|
  |PartialDifferentialRing&| NIL |Ring&| NIL NIL
  NIL NIL |AbelianGroup&| NIL |Group&|
  |AbelianMonoid&| |Monoid&| |AbelianSemiGroup&|
  |SemiGroup&| |SetCategory&| NIL NIL
  |BasicType&| NIL |InnerEvalable&|)
(CONS '#(|VectorSpace| 6) (|Module| 6)
  (|PartialDifferentialRing| 25)
  (|BiModule| 6 6) (|Ring|)
  (|LeftModule| 6) (|RightModule| 6)
  (|Rng|) (|LeftModule| $$)
  (|AbelianGroup|)
  (|CancellationAbelianMonoid|) (|Group|)
  (|AbelianMonoid|) (|Monoid|)
  (|AbelianSemiGroup|) (|SemiGroup|)
  (|SetCategory|) (|Type|)
  (|CoercibleTo| 41) (|BasicType|)
  (|CoercibleTo| 38)
  (|InnerEvalable| 25 6))
(|makeByteWordVec2| 97
  '(1 0 0 0 8 1 6 10 0 11 1 14 13 0 15 0
    6 0 16 1 0 18 0 19 3 6 0 0 25 6 26 3
    0 0 0 25 6 27 3 6 0 0 28 29 30 3 0 0
    0 28 29 31 2 6 0 0 32 33 2 0 0 0 0 34
    2 6 0 0 35 36 2 0 0 0 18 37 2 6 38 0
    0 39 2 0 38 0 0 40 1 6 41 0 42 2 41 0
    0 0 43 1 0 41 0 44 1 0 38 0 45 2 6 0
    0 0 46 2 0 0 0 0 47 2 0 0 6 0 48 2 0
    0 0 6 49 1 6 0 0 50 1 0 0 0 51 2 0 0
    0 0 52 2 0 0 6 0 53 2 0 0 0 6 54 0 0
    0 55 2 6 0 0 0 56 1 0 0 0 57 2 6 0 0
    0 58 2 0 0 0 0 59 2 0 0 6 0 60 2 0 0
    0 6 61 0 6 0 62 0 0 0 63 1 6 64 0 65
    1 0 64 0 66 1 0 64 0 67 1 0 64 0 68 1
    6 0 0 69 1 0 0 0 70 0 6 71 72 0 0 71
    73 1 6 0 74 75 2 0 0 74 0 76 1 79 77
    78 80 1 77 82 0 83 2 6 0 0 25 84 2 0
    0 0 25 85 1 86 0 71 87 0 0 86 88 2 6
    0 0 0 89 2 0 0 0 0 90 2 6 0 0 91 92 2
    0 0 0 0 93 2 2 38 0 0 1 1 20 38 0 1 1
    0 0 0 22 2 20 64 0 0 1 2 10 0 0 0 93
    0 22 0 1 1 20 0 0 8 1 16 64 0 68 1 0
    6 0 21 1 16 64 0 66 1 16 38 0 1 2 0 0
    23 0 24 1 0 6 0 9 1 20 0 0 57 1 16 64
    0 67 1 2 97 0 1 1 11 0 0 70 1 2 96 0
    1 1 9 18 0 19 2 8 0 0 0 34 2 8 0 0 18
    37 3 7 0 0 25 6 27 3 7 0 0 28 29 31 2
    0 0 6 6 17 0 1 86 88 2 4 0 0 28 1 2 4

```

```

0 0 25 85 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 6 0 0 0 1 2 6 0 0 0 1 1 3 0 74
1 1 2 41 0 44 1 2 38 0 45 0 3 71 73 2
6 0 0 74 1 2 16 0 0 71 1 2 18 0 0 94
1 0 20 0 55 0 16 0 63 2 4 0 0 28 1 2
4 0 0 25 1 3 4 0 0 28 95 1 3 4 0 0 25
71 1 2 2 38 0 0 40 2 0 0 6 6 20 2 11
0 0 0 90 2 1 0 0 6 1 1 20 0 0 51 2 20
0 0 0 52 2 20 0 6 0 53 2 20 0 0 6 54
2 23 0 0 0 47 2 23 0 6 0 48 2 23 0 0
6 49 2 6 0 0 74 1 2 16 0 0 71 1 2 18
0 0 94 1 2 20 0 71 0 1 2 20 0 74 0 76
2 23 0 94 0 1 2 18 0 0 0 59 2 18 0 0
6 61 2 18 0 6 0 60))))))
'lookupComplete))

```

1.7.3 The “compilerInfo”

```

(SETQ |$CategoryFrame|
  (|put| '|Equation| '|isFunction|
    '(((|eval| ($ $ (|List| (|Symbol|)) (|List| |#1|)))
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ELT $ 31))
    ((|eval| ($ $ (|Symbol|) |#1|))
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ELT $ 27))
    ((~= ((|Boolean|) $ $)) (|has| |#1| (|SetCategory|))
      (ELT $ NIL))
    ((= ((|Boolean|) $ $)) (|has| |#1| (|SetCategory|))
      (ELT $ 40))
    ((|coerce| ((|OutputForm|) $))
      (|has| |#1| (|SetCategory|)) (ELT $ 44))
    ((|hash| ((|SingleInteger|) $))
      (|has| |#1| (|SetCategory|)) (ELT $ NIL))
    ((|latex| ((|String|) $)) (|has| |#1| (|SetCategory|))
      (ELT $ NIL))
    ((|coerce| ((|Boolean|) $)) (|has| |#1| (|SetCategory|))
      (ELT $ 45))
    ((+ ($ $ $)) (|has| |#1| (|AbelianSemiGroup|))
      (ELT $ 47))
    ((* ($ (|PositiveInteger|) $))
      (|has| |#1| (|AbelianSemiGroup|)) (ELT $ NIL))
    ((|Zero| ($)) (|has| |#1| (|AbelianGroup|))
      (CONST $ 55))
    ((|sample| ($))
      (OR (|has| |#1| (|AbelianGroup|))
        (|has| |#1| (|Monoid|)))
      (CONST $ NIL))
    ((|zero?| ((|Boolean|) $)) (|has| |#1| (|AbelianGroup|))

```

```

(ELT $ NIL))
((* ($ (|NonNegativeInteger|) $))
  (|has| |#1| (|AbelianGroup|)) (ELT $ NIL))
(|subtractIfCan| ((|Union| $ "failed") $ $))
  (|has| |#1| (|AbelianGroup|)) (ELT $ NIL))
((- ($ $)) (|has| |#1| (|AbelianGroup|)) (ELT $ 51))
((- ($ $ $)) (|has| |#1| (|AbelianGroup|)) (ELT $ 52))
((* ($ (|Integer|) $)) (|has| |#1| (|AbelianGroup|))
  (ELT $ 76))
((* ($ $ $)) (|has| |#1| (|SemiGroup|)) (ELT $ 59))
(** ($ $ (|PositiveInteger|)))
  (|has| |#1| (|SemiGroup|)) (ELT $ NIL))
(^ ($ $ (|PositiveInteger|)))
  (|has| |#1| (|SemiGroup|)) (ELT $ NIL))
(|One| ($)) (|has| |#1| (|Monoid|)) (CONST $ 63))
(|one?| ((|Boolean|) $)) (|has| |#1| (|Monoid|))
  (ELT $ NIL))
(** ($ $ (|NonNegativeInteger|)))
  (|has| |#1| (|Monoid|)) (ELT $ NIL))
(^ ($ $ (|NonNegativeInteger|)))
  (|has| |#1| (|Monoid|)) (ELT $ NIL))
(|recip| ((|Union| $ "failed") $))
  (|has| |#1| (|Monoid|)) (ELT $ 66))
(|inv| ($ $))
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))
  (ELT $ 70))
(/ ($ $ $))
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))
  (ELT $ 90))
(** ($ $ (|Integer|))) (|has| |#1| (|Group|))
  (ELT $ NIL))
(^ ($ $ (|Integer|))) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|conjugate| ($ $ $)) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|commutator| ($ $ $)) (|has| |#1| (|Group|))
  (ELT $ NIL))
(|characteristic| ((|NonNegativeInteger|)))
  (|has| |#1| (|Ring|)) (ELT $ 73))
(|coerce| ($ (|Integer|))) (|has| |#1| (|Ring|))
  (ELT $ NIL))
(* ($ |#1| $)) (|has| |#1| (|SemiGroup|)) (ELT $ 60))
(* ($ $ |#1|)) (|has| |#1| (|SemiGroup|)) (ELT $ 61))
(|differentiate| ($ $ (|Symbol|)))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ 85))
(|differentiate| ($ $ (|List| (|Symbol|))))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
(|differentiate|

```

```

($ $ (|Symbol|) (|NonNegativeInteger|))
(|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
(ELT $ NIL))
((|differentiate|
  ($ $ (|List| (|Symbol|))
    (|List| (|NonNegativeInteger|))))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
((D ($ $ (|Symbol|))
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ELT $ NIL))
  ((D ($ $ (|List| (|Symbol|)))
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
    (ELT $ NIL))
    ((D ($ $ (|Symbol|) (|NonNegativeInteger|))
      (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
      (ELT $ NIL))
      ((D ($ $ (|List| (|Symbol|))
        (|List| (|NonNegativeInteger|))))
        (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
        (ELT $ NIL))
        (/( $ $ |#1|) (|has| |#1| (|Field|)) (ELT $ NIL))
        ((|dimension| ((|CardinalNumber|))
          (|has| |#1| (|Field|)) (ELT $ 88))
          ((|subst| ($ $ $)) (|has| |#1| (|ExpressionSpace|))
            (ELT $ 93))
          ((|factorAndSplit| ((|List| $) $))
            (|has| |#1| (|IntegralDomain|)) (ELT $ 19))
          ((|rightOne| ((|Union| $ "failed") $))
            (|has| |#1| (|Monoid|)) (ELT $ 68))
          ((|leftOne| ((|Union| $ "failed") $))
            (|has| |#1| (|Monoid|)) (ELT $ 67))
          ((- ($ $ |#1|)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 54))
          ((- ($ |#1| $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 53))
          ((|rightZero| ($ $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 8))
          ((|leftZero| ($ $)) (|has| |#1| (|AbelianGroup|))
            (ELT $ 57))
          ((+ ($ $ |#1|)) (|has| |#1| (|AbelianSemiGroup|))
            (ELT $ 49))
          ((+ ($ |#1| $)) (|has| |#1| (|AbelianSemiGroup|))
            (ELT $ 48))
          ((|eval| ($ $ (|List| $)))
            (AND (|has| |#1| (|Evalable| |#1|))
              (|has| |#1| (|SetCategory|)))
            (ELT $ 37))
          ((|eval| ($ $ $))
            (AND (|has| |#1| (|Evalable| |#1|))

```

```

(|has| |#1| (|SetCategory|)))
(ELT $ 34))
((|map| ($ (|Mapping| |#1| |#1|) $)) T (ELT $ 24))
((|rhs| (|#1| $)) T (ELT $ 21))
((|lhs| (|#1| $)) T (ELT $ 9))
((|swap| ($ $)) T (ELT $ 22))
((|equation| ($ |#1| |#1|)) T (ELT $ 17))
((= ($ |#1| |#1|)) T (ELT $ 20)))
(|addModemap| '|Equation| '(|Equation| |#1|)
'(|Join| (|Type|)
(CATEGORY |domain|
(SIGNATURE = ($ |#1| |#1|))
(SIGNATURE |equation| ($ |#1| |#1|))
(SIGNATURE |swap| ($ $))
(SIGNATURE |lhs| (|#1| $))
(SIGNATURE |rhs| (|#1| $))
(SIGNATURE |map|
($ (|Mapping| |#1| |#1|) $))
(IF (|has| |#1|
(|InnerEvalable| (|Symbol|) |#1|))
(ATTRIBUTE
(|InnerEvalable| (|Symbol|) |#1|))
|noBranch|)
(IF (|has| |#1| (|SetCategory|))
(PROGN
(ATTRIBUTE (|SetCategory|))
(ATTRIBUTE
(|CoercibleTo| (|Boolean|)))
(IF (|has| |#1| (|Evalable| |#1|))
(PROGN
(SIGNATURE |eval| ($ $ $))
(SIGNATURE |eval|
($ $ (|List| $))))
|noBranch|))
|noBranch|)
(IF (|has| |#1| (|AbelianSemiGroup|))
(PROGN
(ATTRIBUTE (|AbelianSemiGroup|))
(SIGNATURE + ($ |#1| $))
(SIGNATURE + ($ $ |#1|))
|noBranch|)
(IF (|has| |#1| (|AbelianGroup|))
(PROGN
(ATTRIBUTE (|AbelianGroup|))
(SIGNATURE |leftZero| ($ $))
(SIGNATURE |rightZero| ($ $))
(SIGNATURE - ($ |#1| $))
(SIGNATURE - ($ $ |#1|))
|noBranch|)
(IF (|has| |#1| (|SemiGroup|))

```

```

      (PROGN
        (ATTRIBUTE (|SemiGroup|))
        (SIGNATURE * ($ |#1| $))
        (SIGNATURE * ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|Monoid|))
      (PROGN
        (ATTRIBUTE (|Monoid|))
        (SIGNATURE |leftOne|
          ((|Union| $ "failed") $))
        (SIGNATURE |rightOne|
          ((|Union| $ "failed") $)))
        |noBranch|)
    (IF (|has| |#1| (|Group|))
      (PROGN
        (ATTRIBUTE (|Group|))
        (SIGNATURE |leftOne|
          ((|Union| $ "failed") $))
        (SIGNATURE |rightOne|
          ((|Union| $ "failed") $)))
        |noBranch|)
    (IF (|has| |#1| (|Ring|))
      (PROGN
        (ATTRIBUTE (|Ring|))
        (ATTRIBUTE (|BiModule| |#1| |#1|)))
        |noBranch|)
    (IF (|has| |#1| (|CommutativeRing|))
      (ATTRIBUTE (|Module| |#1|))
      |noBranch|)
    (IF (|has| |#1| (|IntegralDomain|))
      (SIGNATURE |factorAndSplit|
        ((|List| $) $))
      |noBranch|)
    (IF (|has| |#1|
      (|PartialDifferentialRing|
        (|Symbol|)))
      (ATTRIBUTE
        (|PartialDifferentialRing|
          (|Symbol|)))
      |noBranch|)
    (IF (|has| |#1| (|Field|))
      (PROGN
        (ATTRIBUTE (|VectorSpace| |#1|))
        (SIGNATURE / ($ $ $))
        (SIGNATURE |inv| ($ $)))
        |noBranch|)
    (IF (|has| |#1| (|ExpressionSpace|))
      (SIGNATURE |subst| ($ $ $))
      |noBranch|)))
(|Type|))

```

```

T '|Equation|
(|put| '|Equation| '|model|
  '|Mapping|
    (|Join| (|Type|)
      (CATEGORY |domain|
        (SIGNATURE = ($ |#1| |#1|))
        (SIGNATURE |equation|
          ($ |#1| |#1|))
        (SIGNATURE |swap| ($ $))
        (SIGNATURE |lhs| (|#1| $))
        (SIGNATURE |rhs| (|#1| $))
        (SIGNATURE |map|
          ($ (|Mapping| |#1| |#1|) $))
        (IF
          (|has| |#1|
            (|InnerEvalable| (|Symbol|)
              |#1|))
          (ATTRIBUTE
            (|InnerEvalable| (|Symbol|)
              |#1|))
          |noBranch|)
        (IF (|has| |#1| (|SetCategory|))
          (PROGN
            (ATTRIBUTE (|SetCategory|))
            (ATTRIBUTE
              (|CoercibleTo| (|Boolean|)))
            (IF
              (|has| |#1|
                (|Evalable| |#1|))
              (PROGN
                (SIGNATURE |eval| ($ $ $))
                (SIGNATURE |eval|
                  ($ $ (|List| $))))
              |noBranch|))
          |noBranch|)
        (IF
          (|has| |#1|
            (|AbelianSemiGroup|))
          (PROGN
            (ATTRIBUTE
              (|AbelianSemiGroup|))
            (SIGNATURE + ($ |#1| $))
            (SIGNATURE + ($ $ |#1|)))
          |noBranch|)
        (IF (|has| |#1| (|AbelianGroup|))
          (PROGN
            (ATTRIBUTE (|AbelianGroup|))
            (SIGNATURE |leftZero| ($ $))
            (SIGNATURE |rightZero| ($ $))
            (SIGNATURE - ($ |#1| $))

```

```

        (SIGNATURE - ($ $ |#1|)))
|noBranch|
(IF (|has| |#1| (|SemiGroup|))
  (PROGN
    (ATTRIBUTE (|SemiGroup|))
    (SIGNATURE * ($ |#1| $))
    (SIGNATURE * ($ $ |#1|)))
|noBranch|
(IF (|has| |#1| (|Monoid|))
  (PROGN
    (ATTRIBUTE (|Monoid|))
    (SIGNATURE |leftOne|
      ((|Union| $ "failed") $))
    (SIGNATURE |rightOne|
      ((|Union| $ "failed") $)))
|noBranch|
(IF (|has| |#1| (|Group|))
  (PROGN
    (ATTRIBUTE (|Group|))
    (SIGNATURE |leftOne|
      ((|Union| $ "failed") $))
    (SIGNATURE |rightOne|
      ((|Union| $ "failed") $)))
|noBranch|
(IF (|has| |#1| (|Ring|))
  (PROGN
    (ATTRIBUTE (|Ring|))
    (ATTRIBUTE
      (|BiModule| |#1| |#1|)))
|noBranch|
(IF
  (|has| |#1| (|CommutativeRing|))
  (ATTRIBUTE (|Module| |#1|))
|noBranch|
(IF
  (|has| |#1| (|IntegralDomain|))
  (SIGNATURE |factorAndSplit|
    ((|List| $) $))
|noBranch|
(IF
  (|has| |#1|
    (|PartialDifferentialRing|
      (|Symbol|)))
  (ATTRIBUTE
    (|PartialDifferentialRing|
      (|Symbol|)))
|noBranch|
(IF (|has| |#1| (|Field|))
  (PROGN
    (ATTRIBUTE

```

```

(|VectorSpace| |#1|))
(SIGNATURE / ($ $ $))
(SIGNATURE |inv| ($ $ $))
|noBranch|)
(IF
(|has| |#1| (|ExpressionSpace|))
(SIGNATURE |subst| ($ $ $))
|noBranch|)))
(|Type|))
|$CategoryFrame|))))

```

1.7.4 The “constructorForm”

```
(|Equation| S)
```

1.7.5 The “constructorKind”

```
|domain|
```

1.7.6 The “constructorModemap”

```

(((|Equation| |#1|)
(|Join| (|Type|)
(CATEGORY |domain| (SIGNATURE = ($ |#1| |#1|))
(SIGNATURE |equation| ($ |#1| |#1|))
(SIGNATURE |swap| ($ $)) (SIGNATURE |lhs| (|#1| $))
(SIGNATURE |rhs| (|#1| $))
(SIGNATURE |map| ($ (|Mapping| |#1| |#1|) $))
(IF (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
(ATTRIBUTE (|InnerEvalable| (|Symbol|) |#1|))
|noBranch|)
(IF (|has| |#1| (|SetCategory|))
(PROGN
(ATTRIBUTE (|SetCategory|))
(ATTRIBUTE (|CoercibleTo| (|Boolean|)))
(IF (|has| |#1| (|Evalable| |#1|))
(PROGN
(SIGNATURE |eval| ($ $ $))
(SIGNATURE |eval| ($ $ (|List| $))))
|noBranch|))
|noBranch|)
(IF (|has| |#1| (|AbelianSemiGroup|))
(PROGN
(ATTRIBUTE (|AbelianSemiGroup|))
(SIGNATURE + ($ |#1| $))
(SIGNATURE + ($ $ |#1|)))
|noBranch|)

```

```

(IF (|has| |#1| (|AbelianGroup|))
  (PROGN
    (ATTRIBUTE (|AbelianGroup|))
    (SIGNATURE |leftZero| ($ $))
    (SIGNATURE |rightZero| ($ $))
    (SIGNATURE - ($ |#1| $))
    (SIGNATURE - ($ $ |#1|)))
  |noBranch|)
(IF (|has| |#1| (|SemiGroup|))
  (PROGN
    (ATTRIBUTE (|SemiGroup|))
    (SIGNATURE * ($ |#1| $))
    (SIGNATURE * ($ $ |#1|)))
  |noBranch|)
(IF (|has| |#1| (|Monoid|))
  (PROGN
    (ATTRIBUTE (|Monoid|))
    (SIGNATURE |leftOne| ((|Union| $ "failed") $))
    (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
  |noBranch|)
(IF (|has| |#1| (|Group|))
  (PROGN
    (ATTRIBUTE (|Group|))
    (SIGNATURE |leftOne| ((|Union| $ "failed") $))
    (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
  |noBranch|)
(IF (|has| |#1| (|Ring|))
  (PROGN
    (ATTRIBUTE (|Ring|))
    (ATTRIBUTE (|BiModule| |#1| |#1|)))
  |noBranch|)
(IF (|has| |#1| (|CommutativeRing|))
  (ATTRIBUTE (|Module| |#1|)) |noBranch|)
(IF (|has| |#1| (|IntegralDomain|))
  (SIGNATURE |factorAndSplit| ((|List| $) $))
  |noBranch|)
(IF (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ATTRIBUTE (|PartialDifferentialRing| (|Symbol|)))
  |noBranch|)
(IF (|has| |#1| (|Field|))
  (PROGN
    (ATTRIBUTE (|VectorSpace| |#1|))
    (SIGNATURE / ($ $ $))
    (SIGNATURE |inv| ($ $)))
  |noBranch|)
(IF (|has| |#1| (|ExpressionSpace|))
  (SIGNATURE |subst| ($ $ $)) |noBranch|))
(|Type|)
(T |Equation|)

```

1.7.7 The “constructorCategory”

```

(|Join| (|Type|)
  (CATEGORY |domain| (SIGNATURE = ($ |#1| |#1|))
    (SIGNATURE |equation| ($ |#1| |#1|))
    (SIGNATURE |swap| ($ $)) (SIGNATURE |lhs| (|#1| $))
    (SIGNATURE |rhs| (|#1| $))
    (SIGNATURE |map| ($ (|Mapping| |#1| |#1|) $))
    (IF (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))
      (ATTRIBUTE (|InnerEvalable| (|Symbol|) |#1|))
      |noBranch|)
    (IF (|has| |#1| (|SetCategory|))
      (PROGN
        (ATTRIBUTE (|SetCategory|))
        (ATTRIBUTE (|CoercibleTo| (|Boolean|)))
        (IF (|has| |#1| (|Evalable| |#1|))
          (PROGN
            (SIGNATURE |eval| ($ $ $))
            (SIGNATURE |eval| ($ $ (|List| $))))
          |noBranch|))
      |noBranch|)
    (IF (|has| |#1| (|AbelianSemiGroup|))
      (PROGN
        (ATTRIBUTE (|AbelianSemiGroup|))
        (SIGNATURE + ($ |#1| $))
        (SIGNATURE + ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|AbelianGroup|))
      (PROGN
        (ATTRIBUTE (|AbelianGroup|))
        (SIGNATURE |leftZero| ($ $))
        (SIGNATURE |rightZero| ($ $))
        (SIGNATURE - ($ |#1| $))
        (SIGNATURE - ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|SemiGroup|))
      (PROGN
        (ATTRIBUTE (|SemiGroup|))
        (SIGNATURE * ($ |#1| $))
        (SIGNATURE * ($ $ |#1|)))
      |noBranch|)
    (IF (|has| |#1| (|Monoid|))
      (PROGN
        (ATTRIBUTE (|Monoid|))
        (SIGNATURE |leftOne| ((|Union| $ "failed") $))
        (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
      |noBranch|)
    (IF (|has| |#1| (|Group|))
      (PROGN
        (ATTRIBUTE (|Group|))

```

```

        (SIGNATURE |leftOne| ((|Union| $ "failed") $))
        (SIGNATURE |rightOne| ((|Union| $ "failed") $)))
|noBranch|)
(IF (|has| |#1| (|Ring|))
  (PROGN
    (ATTRIBUTE (|Ring|))
    (ATTRIBUTE (|BiModule| |#1| |#1|)))
|noBranch|)
(IF (|has| |#1| (|CommutativeRing|))
  (ATTRIBUTE (|Module| |#1|)) |noBranch|)
(IF (|has| |#1| (|IntegralDomain|))
  (SIGNATURE |factorAndSplit| ((|List| $) $)) |noBranch|)
(IF (|has| |#1| (|PartialDifferentialRing| (|Symbol|)))
  (ATTRIBUTE (|PartialDifferentialRing| (|Symbol|)))
|noBranch|)
(IF (|has| |#1| (|Field|))
  (PROGN
    (ATTRIBUTE (|VectorSpace| |#1|))
    (SIGNATURE / ($ $ $))
    (SIGNATURE |inv| ($ $)))
|noBranch|)
(IF (|has| |#1| (|ExpressionSpace|))
  (SIGNATURE |subst| ($ $ $)) |noBranch|)))

```

1.7.8 The “sourceFile”

"/research/test/int/algebra/EQ.spad"

1.7.9 The “modemaps”

```

((= (*1 *1 *2 *2)
  (AND (|isDomain| *1 (|Equation| *2)) (|ofCategory| *2 (|Type|))))
  (|equation| (*1 *1 *2 *2)
    (AND (|isDomain| *1 (|Equation| *2)) (|ofCategory| *2 (|Type|))))
  (|swap| (*1 *1 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|lhs| (*1 *2 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|rhs| (*1 *2 *1)
    (AND (|isDomain| *1 (|Equation| *2))
      (|ofCategory| *2 (|Type|))))
  (|map| (*1 *1 *2 *1)
    (AND (|isDomain| *2 (|Mapping| *3 *3))
      (|ofCategory| *3 (|Type|))
      (|isDomain| *1 (|Equation| *3))))
  (|eval| (*1 *1 *1 *1)

```

```

      (AND (|ofCategory| *2 (|Evalable| *2))
            (|ofCategory| *2 (|SetCategory|))
            (|ofCategory| *2 (|Type|))
            (|isDomain| *1 (|Equation| *2))))
(|eval| (*1 *1 *1 *2)
      (AND (|isDomain| *2 (|List| (|Equation| *3)))
            (|ofCategory| *3 (|Evalable| *3))
            (|ofCategory| *3 (|SetCategory|))
            (|ofCategory| *3 (|Type|))
            (|isDomain| *1 (|Equation| *3))))
(+ (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianSemiGroup|))
        (|ofCategory| *2 (|Type|))))
(+ (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianSemiGroup|))
        (|ofCategory| *2 (|Type|))))
(|leftZero| (*1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|))
        (|ofCategory| *2 (|Type|))))
(|rightZero| (*1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|))
        (|ofCategory| *2 (|Type|))))
(- (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|)) (|ofCategory| *2 (|Type|))))
(- (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|AbelianGroup|)) (|ofCategory| *2 (|Type|))))
(|leftOne| (*1 *1 *1)
  (|partial| AND (|isDomain| *1 (|Equation| *2))
                 (|ofCategory| *2 (|Monoid|)) (|ofCategory| *2 (|Type|))))
(|rightOne| (*1 *1 *1)
  (|partial| AND (|isDomain| *1 (|Equation| *2))
                 (|ofCategory| *2 (|Monoid|)) (|ofCategory| *2 (|Type|))))
(|factorAndSplit| (*1 *2 *1)
  (AND (|isDomain| *2 (|List| (|Equation| *3)))
        (|isDomain| *1 (|Equation| *3))
        (|ofCategory| *3 (|IntegralDomain|))
        (|ofCategory| *3 (|Type|))))
(|subst| (*1 *1 *1 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|ExpressionSpace|))
        (|ofCategory| *2 (|Type|))))
(* (*1 *1 *1 *2)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|SemiGroup|)) (|ofCategory| *2 (|Type|))))

```

```

(* (*1 *1 *2 *1)
  (AND (|isDomain| *1 (|Equation| *2))
        (|ofCategory| *2 (|SemiGroup|)) (|ofCategory| *2 (|Type|))))
(/ (*1 *1 *1 *1)
  (OR (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Field|)) (|ofCategory| *2 (|Type|)))
      (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Group|)) (|ofCategory| *2 (|Type|))))
(|inv| (*1 *1 *1)
  (OR (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Field|))
            (|ofCategory| *2 (|Type|)))
      (AND (|isDomain| *1 (|Equation| *2))
            (|ofCategory| *2 (|Group|))
            (|ofCategory| *2 (|Type|)))))

```

1.7.10 The “operationAlist”

```

((~= (((|Boolean|) $) NIL (|has| |#1| (|SetCategory|))))
(|zero?| (((|Boolean|) $) NIL (|has| |#1| (|AbelianGroup|))))
(|swap| (($ $) 22))
(|subtractIfCan|
  (((|Union| $ "failed") $) NIL (|has| |#1| (|AbelianGroup|))))
(|subst| (($ $ $) 93 (|has| |#1| (|ExpressionSpace|))))
(|sample|
  (($) NIL
    (OR (|has| |#1| (|AbelianGroup|)) (|has| |#1| (|Monoid|))) CONST))
(|rightZero| (($ $) 8 (|has| |#1| (|AbelianGroup|))))
(|rightOne| (((|Union| $ "failed") $) 68 (|has| |#1| (|Monoid|))))
(|rhs| ((|#1| $) 21))
(|recip| (((|Union| $ "failed") $) 66 (|has| |#1| (|Monoid|))))
(|one?| (((|Boolean|) $) NIL (|has| |#1| (|Monoid|))))
(|map| (($ (|Mapping| |#1| |#1|) $) 24) (|lhs| ((|#1| $) 9))
(|leftZero| (($ $) 57 (|has| |#1| (|AbelianGroup|))))
(|leftOne| (((|Union| $ "failed") $) 67 (|has| |#1| (|Monoid|))))
(|latex| (((|String|) $) NIL (|has| |#1| (|SetCategory|))))
(|inv| (($ $) 70 (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))))
(|hash| (((|SingleInteger|) $) NIL (|has| |#1| (|SetCategory|))))
(|factorAndSplit| (((|List| $) $) 19 (|has| |#1| (|IntegralDomain|))))
(|eval| (($ $ $) 34
  (AND (|has| |#1| (|Evalable| |#1|))
        (|has| |#1| (|SetCategory|))))
  (($ $ (|List| $)) 37
    (AND (|has| |#1| (|Evalable| |#1|))
          (|has| |#1| (|SetCategory|))))
  (($ $ (|Symbol|) |#1|) 27
    (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
  (($ $ (|List| (|Symbol|)) (|List| |#1|)) 31
    (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|))))

```

```

(|equation| (($ |#1| |#1|) 17))
(|dimension| (((|CardinalNumber|)) 88 (|has| |#1| (|Field|))))
(|differentiate|
  (($ $ (|List| (|Symbol|)) (|List| (|NonNegativeInteger|))) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) (|NonNegativeInteger|)) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|List| (|Symbol|))) NIL
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) 85
    (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(|conjugate| (($ $ $) NIL (|has| |#1| (|Group|))))
(|commutator| (($ $ $) NIL (|has| |#1| (|Group|))))
(|coerce| (($ (|Integer|)) NIL (|has| |#1| (|Ring|))))
  (((|Boolean|) $) 45 (|has| |#1| (|SetCategory|)))
  (((|OutputForm|) $) 44 (|has| |#1| (|SetCategory|)))
(|characteristic| (((|NonNegativeInteger|)) 73 (|has| |#1| (|Ring|))))
(^ (($ $ (|Integer|)) NIL (|has| |#1| (|Group|)))
  (($ $ (|NonNegativeInteger|)) NIL (|has| |#1| (|Monoid|)))
  (($ $ (|PositiveInteger|)) NIL (|has| |#1| (|SemiGroup|))))
(|Zero| (($) 55 (|has| |#1| (|AbelianGroup|)) CONST))
(|One| (($) 63 (|has| |#1| (|Monoid|)) CONST))
(|D| (($ $ (|List| (|Symbol|)) (|List| (|NonNegativeInteger|))) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) (|NonNegativeInteger|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|List| (|Symbol|))) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  (($ $ (|Symbol|)) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(= (($ |#1| |#1|) 20)
  (((|Boolean|) $ $) 40 (|has| |#1| (|SetCategory|))))
(/ (($ $ |#1|) NIL (|has| |#1| (|Field|)))
  (($ $ $) 90 (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))))
(- (($ |#1| $) 53 (|has| |#1| (|AbelianGroup|)))
  (($ $ |#1|) 54 (|has| |#1| (|AbelianGroup|)))
  (($ $ $) 52 (|has| |#1| (|AbelianGroup|)))
  (($ $) 51 (|has| |#1| (|AbelianGroup|))))
(+ (($ |#1| $) 48 (|has| |#1| (|AbelianSemiGroup|)))
  (($ $ |#1|) 49 (|has| |#1| (|AbelianSemiGroup|)))
  (($ $ $) 47 (|has| |#1| (|AbelianSemiGroup|))))
(** (($ $ (|Integer|)) NIL (|has| |#1| (|Group|)))
  (($ $ (|NonNegativeInteger|)) NIL (|has| |#1| (|Monoid|)))
  (($ $ (|PositiveInteger|)) NIL (|has| |#1| (|SemiGroup|))))
(* (($ $ |#1|) 61 (|has| |#1| (|SemiGroup|)))
  (($ |#1| $) 60 (|has| |#1| (|SemiGroup|)))
  (($ $ $) 59 (|has| |#1| (|SemiGroup|)))
  (($ (|Integer|) $) 76 (|has| |#1| (|AbelianGroup|)))
  (($ (|NonNegativeInteger|) $) NIL (|has| |#1| (|AbelianGroup|)))
  (($ (|PositiveInteger|) $) NIL (|has| |#1| (|AbelianSemiGroup|))))

```

1.7.11 The “superDomain”

1.7.12 The “signaturesAndLocals”

```

((|EQ;subst;3$;43| ($ $ $)) (|EQ;inv;2$;42| ($ $))
(|EQ;/;3$;41| ($ $ $)) (|EQ;dimension;Cn;40| ((|CardinalNumber|)))
(|EQ;differentiate;$S$;39| ($ $ (|Symbol|)))
(|EQ;factorAndSplit;$L;38| ((|List| $) $))
(|EQ;*;I2$;37| ($ (|Integer|) $))
(|EQ;characteristic;Nni;36| ((|NonNegativeInteger|)))
(|EQ;rightOne;$U;35| ((|Union| $ "failed") $))
(|EQ;leftOne;$U;34| ((|Union| $ "failed") $)) (|EQ;inv;2$;33| ($ $))
(|EQ;rightOne;$U;32| ((|Union| $ "failed") $))
(|EQ;leftOne;$U;31| ((|Union| $ "failed") $))
(|EQ;recip;$U;30| ((|Union| $ "failed") $)) (|EQ;One;$;29| ($))
(|EQ;*;$S$;28| ($ $ $ S)) (|EQ;*;S2$;27| ($ S $))
(|EQ;*;S2$;26| ($ S $)) (|EQ;*;3$;25| ($ $ $)) (|EQ;-;3$;24| ($ $ $))
(|EQ;Zero;$;23| ($)) (|EQ;rightZero;2$;22| ($ $))
(|EQ;leftZero;2$;21| ($ $)) (|EQ;-;$S$;20| ($ $ S))
(|EQ;-;S2$;19| ($ S $)) (|EQ;-;2$;18| ($ $)) (|EQ;+;$S$;17| ($ $ S))
(|EQ;+;S2$;16| ($ S $)) (|EQ;+;3$;15| ($ $ $))
(|EQ;coerce;$B;14| ((|Boolean|) $))
(|EQ;coerce;$Of;13| ((|OutputForm|) $))
(|EQ;=;2$B;12| ((|Boolean|) $ $)) (|EQ;eval;$L$;11| ($ $ (|List| $)))
(|EQ;eval;3$;10| ($ $ $))
(|EQ;eval;$LL$;9| ($ $ (|List| (|Symbol|)) (|List| S)))
(|EQ;eval;$SS$;8| ($ $ (|Symbol|) S))
(|EQ;map;M2$;7| ($ (|Mapping| S S) $)) (|EQ;swap;2$;6| ($ $))
(|EQ;rhs;$S;5| (S $)) (|EQ;lhs;$S;4| (S $))
(|EQ;equation;2S$;3| ($ S S)) (|EQ;=;2S$;2| ($ S S))
(|EQ;factorAndSplit;$L;1| ((|List| $) $)))

```

1.7.13 The “attributes”

```

((|unitsKnown| OR (|has| |#1| (|Ring|)) (|has| |#1| (|Group|)))
(|rightUnitary| |has| |#1| (|Ring|))
(|leftUnitary| |has| |#1| (|Ring|)))

```

1.7.14 The “predicates”

```

((|HasCategory| |#1| '(|Field|)) (|HasCategory| |#1| '(|SetCategory|))
(|HasCategory| |#1| '(|Ring|))
(|HasCategory| |#1| (LIST '(|PartialDifferentialRing| '(|Symbol|)))
(OR (|HasCategory| |#1| (LIST '(|PartialDifferentialRing| '(|Symbol|)))
(|HasCategory| |#1| '(|Ring|)))
(|HasCategory| |#1| '(|Group|))
(|HasCategory| |#1|
(LIST '(|InnerEvalable| '(|Symbol|) (|devaluate| |#1|))))

```

```

(AND (|HasCategory| |#1| (LIST ' |Evalable| (|devaluate| |#1|)))
      (|HasCategory| |#1| ' (|SetCategory|)))
(|HasCategory| |#1| ' (|IntegralDomain|))
(|HasCategory| |#1| ' (|ExpressionSpace|))
(OR (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Group|)))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Ring|)))
(|HasCategory| |#1| ' (|CommutativeRing|))
(OR (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)))
(|HasCategory| |#1| ' (|Monoid|))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Monoid|)))
(|HasCategory| |#1| ' (|SemiGroup|))
(OR (|HasCategory| |#1| ' (|Group|)) (|HasCategory| |#1| ' (|Monoid|))
      (|HasCategory| |#1| ' (|SemiGroup|)))
(|HasCategory| |#1| ' (|AbelianGroup|))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|Monoid|)))
(|HasCategory| |#1| ' (|AbelianSemiGroup|))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|AbelianSemiGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Ring|)))
(OR (|HasCategory| |#1| (LIST ' |PartialDifferentialRing| ' (|Symbol|)))
      (|HasCategory| |#1| ' (|AbelianGroup|))
      (|HasCategory| |#1| ' (|AbelianSemiGroup|))
      (|HasCategory| |#1| ' (|CommutativeRing|))
      (|HasCategory| |#1| ' (|Field|)) (|HasCategory| |#1| ' (|Group|))
      (|HasCategory| |#1| ' (|Monoid|)) (|HasCategory| |#1| ' (|Ring|))
      (|HasCategory| |#1| ' (|SemiGroup|))
      (|HasCategory| |#1| ' (|SetCategory|))))

```

1.7.15 The “abbreviation”

EQ

1.7.16 The “parents”

```

(((|Type|) . T)
  ((|InnerEvalable| (|Symbol|) S) |has| S
   (|InnerEvalable| (|Symbol|) S))
  ((|CoercibleTo| (|Boolean|)) |has| S (|SetCategory|))

```

```

((|SetCategory|) |has| S (|SetCategory|))
((|AbelianSemiGroup|) |has| S (|AbelianSemiGroup|))
((|AbelianGroup|) |has| S (|AbelianGroup|))
((|SemiGroup|) |has| S (|SemiGroup|)) ((|Monoid|) |has| S (|Monoid|))
((|Group|) |has| S (|Group|)) ((|BiModule| S S) |has| S (|Ring|))
((|Ring|) |has| S (|Ring|)) ((|Module| S) |has| S (|CommutativeRing|))
((|PartialDifferentialRing| (|Symbol|)) |has| S
  (|PartialDifferentialRing| (|Symbol|)))
((|VectorSpace| S) |has| S (|Field|))

```

1.7.17 The “ancestors”

```

(((|AbelianGroup|) |has| S (|AbelianGroup|))
  ((|AbelianMonoid|) |has| S (|AbelianGroup|))
  ((|AbelianSemiGroup|) |has| S (|AbelianSemiGroup|))
  ((|BasicType|) |has| S (|SetCategory|))
  ((|BiModule| S S) |has| S (|Ring|))
  ((|CancellationAbelianMonoid|) |has| S (|AbelianGroup|))
  ((|CoercibleTo| (|OutputForm|)) |has| S (|SetCategory|))
  ((|CoercibleTo| (|Boolean|)) |has| S (|SetCategory|))
  ((|Group|) |has| S (|Group|))
  ((|InnerEvalable| (|Symbol|) S) |has| S
    (|InnerEvalable| (|Symbol|) S))
  ((|LeftModule| $) |has| S (|Ring|))
  ((|LeftModule| S) |has| S (|Ring|))
  ((|Module| S) |has| S (|CommutativeRing|))
  ((|Monoid|) |has| S (|Monoid|))
  ((|PartialDifferentialRing| (|Symbol|)) |has| S
    (|PartialDifferentialRing| (|Symbol|)))
  ((|RightModule| S) |has| S (|Ring|)) ((|Ring|) |has| S (|Ring|))
  ((|Rng|) |has| S (|Ring|)) ((|SemiGroup|) |has| S (|SemiGroup|))
  ((|SetCategory|) |has| S (|SetCategory|)) ((|Type|) . T)
  ((|VectorSpace| S) |has| S (|Field|)))

```

1.7.18 The “documentation”

```

((|constructor|
  (NIL "Equations as mathematical objects. All properties of the basis
    domain,{ } \\spadignore{e.g.} being an abelian group are carried
    over the equation domain,{ } by performing the structural operations
    on the left and on the right hand side."))
  (|subst| (($ $ $)
    "\\spad{subst(eq1,{ }eq2)} substitutes \\spad{eq2} into both sides
    of \\spad{eq1} the \\spad{lhs} of \\spad{eq2} should be a kernel"))
  (|inv| (($ $)
    "\\spad{inv(x)} returns the multiplicative inverse of \\spad{x}.")
  (/ (($ $ $)
    "\\spad{e1/e2} produces a new equation by dividing the left and right

```

```

    hand sides of equations \spad{e1} and \spad{e2}.)")
(|factorAndSplit|
  (((|List| $) $)
    "\spad{factorAndSplit(eq)} make the right hand side 0 and factors the
    new left hand side. Each factor is equated to 0 and put into the
    resulting list without repetitions."))
(|rightOne|
  (((|Union| $ "failed") $)
    "\spad{rightOne(eq)} divides by the right hand side.")
  (((|Union| $ "failed") $)
    "\spad{rightOne(eq)} divides by the right hand side,{ } if possible."))
(|leftOne|
  (((|Union| $ "failed") $)
    "\spad{leftOne(eq)} divides by the left hand side.")
  (((|Union| $ "failed") $)
    "\spad{leftOne(eq)} divides by the left hand side,{ } if possible."))
(* (($ $ |#1|)
  "\spad{eqn*x} produces a new equation by multiplying both sides of
  equation eqn by \spad{x}."
  (($ |#1| $)
    "\spad{x*eqn} produces a new equation by multiplying both sides of
    equation eqn by \spad{x}."))
(- (($ $ |#1|)
  "\spad{eqn-x} produces a new equation by subtracting \spad{x} from
  both sides of equation eqn."
  (($ |#1| $)
    "\spad{x-eqn} produces a new equation by subtracting both sides of
    equation eqn from \spad{x}."))
(|rightZero|
  (($ $) "\spad{rightZero(eq)} subtracts the right hand side."))
(|leftZero|
  (($ $) "\spad{leftZero(eq)} subtracts the left hand side."))
(+ (($ $ |#1|)
  "\spad{eqn+x} produces a new equation by adding \spad{x} to both
  sides of equation eqn."
  (($ |#1| $)
    "\spad{x+eqn} produces a new equation by adding \spad{x} to both
    sides of equation eqn."))
(|eval| (($ $ (|List| $))
  "\spad{eval(eqn,{ } [x1=v1,{ } ... xn=vn])} replaces \spad{xi}
  by \spad{vi} in equation \spad{eqn}."
  (($ $ $)
    "\spad{eval(eqn,{ } x=f)} replaces \spad{x} by \spad{f} in
    equation \spad{eqn}."))
(|map| (($ (|Mapping| |#1| |#1|) $)
  "\spad{map(f,{ }eqn)} constructs a new equation by applying
  \spad{f} to both sides of \spad{eqn}."))
(|rhs| ((|#1| $)
  "\spad{rhs(eqn)} returns the right hand side of equation
  \spad{eqn}."))

```

```
(|lhs| ((|#1| $)
  "\\spad{lhs(eqn)} returns the left hand side of equation
  \\spad{eqn}."))
(|swap| (($ $)
  "\\spad{swap(eq)} interchanges left and right hand side of
  equation \\spad{eq}."))
(|equation|
  (($ |#1| |#1|) "\\spad{equation(a,{b})} creates an equation."))
(= (($ |#1| |#1|) "\\spad{a=b} creates an equation."))
```

1.7.19 The “slotInfo”

```
(|Equation|
  (NIL (~= ((38 0 0) NIL (|has| |#1| (|SetCategory|))))
    (|zero?| ((38 0) NIL (|has| |#1| (|AbelianGroup|))))
    (|swap| ((0 0) 22))
    (|subtractIfCan| ((64 0 0) NIL (|has| |#1| (|AbelianGroup|))))
    (|subst| ((0 0 0) 93 (|has| |#1| (|ExpressionSpace|))))
    (|sample|
      ((0) NIL
        (OR (|has| |#1| (|AbelianGroup|))
          (|has| |#1| (|Monoid|))))
      CONST))
    (|rightZero| ((0 0) 8 (|has| |#1| (|AbelianGroup|))))
    (|rightOne| ((64 0) 68 (|has| |#1| (|Monoid|))))
    (|rhs| ((6 0) 21))
    (|recip| ((64 0) 66 (|has| |#1| (|Monoid|))))
    (|one?| ((38 0) NIL (|has| |#1| (|Monoid|))))
    (|map| ((0 23 0) 24) (|lhs| ((6 0) 9))
    (|leftZero| ((0 0) 57 (|has| |#1| (|AbelianGroup|))))
    (|leftOne| ((64 0) 67 (|has| |#1| (|Monoid|))))
    (|latex| ((97 0) NIL (|has| |#1| (|SetCategory|))))
    (|inv| ((0 0) 70
      (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|))))
    (|hash| ((96 0) NIL (|has| |#1| (|SetCategory|))))
    (|factorAndSplit| ((18 0) 19 (|has| |#1| (|IntegralDomain|))))
    (|eval| ((0 0 28 29) 31
      (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
      ((0 0 25 6) 27
        (|has| |#1| (|InnerEvalable| (|Symbol|) |#1|)))
      ((0 0 18) 37
        (AND (|has| |#1| (|Evalable| |#1|))
          (|has| |#1| (|SetCategory|))))
      ((0 0 0) 34
        (AND (|has| |#1| (|Evalable| |#1|))
          (|has| |#1| (|SetCategory|))))
    (|equation| ((0 6 6) 17))
    (|dimension| ((86) 88 (|has| |#1| (|Field|))))
    (|differentiate|
```

```

((0 0 25 71) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 28 95) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 25) 85
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
((0 0 28) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
(|conjugate| ((0 0 0) NIL (|has| |#1| (|Group|))))
(|commutator| ((0 0 0) NIL (|has| |#1| (|Group|))))
(|coerce| ((38 0) 45 (|has| |#1| (|SetCategory|))))
  ((41 0) 44 (|has| |#1| (|SetCategory|)))
  ((0 74) NIL (|has| |#1| (|Ring|)))
(|characteristic| ((71) 73 (|has| |#1| (|Ring|))))
(^ ((0 0 94) NIL (|has| |#1| (|SemiGroup|)))
  ((0 0 71) NIL (|has| |#1| (|Monoid|)))
  ((0 0 74) NIL (|has| |#1| (|Group|))))
(|Zero| ((0) 55 (|has| |#1| (|AbelianGroup|)) CONST))
(|One| ((0) 63 (|has| |#1| (|Monoid|)) CONST))
(D ((0 0 25 71) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 28 95) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 25) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 0 28) NIL
  (|has| |#1| (|PartialDifferentialRing| (|Symbol|))))
  ((0 6 6) 20) ((38 0 0) 40 (|has| |#1| (|SetCategory|))))
(/ ((0 0 6) NIL (|has| |#1| (|Field|)))
  ((0 0 0) 90
  (OR (|has| |#1| (|Field|)) (|has| |#1| (|Group|)))))
(- ((0 0 6) 54 (|has| |#1| (|AbelianGroup|)))
  ((0 6 0) 53 (|has| |#1| (|AbelianGroup|)))
  ((0 0 0) 52 (|has| |#1| (|AbelianGroup|)))
  ((0 0) 51 (|has| |#1| (|AbelianGroup|))))
(+ ((0 0 6) 49 (|has| |#1| (|AbelianSemiGroup|)))
  ((0 6 0) 48 (|has| |#1| (|AbelianSemiGroup|)))
  ((0 0 0) 47 (|has| |#1| (|AbelianSemiGroup|))))
(** ((0 0 94) NIL (|has| |#1| (|SemiGroup|)))
  ((0 0 71) NIL (|has| |#1| (|Monoid|)))
  ((0 0 74) NIL (|has| |#1| (|Group|))))
(* ((0 6 0) 60 (|has| |#1| (|SemiGroup|)))
  ((0 0 6) 61 (|has| |#1| (|SemiGroup|)))
  ((0 0 0) 59 (|has| |#1| (|SemiGroup|)))
  ((0 94 0) NIL (|has| |#1| (|AbelianSemiGroup|)))
  ((0 74 0) 76 (|has| |#1| (|AbelianGroup|)))
  ((0 71 0) NIL (|has| |#1| (|AbelianGroup|)))))

```

1.7.20 The “index”

```
((("slot1Info" 0 32444) ("documentation" 0 29640) ("ancestors" 0 28691)
  ("parents" 0 28077) ("abbreviation" 0 28074) ("predicates" 0 25442)
  ("attributes" 0 25304) ("signaturesAndLocals" 0 23933)
  ("superDomain" 0 NIL) ("operationAlist" 0 20053) ("modemaps" 0 17216)
  ("sourceFile" 0 17179) ("constructorCategory" 0 15220)
  ("constructorModemap" 0 13215) ("constructorKind" 0 13206)
  ("constructorForm" 0 13191) ("compilerInfo" 0 4433)
  ("loadTimeStuff" 0 20))
```


Chapter 2

Compiler top level

2.1 Global Data Structures

2.2 Pratt Parsing

Parsing involves understanding the association of symbols and operators. Vaughn Pratt [8] poses the question “Given a substring AEB where A takes a right argument, B a left, and E is an expression, does E associate with A or B?”.

Floyd [9] associates a precedence with operators, storing them in a table, called “binding powers”. The expression E would associate with the argument position having the highest binding power. This leads to a large set of numbers, one for every situation.

Pratt assigns data types to “classes” and then creates a total order on the classes. He lists, in ascending order, Outcomes, Booleans, Graphs (trees, lists, etc), Strings, Algebraics (e.g. Integer, complex numbers, polynomials, real arrays) and references (e.g. the left hand side of assignments). Thus, Strings \leq References. The key restriction is “that the class of the type at any argument that might participate in an association problem not be less than the class of the data type of the result of the function taking that argument”.

For a less-than comparison (“ $<$ ”) the argument types are Algebraics but the result type is Boolean. Since Algebraics are greater than Boolean we can associate the Algebraics together and apply them as arguments to the Boolean.

In more detail, there an “association” is a function of 4 types:

- a_A – The data type of the right argument
- r_A – The return type of the right argument
- a_B – The data type of the left argument
- r_B – The return type of the left argument

Note that the return types might depend on the type of the expression E . If all 4 are of the same class then the association is to the left.

Using these ideas and given the restriction above, Pratt proves that every association problem has at most one solution consistent with the data types of the associated operators.

Pratt proves that there exists an assignment of integers to the argument positions of each token in the language such that the correct association, if any, is always in the direction of the argument position with the larger number, with ties being broken to the left.

To construct the proper numbers, first assign even integers to the data type classes. Then to each argument position assign an integer lying strictly (where possible) between the integers corresponding to the classes of the argument and result types.

For tokens like “and”, “or”, +, *, and the Booleans and Algebras can be subdivided into pseudo-classes so that

terms < factors < primaries

Then + is defined over terms, * over factors, and over primaries with coercions allowed from primaries to factors to terms. To be consistent with Algol, the primaries should be a right associative class (e.g. xyz)

2.3)compile

This is the implementation of the)compile command.

You use this command to invoke the new Axiom library compiler or the old Axiom system compiler. The)compile system command is actually a combination of Axiom processing and a call to the Aldor compiler. It is performing double-duty, acting as a front-end to both the Aldor compiler and the old Axiom system compiler. (The old Axiom system compiler was written in Lisp and was an integral part of the Axiom environment. The Aldor compiler is written in C and executed by the operating system when called from within Axiom.)

User Level Required: compiler

Command Syntax:

```
)compile
)compile fileName
)compile fileName.spad
)compile directory/fileName.spad
)compile fileName )old
)compile fileName )translate
)compile fileName )quiet
)compile fileName )noquiet
```

```

)compile fileName )moreargs
)compile fileName )onlyargs
)compile fileName )break
)compile fileName )nobreak
)compile fileName )library
)compile fileName )nolibrary
)compile fileName )vartrace
)compile fileName )constructor nameOrAbbrev

```

These command forms invoke the Aldor compiler.

```

)compile fileName.as
)compile directory/fileName.as
)compile fileName.ao
)compile directory/fileName.ao
)compile fileName.al
)compile directory/fileName.al
)compile fileName.lsp
)compile directory/fileName.lsp
)compile fileName )new

```

Command Description:

The first thing `)compile` does is look for a source code filename among its arguments. Thus

```

)compile mycode.spad
)compile /u/jones/mycode.spad
)compile mycode

```

all invoke `)compiler` on the file `/u/jones/mycode.spad` if the current Axiom working directory is `/u/jones`. (Recall that you can set the working directory via the `)cd` command. If you don't set it explicitly, it is the directory from which you started Axiom.)

If you omit the file extension, the command looks to see if you have specified the `)new` or `)old` option. If you have given one of these options, the corresponding compiler is used.

The command first looks in the standard system directories for files with extension `.as`, `.ao` and `.al` and then files with extension `.spad`. The first file found has the appropriate compiler invoked on it. If the command cannot find a matching file, an error message is displayed and the command terminates.

The first thing `)compile` does is look for a source code filename among its arguments. Thus

```

)compile mycode
)co mycode
)co mycode.spad

```

all invoke `)compiler` on the file `/u/jones/mycode.spad` if the current Axiom working directory is `/u/jones`. Recall that you can set the working directory via the `)cd` command. If you don't set it explicitly, it is the directory from which you started Axiom.

This is frequently all you need to compile your file.

This simple command:

1. Invokes the Spad compiler and produces Lisp output.
2. Calls the Lisp compiler if the compilation was successful.
3. Uses the `)library` command to tell Axiom about the contents of your compiled file and arrange to have those contents loaded on demand.

Should you not want the `)library` command automatically invoked, call `)compile` with the `)nolibrary` option. For example,

```

)compile mycode )nolibrary

```

By default, the `)library` system command *exposes* all domains and categories it processes. This means that the Axiom interpreter will consider those domains and categories when it is trying to resolve a reference to a function. Sometimes domains and categories should not be exposed. For example, a domain may just be used privately by another domain and may not be meant for top-level use. The `)library` command should still be used, though, so that the code will be loaded on demand. In this case, you should use the `)nolibrary` option on `)compile` and the `)noexpose` option in the `)library` command. For example,

```

)compile mycode )nolibrary
)library mycode )noexpose

```

Once you have established your own collection of compiled code, you may find it handy to use the `)dir` option on the `)library` command. This causes `)library` to process all compiled code in the specified directory. For example,

```

)library )dir /u/jones/quantum

```

You must give an explicit directory after `)dir`, even if you want all compiled code in the current working directory processed, e.g.

```

)library )dir .

```

2.3.1 Spad compiler

This command compiles files with file extension `.spad` with the Spad system compiler.

The `)translate` option is used to invoke a special version of the old system compiler that will translate a `.spad` file to a `.as` file. That is, the `.spad` file will be parsed and analyzed and a file using the new syntax will be created.

By default, the `.as` file is created in the same directory as the `.spad` file. If that directory is not writable, the current directory is used. If the current directory is not writable, an error message is given and the command terminates. Note that `)translate` implies the `)old` option so the file extension can safely be omitted. If `)translate` is given, all other options are ignored. Please be aware that the translation is not necessarily one hundred percent complete or correct. You should attempt to compile the output with the Aldor compiler and make any necessary corrections.

You can compile category, domain, and package constructors contained in files with file extension `.spad`. You can compile individual constructors or every constructor in a file.

The full filename is remembered between invocations of this command and `)edit` commands. The sequence of commands

```
)compile matrix.spad
)edit
)compile
```

will call the compiler, edit, and then call the compiler again on the file **matrix.spad**. If you do not specify a *directory*, the working current directory is searched for the file. If the file is not found, the standard system directories are searched.

If you do not give any options, all constructors within a file are compiled. Each constructor should have an `)abbreviation` command in the file in which it is defined. We suggest that you place the `)abbreviation` commands at the top of the file in the order in which the constructors are defined.

The `)library` option causes directories containing the compiled code for each constructor to be created in the working current directory. The name of such a directory consists of the constructor abbreviation and the `.nrlib` file extension. For example, the directory containing the compiled code for the **MATRIX** constructor is called **MATRIX.nrlib**. The `)nolibrary` option says that such files should not be created. The default is `)library`. Note that the semantics of `)library` and `)nolibrary` for the new Aldor compiler and for the old system compiler are completely different.

The `)vartrace` option causes the compiler to generate extra code for the constructor to support conditional tracing of variable assignments. Without this option, this code is suppressed and one cannot use the `)vars` option for the trace command.

The `)constructor` option is used to specify a particular constructor to compile. All other constructors in the file are ignored. The constructor name or abbreviation follows `)constructor`. Thus either

```
)compile matrix.spad )constructor RectangularMatrix
```

or

```
)compile matrix.spad )constructor RMATRIX
```

compiles the `RectangularMatrix` constructor defined in `matrix.spad`.

The `)break` and `)nobreak` options determine what the spad compiler does when it encounters an error. `)break` is the default and it indicates that processing should stop at the first error. The value of the `)set break` variable then controls what happens.

2.4 Operator Precedence Table Initialization

```
; PURPOSE: This file sets up properties which are used by the Boot lexical
;           analyzer for bottom-up recognition of operators. Also certain
;           other character-class definitions are included, as well as
;           table accessing functions.
;
; ORGANIZATION: Each section is organized in terms of Creation and Access code.
;
;               1. Led and Nud Tables
;               2. GLIPH Table
;               3. RENAMETOK Table
;               4. GENERIC Table
;               5. Character syntax class predicates
```

2.4.1 LED and NUD Tables

```
; **** 1. LED and NUD Tables

; ** TABLE PURPOSE

; Led and Nud have to do with operators. An operator with a Led property takes
; an operand on its left (infix/suffix operator).

; An operator with a Nud takes no operand on its left (prefix/nilfix).
; Some have both (e.g. - ). This terminology is from the Pratt parser.
; The translator for Scratchpad II is a modification of the Pratt parser which
; branches to special handlers when it is most convenient and practical to
; do so (Pratt's scheme cannot handle local contexts very easily).

; Both LEDs and NUDs have right and left binding powers. This is meaningful
; for prefix and infix operators. These powers are stored as the values of
; the LED and NUD properties of an atom, if the atom has such a property.
; The format is:

;           <Operator Left-Binding-Power Right-Binding-Power <Special-Handler>>
```

```
; where the Special-Handler is the name of a function to be evaluated when that
; keyword is encountered.

; The default values of Left and Right Binding-Power are NIL. NIL is a
; legitimate value signifying no precedence. If the Special-Handler is NIL,
; this is just an ordinary operator (as opposed to a surfix operator like
; if-then-else).
;
; The Nud value gives the precedence when the operator is a prefix op.
; The Led value gives the precedence when the operator is an infix op.
; Each op has 2 priorities, left and right.
; If the right priority of the first is greater than or equal to the
; left priority of the second then collect the second operator into
; the right argument of the first operator.
```

— LEDNUDTables —

```
; ** TABLE CREATION

(defun makenewop (x y) (makeop x y '|PARSE-NewKEY|))

(defun makeop (x y keyname)
  (if (or (not (cdr x)) (numberp (second x)))
      (setq x (cons (first x) x)))
      (if (and (alpha-char-p (elt (princ-to-string (first x)) 0))
                (not (member (first x) (eval keyname))))
          (set keyname (cons (first x) (eval keyname))))
          (put (first x) y x)
          (second x)))

(setq |PARSE-NewKEY| nil) ;;list of keywords

(mapcar #'(LAMBDA(J) (MAKENEWOP J '|Led|))
  '((* 800 801) (|rem| 800 801) (|mod| 800 801)
    (|quo| 800 801) (|div| 800 801)
    (/ 800 801) (** 900 901) (^ 900 901)
    (|exquo| 800 801) (+ 700 701)
    (\- 700 701) (\-> 1001 1002) (\<- 1001 1002)
    (\: 996 997) (\:\: 996 997)
    (\@ 996 997) (|pretend| 995 996)
    (\.) (\! \! 1002 1001)
    (\, 110 111)
    (\; 81 82 (|PARSE-SemiColon|))
    (\< 400 400) (\> 400 400)
    (\<\< 400 400) (\>\> 400 400)
    (\<= 400 400) (\>= 400 400)
    (= 400 400) (^= 400 400)
    (\~= 400 400))
```

```

(|in| 400 400)      (|case| 400 400)
(|add| 400 120)     (|with| 2000 400 (|PARSE-InfixWith|))
(|has| 400 400)
(|where| 121 104)    ; must be 121 for SPAD, 126 for boot--> nboot
(|when| 112 190)
(|otherwise| 119 190 (|PARSE-Suffix|))
(|is| 400 400)      (|isnt| 400 400)
(|and| 250 251)     (|or| 200 201)
(|/\ 250 251)       (|\/ 200 201)
(|\.\. SEGMENT 401 699 (|PARSE-Seg|))
(=> 123 103)
(+-> 995 112)
(== DEF 122 121)
(==> MDEF 122 121)
(| 108 111)         ;was 190 190
(|:- LETD 125 124) (|:= LET 125 124)))

(mapcar #'(LAMBDA (J) (MAKENEWOP J '(|Nud|))
'(|for| 130 350 (|PARSE-Loop|))
(|while| 130 190 (|PARSE-Loop|))
(|until| 130 190 (|PARSE-Loop|))
(|repeat| 130 190 (|PARSE-Loop|))
(|import| 120 0 (|PARSE-Import|) )
(|unless|)
(|add| 900 120)
(|with| 1000 300 (|PARSE-With|))
(|has| 400 400)
(|- 701 700) ; right-prec. wants to be -1 + left-prec
;; (|+ 701 700)
(|# 999 998)
(|! 1002 1001)
(|' 999 999 (|PARSE-Data|))
(|<< 122 120 (|PARSE-LabelExpr|))
(|>>)
(|^ 260 259 NIL)
(|-> 1001 1002)
(|: 194 195)
(|not| 260 259 NIL)
(|~ 260 259 nil)
(|= 400 700)
(|return| 202 201 (|PARSE-Return|))
(|leave| 202 201 (|PARSE-Leave|))
(|exit| 202 201 (|PARSE-Exit|))
(|from|)
(|iterate|)
(|yield|)
(|if| 130 0 (|PARSE-Conditional|)) ; was 130
(| 0 190)
(|suchthat|)
(|then| 0 114)

```

```
(|else| 0 114)))
```

2.5 Glyph Table

Gliph is a symbol clump. The gliph property of a symbol gives the tree describing the tokens which begin with that symbol. The token reader uses the gliph property to determine the longest token. Thus `:=` is read as one token not as `:` followed by `=`.

— GLIPHTable —

```
(mapcar #'(lambda (x) (put (car x) 'gliph (cdr x)))
  '(
    ( \| (\))      )
    ( *  (*)       )
    ( \ ( (<) (\|) )
    ( +  (- (>))   )
    ( -  (>)       )
    ( <  (=) (<)   )
  ;;   ( /  (\\)      ) breaks */xxx
    ( \\ (/)       )
    ( >  (=) (>) (\))
    ( =  (= (>)) (>) )
    ( \. (\.)      )
    ( ^  (=)       )
    ( \~ (=)       )
    ( \: (=) (-) (\:)))
```

2.5.1 Rename Token Table

RENAMETOK defines alternate token strings which can be used for different keyboards which define equivalent tokens.

— RENAMETOKTable —

```
(mapcar
  #'(lambda (x) (put (car x) 'renametok (cadr x)) (makenewop x nil))
  '((\(\| \[]      ; (| |) means []
    (\|\) \])
    (\(< \{       ; (< >) means {}
    (>\) \})))
```

2.5.2 Generic function table

GENERIC operators be suffixed by \$ qualifications in SPAD code. \$ is then followed by a domain label, such as I for Integer, which signifies which domain the operator refers to. For example `+$Integer` is `+` for Integers.

— **GENERICTable** —

```
(mapcar #'(lambda (x) (put x 'generic 'true))
  '(- = * |rem| |mod| |quo| |div| / ** |exquo| + - < > <= >= ^= ))
```

2.6 Giant steps, Baby steps

We will walk through the compiler with the EQ.spad example using a Giant-steps, Baby-steps approach. That is, we will show the large scale (Giant) transformations at each stage of compilation and discuss the details (Baby) in subsequent chapters.

Chapter 3

The Parser

3.1 EQ.spad

We will explain the compilation function using the file `EQ.spad`. We trace the execution of the various functions to understand the actual call parameters and results returned. The `EQ.spad` file is:

```
)abbrev domain EQ Equation
--FOR THE BENEFIT OF LIBAXO GENERATION
++ Author: Stephen M. Watt, enhancements by Johannes Grabmeier
++ Date Created: April 1985
++ Date Last Updated: June 3, 1991; September 2, 1992
++ Basic Operations: =
++ Related Domains:
++ Also See:
++ AMS Classifications:
++ Keywords: equation
++ Examples:
++ References:
++ Description:
++ Equations as mathematical objects. All properties of the basis domain,
++ e.g. being an abelian group are carried over the equation domain, by
++ performing the structural operations on the left and on the
++ right hand side.
-- The interpreter translates "=" to "equation". Otherwise, it will
-- find a modemap for "=" in the domain of the arguments.

Equation(S: Type): public == private where
  Ex ==> OutputForm
  public ==> Type with
    "=": (S, S) -> $
    ++ a=b creates an equation.
```

```

equation: (S, S) -> $
  ++ equation(a,b) creates an equation.
swap: $ -> $
  ++ swap(eq) interchanges left and right hand side of equation eq.
lhs: $ -> S
  ++ lhs(eqn) returns the left hand side of equation eqn.
rhs: $ -> S
  ++ rhs(eqn) returns the right hand side of equation eqn.
map: (S -> S, $) -> $
  ++ map(f,eqn) constructs a new equation by applying f to both
  ++ sides of eqn.
if S has InnerEvalable(Symbol,S) then
  InnerEvalable(Symbol,S)
if S has SetCategory then
  SetCategory
  CoercibleTo Boolean
  if S has Evalable(S) then
    eval: ($, $) -> $
      ++ eval(eqn, x=f) replaces x by f in equation eqn.
    eval: ($, List $) -> $
      ++ eval(eqn, [x1=v1, ... xn=vn]) replaces xi by vi in equation eqn.
if S has AbelianSemiGroup then
  AbelianSemiGroup
  "+": (S, $) -> $
    ++ x+eqn produces a new equation by adding x to both sides of
    ++ equation eqn.
  "+": ($, S) -> $
    ++ eqn+x produces a new equation by adding x to both sides of
    ++ equation eqn.
if S has AbelianGroup then
  AbelianGroup
  leftZero : $ -> $
    ++ leftZero(eq) subtracts the left hand side.
  rightZero : $ -> $
    ++ rightZero(eq) subtracts the right hand side.
  "-": (S, $) -> $
    ++ x-eqn produces a new equation by subtracting both sides of
    ++ equation eqn from x.
  "-": ($, S) -> $
    ++ eqn-x produces a new equation by subtracting x from both sides of
    ++ equation eqn.
if S has SemiGroup then
  SemiGroup
  "*": (S, $) -> $
    ++ x*eqn produces a new equation by multiplying both sides of
    ++ equation eqn by x.
  "*": ($, S) -> $
    ++ eqn*x produces a new equation by multiplying both sides of
    ++ equation eqn by x.
if S has Monoid then

```

```

Monoid
leftOne : $ -> Union($,"failed")
  ++ leftOne(eq) divides by the left hand side, if possible.
rightOne : $ -> Union($,"failed")
  ++ rightOne(eq) divides by the right hand side, if possible.
if S has Group then
  Group
  leftOne : $ -> Union($,"failed")
    ++ leftOne(eq) divides by the left hand side.
  rightOne : $ -> Union($,"failed")
    ++ rightOne(eq) divides by the right hand side.
if S has Ring then
  Ring
  BiModule(S,S)
if S has CommutativeRing then
  Module(S)
  --Algebra(S)
if S has IntegralDomain then
  factorAndSplit : $ -> List $
    ++ factorAndSplit(eq) make the right hand side 0 and
    ++ factors the new left hand side. Each factor is equated
    ++ to 0 and put into the resulting list without repetitions.
if S has PartialDifferentialRing(Symbol) then
  PartialDifferentialRing(Symbol)
if S has Field then
  VectorSpace(S)
  "/" : ($, $) -> $
    ++ e1/e2 produces a new equation by dividing the left and right
    ++ hand sides of equations e1 and e2.
  inv : $ -> $
    ++ inv(x) returns the multiplicative inverse of x.
if S has ExpressionSpace then
  subst : ($, $) -> $
    ++ subst(eq1,eq2) substitutes eq2 into both sides of eq1
    ++ the lhs of eq2 should be a kernel

private ==> add
Rep := Record(lhs: S, rhs: S)
eq1,eq2: $
s : S
if S has IntegralDomain then
  factorAndSplit eq ==
    (S has factor : S -> Factored S) =>
      eq0 := rightZero eq
      [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
  [eq]
  l:S = r:S == [l, r]
  equation(l, r) == [l, r] -- hack! See comment above.
  lhs eqn == eqn.lhs
  rhs eqn == eqn.rhs

```

```

swap eqn      == [rhs eqn, lhs eqn]
map(fn, eqn)  == equation(fn(eqn.lhs), fn(eqn.rhs))

if S has InnerEvalable(Symbol,S) then
  s:Symbol
  ls:List Symbol
  x:S
  lx:List S
  eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x)
  eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) = eval(eqn.rhs,ls,lx)
if S has Evalable(S) then
  eval(eqn1:$, eqn2:$):$ ==
    eval(eqn1.lhs, eqn2 pretend Equation S) =
      eval(eqn1.rhs, eqn2 pretend Equation S)
  eval(eqn1:$, leqn2:List $):$ ==
    eval(eqn1.lhs, leqn2 pretend List Equation S) =
      eval(eqn1.rhs, leqn2 pretend List Equation S)
if S has SetCategory then
  eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and
               (eq1.rhs = eq2.rhs)@Boolean
  coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex
  coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs
if S has AbelianSemiGroup then
  eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs
  s + eq2 == [s,s] + eq2
  eq1 + s == eq1 + [s,s]
if S has AbelianGroup then
  - eq == (- lhs eq) = (-rhs eq)
  s - eq2 == [s,s] - eq2
  eq1 - s == eq1 - [s,s]
  leftZero eq == 0 = rhs eq - lhs eq
  rightZero eq == lhs eq - rhs eq = 0
  0 == equation(0$S,0$S)
  eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs
if S has SemiGroup then
  eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  1:S * eqn:$ == 1 * eqn.lhs = 1 * eqn.rhs
  eqn:$ * 1:S == eqn.lhs * 1 = eqn.rhs * 1
  -- We have to be a bit careful here: raising to a +ve integer is OK
  -- (since it's the equivalent of repeated multiplication)
  -- but other powers may cause contradictions
  -- Watch what else you add here! JHD 2/Aug 1990
if S has Monoid then
  1 == equation(1$S,1$S)
  recip eq ==
    (lh := recip lhs eq) case "failed" => "failed"
    (rh := recip rhs eq) case "failed" => "failed"
    [lh :: S, rh :: S]
  leftOne eq ==

```

```

      (re := recip lhs eq) case "failed" => "failed"
      1 = rhs eq * re
    rightOne eq ==
      (re := recip rhs eq) case "failed" => "failed"
      lhs eq * re = 1
  if S has Group then
    inv eq == [inv lhs eq, inv rhs eq]
    leftOne eq == 1 = rhs eq * inv rhs eq
    rightOne eq == lhs eq * inv rhs eq = 1
  if S has Ring then
    characteristic() == characteristic()$S
    i:Integer * eq:$ == (i::S) * eq
  if S has IntegralDomain then
    factorAndSplit eq ==
      (S has factor : S -> Factored S) =>
        eq0 := rightZero eq
        [equation(rcf.factor,0) for rcf in factors factor lhs eq0]
      (S has Polynomial Integer) =>
        eq0 := rightZero eq
        MF ==> MultivariateFactorize(Symbol, IndexedExponents Symbol, _
          Integer, Polynomial Integer)
        p : Polynomial Integer := (lhs eq0) pretend Polynomial Integer
        [equation((rcf.factor) pretend S,0) for rcf in factors factor(p)$MF]
      [eq]
  if S has PartialDifferentialRing(Symbol) then
    differentiate(eq:$, sym:Symbol):$ ==
      [differentiate(lhs eq, sym), differentiate(rhs eq, sym)]
  if S has Field then
    dimension() == 2 :: CardinalNumber
    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs
    inv eq == [inv lhs eq, inv rhs eq]
  if S has ExpressionSpace then
    subst(eq1,eq2) ==
      eq3 := eq2 pretend Equation S
      [subst(lhs eq1,eq3),subst(rhs eq1,eq3)]

```

3.2 preparse

The first large transformation of this input occurs in the function `preparse`. The `preparse` function reads the source file and breaks the input into a list of pairs. The first part of the pair is the line number of the input file and the second part of the pair is the actual source text as a string.

One feature that is the added semicolons at the end of the strings where the “pile” structure of the code has been converted to a semicolon delimited form.

3.2.1 defvar \$index

— initvars —

```
(defvar $index 0 "File line number of most recently read line")
```

—————

3.2.2 defvar \$linelist

— initvars —

```
(defvar $linelist nil "Stack of preparsed lines")
```

—————

3.2.3 defvar \$echolinestack

— initvars —

```
(defvar $echolinestack nil "Stack of lines to list")
```

—————

3.2.4 defvar \$preparse-last-line

— initvars —

```
(defvar $preparse-last-line nil "Most recently read line")
```

—————

3.3 Parsing routines

The **initialize-preparse** expects to be called before the **preparse** function. It initializes the state, in particular, it reads a single line from the input stream and stores it in

`$preparse-last-line`. The caller gives a stream and the `$preparse-last-line` variable is initialized as:

```
2> (INITIALIZE-PREPARSE #<input stream "/tmp/EQ.spad">)
<2 (INITIALIZE-PREPARSE ")abbrev domain EQ Equation")
```

3.3.1 defun initialize-preparse

```
[get-a-line p608]
[$index p72]
[$linelist p72]
[$echolinestack p72]
[$preparse-last-line p72]
```

— defun initialize-preparse —

```
(defun initialize-preparse (strm)
  (setq $index 0)
  (setq $linelist nil)
  (setq $echolinestack nil)
  (setq $preparse-last-line (get-a-line strm)))
```

The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```
2> (PREPARSE #<input stream "/tmp/EQ.spad">)
3> (PREPARSE1 (")abbrev domain EQ Equation"))
4> (|doSystemCommand| "abbrev domain EQ Equation")
<4 (|doSystemCommand| NIL)
<3 (PREPARSE1 ( ...[snip]... )
<2 (PREPARSE (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . " public ==> Type with")
(22 . " (\ "=": (S, S) -> $;")
(24 . " equation: (S, S) -> $;")
(26 . " swap: $ -> $;")
(28 . " lhs: $ -> S;")
(30 . " rhs: $ -> S;")
(32 . " map: (S -> S, $) -> $;")
(35 . " if S has InnerEvalable(Symbol,S) then")
(36 . " InnerEvalable(Symbol,S);")
(37 . " if S has SetCategory then")
(38 . " (SetCategory;")
(39 . " CoercibleTo Boolean;")
```

```

(40 . "      if S has Evaluable(S) then")
(41 . "      (eval: ($, $) -> $;")
(43 . "      eval: ($, List $) -> $));")
(45 . "  if S has AbelianSemiGroup then")
(46 . "    (AbelianSemiGroup;")
(47 . "    \"+\": (S, $) -> $;")
(50 . "    \"+\": ($, S) -> $;")
(53 . "  if S has AbelianGroup then")
(54 . "    (AbelianGroup;")
(55 . "    leftZero : $ -> $;")
(57 . "    rightZero : $ -> $;")
(59 . "    \"-\": (S, $) -> $;")
(62 . "    \"-\": ($, S) -> $;")
(65 . "  if S has SemiGroup then")
(66 . "    (SemiGroup;")
(67 . "    \": (S, $) -> $;")
(70 . "    \": ($, S) -> $;")
(73 . "  if S has Monoid then")
(74 . "    (Monoid;")
(75 . "    leftOne : $ -> Union($,\"failed\");")
(77 . "    rightOne : $ -> Union($,\"failed\");")
(79 . "  if S has Group then")
(80 . "    (Group;")
(81 . "    leftOne : $ -> Union($,\"failed\");")
(83 . "    rightOne : $ -> Union($,\"failed\");")
(85 . "  if S has Ring then")
(86 . "    (Ring;")
(87 . "    BiModule(S,S));")
(88 . "  if S has CommutativeRing then")
(89 . "    Module(S;")
(91 . "  if S has IntegralDomain then")
(92 . "    factorAndSplit : $ -> List $;")
(96 . "  if S has PartialDifferentialRing(Symbol) then")
(97 . "    PartialDifferentialRing(Symbol);")
(98 . "  if S has Field then")
(99 . "    (VectorSpace(S;")
(100 . "    \"/\": ($, $) -> $;")
(103 . "    inv: $ -> $;")
(105 . "  if S has ExpressionSpace then")
(106 . "    subst: ($, $) -> $;")
(109 . "  private ==> add")
(110 . "    (Rep := Record(lhs: S, rhs: S);")
(111 . "    eq1,eq2: $;")
(112 . "    s : S;")
(113 . "    if S has IntegralDomain then")
(114 . "      factorAndSplit eq ==")
(115 . "        ((S has factor : S -> Factored S) ==>")
(116 . "          (eq0 := rightZero eq;")
(117 . "            [equation(rcf.factor,0)
              for rcf in factors factor lhs eq0]));")

```

```

(118 . "      [eq]);")
(119 . "    l:S = r:S      == [l, r];")
(120 . "    equation(l, r) == [l, r];")
(121 . "    lhs eqn          == eqn.lhs;")
(122 . "    rhs eqn           == eqn.rhs;")
(123 . "    swap eqn          == [rhs eqn, lhs eqn];")
(124 . "    map(fn, eqn)       == equation(fn(eqn.lhs), fn(eqn.rhs));")
(125 . "    if S has InnerEvalable(Symbol,S) then")
(126 . "      (s:Symbol;")
(127 . "      ls:List Symbol;")
(128 . "      x:S;")
(129 . "      lx:List S;")
(130 . "      eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x);")
(131 . "      eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) =
                                eval(eqn.rhs,ls,lx));")
(132 . "    if S has Evalable(S) then")
(133 . "      (eval(eqn1:$, eqn2:$):$ ==")
(134 . "        eval(eqn1.lhs, eqn2 pretend Equation S) ==")
(135 . "          eval(eqn1.rhs, eqn2 pretend Equation S);")
(136 . "      eval(eqn1:$, leqn2:List $):$ ==")
(137 . "        eval(eqn1.lhs, leqn2 pretend List Equation S) ==")
(138 . "          eval(eqn1.rhs, leqn2 pretend List Equation S));")
(139 . "    if S has SetCategory then")
(140 . "      (eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and")
(141 . "        (eq1.rhs = eq2.rhs)@Boolean;")
(142 . "      coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex;")
(143 . "      coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs;")
(144 . "    if S has AbelianSemiGroup then")
(145 . "      (eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs;")
(146 . "      s + eq2 == [s,s] + eq2;")
(147 . "      eq1 + s == eq1 + [s,s]);")
(148 . "    if S has AbelianGroup then")
(149 . "      (- eq == (- lhs eq) = (-rhs eq);")
(150 . "      s - eq2 == [s,s] - eq2;")
(151 . "      eq1 - s == eq1 - [s,s];")
(152 . "      leftZero eq == 0 = rhs eq - lhs eq;")
(153 . "      rightZero eq == lhs eq - rhs eq = 0;")
(154 . "      0 == equation(0$S,0$S);")
(155 . "      eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs;")
(156 . "    if S has SemiGroup then")
(157 . "      (eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs;")
(158 . "      l:S * eqn:$ == l * eqn.lhs = l * eqn.rhs;")
(159 . "      l:S * eqn:$ == l * eqn.lhs = l * eqn.rhs;")
(160 . "      eqn:$ * l:S == eqn.lhs * l = eqn.rhs * l;")
(165 . "    if S has Monoid then")
(166 . "      (1 == equation(1$S,1$S);")
(167 . "      recip eq ==")
(168 . "        ((lh := recip lhs eq) case \"failed\" => \"failed\");")
(169 . "        (rh := recip rhs eq) case \"failed\" => \"failed\");")
(170 . "      [lh :: S, rh :: S]);")

```

```

(171 . "      leftOne eq ==")
(172 . "      ((re := recip lhs eq) case \"failed\" => \"failed\");")
(173 . "      1 = rhs eq * re);")
(174 . "      rightOne eq ==")
(175 . "      ((re := recip rhs eq) case \"failed\" => \"failed\");")
(176 . "      lhs eq * re = 1));")
(177 . "  if S has Group then")
(178 . "    (inv eq == [inv lhs eq, inv rhs eq];")
(179 . "    leftOne eq == 1 = rhs eq * inv rhs eq;")
(180 . "    rightOne eq == lhs eq * inv rhs eq = 1);")
(181 . "  if S has Ring then")
(182 . "    (characteristic() == characteristic()$S;")
(183 . "    i:Integer * eq:$ == (i::S) * eq);")
(184 . "  if S has IntegralDomain then")
(185 . "    factorAndSplit eq ==")
(186 . "    ((S has factor : S -> Factored S) =>")
(187 . "    (eq0 := rightZero eq;")
(188 . "    [equation(rcf.factor,0)
      for rcf in factors factor lhs eq0]));")
(189 . "    (S has Polynomial Integer) =>")
(190 . "    (eq0 := rightZero eq;")
(191 . "    MF ==> MultivariateFactorize(Symbol,
      IndexedExponents Symbol,
      Integer, Polynomial Integer);")
(193 . "    p : Polynomial Integer :=
      (lhs eq0) pretend Polynomial Integer;")
(194 . "    [equation((rcf.factor) pretend S,0)
      for rcf in factors factor(p)$MF]);")
(195 . "    [eq]);")
(196 . "  if S has PartialDifferentialRing(Symbol) then")
(197 . "    differentiate(eq:$, sym:Symbol):$ ==")
(198 . "    [differentiate(lhs eq, sym), differentiate(rhs eq, sym)];")
(199 . "  if S has Field then")
(200 . "    (dimension() == 2 :: CardinalNumber;")
(201 . "    eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs;")
(202 . "    inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "  if S has ExpressionSpace then")
(204 . "    subst(eq1,eq2) ==")
(205 . "    (eq3 := eq2 pretend Equation S;")
(206 . "    [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))")

```

3.3.2 defun preparse

```

[preparse p76]
[preparse1 p81]
[parseprint p528]
[ifcar p??]
[$comblocklist p525]

```

```

[ $skipme p?? ]
[ $preparse-last-line p72 ]
[ $index p72 ]
[ $docList p?? ]
[ $preparseReportIfTrue p?? ]
[ $headerDocumentation p?? ]
[ $maxSignatureLineNumber p?? ]
[ $constructorLineNumber p?? ]

```

— **defun preparse** —

```

(defun preparse (strm &aux (stack ()))
  (declare (special $comblocklist $skipme $preparse-last-line $index |$docList|
                    $preparseReportIfTrue |$headerDocumentation|
                    |$maxSignatureLineNumber| |$constructorLineNumber|))
  (setq $comblocklist nil)
  (setq $skipme nil)
  (when $preparse-last-line
    (if (consp $preparse-last-line)
        (setq stack $preparse-last-line)
        (push $preparse-last-line stack))
    (setq $index (- $index (length stack))))
  (let ((u (preparse1 stack)))
    (if $skipme
        (preparse strm)
        (progn
          (when $preparseReportIfTrue (parseprint u))
          (setq |$headerDocumentation| nil)
          (setq |$docList| nil)
          (setq |$maxSignatureLineNumber| 0)
          (setq |$constructorLineNumber| (ifcar (ifcar u)))
          u))))

```

The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```

2> (PREPARSE #<input stream "/tmp/EQ.spad">)
3> (PREPARSE1 ("abbrev domain EQ Equation"))
4> (|doSystemCommand| "abbrev domain EQ Equation")
<4 (|doSystemCommand| NIL)
<3 (PREPARSE1 (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . "   public ==> Type with")
(22 . "   (\ "=": (S, S) -> $;")
(24 . "   equation: (S, S) -> $;")

```

```

(26 . "      swap: $ -> $;")
(28 . "      lhs: $ -> S;")
(30 . "      rhs: $ -> S;")
(32 . "      map: (S -> S, $) -> $;")
(35 . "      if S has InnerEvalable(Symbol,S) then")
(36 . "          InnerEvalable(Symbol,S);")
(37 . "      if S has SetCategory then")
(38 . "          (SetCategory;")
(39 . "              CoercibleTo Boolean;")
(40 . "              if S has Evalable(S) then")
(41 . "                  (eval: ($, $) -> $;")
(43 . "                      eval: ($, List $) -> $);")
(45 . "      if S has AbelianSemiGroup then")
(46 . "          (AbelianSemiGroup;")
(47 . "              \"+\": (S, $) -> $;")
(50 . "              \"+\": ($, S) -> $);")
(53 . "      if S has AbelianGroup then")
(54 . "          (AbelianGroup;")
(55 . "              leftZero : $ -> $;")
(57 . "              rightZero : $ -> $;")
(59 . "              \"-\": (S, $) -> $;")
(62 . "              \"-\": ($, S) -> $);")
(65 . "      if S has SemiGroup then")
(66 . "          (SemiGroup;")
(67 . "              \"*\": (S, $) -> $;")
(70 . "              \"*\": ($, S) -> $);")
(73 . "      if S has Monoid then")
(74 . "          (Monoid;")
(75 . "              leftOne : $ -> Union($,\"failed\");")
(77 . "              rightOne : $ -> Union($,\"failed\");")
(79 . "      if S has Group then")
(80 . "          (Group;")
(81 . "              leftOne : $ -> Union($,\"failed\");")
(83 . "              rightOne : $ -> Union($,\"failed\");")
(85 . "      if S has Ring then")
(86 . "          (Ring;")
(87 . "              BiModule(S,S);")
(88 . "      if S has CommutativeRing then")
(89 . "          Module(S);")
(91 . "      if S has IntegralDomain then")
(92 . "          factorAndSplit : $ -> List $;")
(96 . "      if S has PartialDifferentialRing(Symbol) then")
(97 . "          PartialDifferentialRing(Symbol);")
(98 . "      if S has Field then")
(99 . "          (VectorSpace(S);")
(100 . "              \"/\": ($, $) -> $;")
(103 . "              inv: $ -> $);")
(105 . "      if S has ExpressionSpace then")
(106 . "          subst: ($, $) -> $;")
(109 . "      private ==> add")

```

```

(110 . " (Rep := Record(lhs: S, rhs: S);")
(111 . "   eq1,eq2: $;")
(112 . "   s : S;")
(113 . "   if S has IntegralDomain then")
(114 . "       factorAndSplit eq ==")
(115 . "           ((S has factor : S -> Factored S) =>)")
(116 . "           (eq0 := rightZero eq;")
(117 . "               [equation(rcf.factor,0)
                   for rcf in factors factor lhs eq0]));")
(118 . "       [eq]);")
(119 . "   l:S = r:S == [l, r];")
(120 . "   equation(l, r) == [l, r];")
(121 . "   lhs eqn == eqn.lhs;")
(122 . "   rhs eqn == eqn.rhs;")
(123 . "   swap eqn == [rhs eqn, lhs eqn];")
(124 . "   map(fn, eqn) == equation(fn(eqn.lhs), fn(eqn.rhs));")
(125 . "   if S has InnerEvalable(Symbol,S) then")
(126 . "       (s:Symbol;")
(127 . "       ls:List Symbol;")
(128 . "       x:S;")
(129 . "       lx:List S;")
(130 . "       eval(eqn,s,x) == eval(eqn.lhs,s,x) = eval(eqn.rhs,s,x);")
(131 . "       eval(eqn,ls,lx) == eval(eqn.lhs,ls,lx) =
                               eval(eqn.rhs,ls,lx));")
(132 . "   if S has Evalable(S) then")
(133 . "       (eval(eqn1:$, eqn2:$):$ ==")
(134 . "           eval(eqn1.lhs, eqn2 pretend Equation S) ==")
(135 . "           eval(eqn1.rhs, eqn2 pretend Equation S));")
(136 . "       eval(eqn1:$, leqn2:List $):$ ==")
(137 . "           eval(eqn1.lhs, leqn2 pretend List Equation S) ==")
(138 . "           eval(eqn1.rhs, leqn2 pretend List Equation S));")
(139 . "   if S has SetCategory then")
(140 . "       (eq1 = eq2 == (eq1.lhs = eq2.lhs)@Boolean and")
(141 . "           (eq1.rhs = eq2.rhs)@Boolean;")
(142 . "       coerce(eqn:$):Ex == eqn.lhs::Ex = eqn.rhs::Ex;")
(143 . "       coerce(eqn:$):Boolean == eqn.lhs = eqn.rhs;")
(144 . "   if S has AbelianSemiGroup then")
(145 . "       (eq1 + eq2 == eq1.lhs + eq2.lhs = eq1.rhs + eq2.rhs;")
(146 . "       s + eq2 == [s,s] + eq2;")
(147 . "       eq1 + s == eq1 + [s,s]);")
(148 . "   if S has AbelianGroup then")
(149 . "       (- eq == (- lhs eq) = (-rhs eq);")
(150 . "       s - eq2 == [s,s] - eq2;")
(151 . "       eq1 - s == eq1 - [s,s];")
(152 . "       leftZero eq == 0 = rhs eq - lhs eq;")
(153 . "       rightZero eq == lhs eq - rhs eq = 0;")
(154 . "       0 == equation(0$S,0$S);")
(155 . "       eq1 - eq2 == eq1.lhs - eq2.lhs = eq1.rhs - eq2.rhs;")
(156 . "   if S has SemiGroup then")
(157 . "       (eq1:$ * eq2:$ == eq1.lhs * eq2.lhs = eq1.rhs * eq2.rhs;")

```

```

(158 . "      l:S * eqn:$ == l      * eqn.lhs = l      * eqn.rhs;")
(159 . "      l:S * eqn:$ == l * eqn.lhs      =      l * eqn.rhs;")
(160 . "      eqn:$ * l:S == eqn.lhs * l      =      eqn.rhs * l;")
(165 . "      if S has Monoid then")
(166 . "      (1 == equation(1$S,1$S);")
(167 . "      recip eq ==")
(168 . "      ((lh := recip lhs eq) case \"failed\" => \"failed\");")
(169 . "      (rh := recip rhs eq) case \"failed\" => \"failed\");")
(170 . "      [lh :: S, rh :: S]);")
(171 . "      leftOne eq ==")
(172 . "      ((re := recip lhs eq) case \"failed\" => \"failed\");")
(173 . "      1 = rhs eq * re;")
(174 . "      rightOne eq ==")
(175 . "      ((re := recip rhs eq) case \"failed\" => \"failed\");")
(176 . "      lhs eq * re = 1));")
(177 . "      if S has Group then")
(178 . "      (inv eq == [inv lhs eq, inv rhs eq];")
(179 . "      leftOne eq == 1 = rhs eq * inv rhs eq;")
(180 . "      rightOne eq == lhs eq * inv rhs eq = 1);")
(181 . "      if S has Ring then")
(182 . "      (characteristic() == characteristic()$S;")
(183 . "      i:Integer * eq:$ == (i::S) * eq;")
(184 . "      if S has IntegralDomain then")
(185 . "      factorAndSplit eq ==")
(186 . "      ((S has factor : S -> Factored S) =>")
(187 . "      (eq0 := rightZero eq;")
(188 . "      [equation(rcf.factor,0)
      for rcf in factors factor lhs eq0]));")
(189 . "      (S has Polynomial Integer) =>")
(190 . "      (eq0 := rightZero eq;")
(191 . "      MF ==> MultivariateFactorize(Symbol,
      IndexedExponents Symbol,
      Integer, Polynomial Integer);")
(193 . "      p : Polynomial Integer :=
      (lhs eq0) pretend Polynomial Integer;")
(194 . "      [equation((rcf.factor) pretend S,0)
      for rcf in factors factor(p)$MF]);")
(195 . "      [eq]);")
(196 . "      if S has PartialDifferentialRing(Symbol) then")
(197 . "      differentiate(eq:$, sym:Symbol):$ ==")
(198 . "      [differentiate(lhs eq, sym), differentiate(rhs eq, sym)];")
(199 . "      if S has Field then")
(200 . "      (dimension() == 2 :: CardinalNumber;")
(201 . "      eq1:$ / eq2:$ == eq1.lhs / eq2.lhs = eq1.rhs / eq2.rhs;")
(202 . "      inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "      if S has ExpressionSpace then")
(204 . "      subst(eq1,eq2) ==")
(205 . "      (eq3 := eq2 pretend Equation S;")
(206 . "      [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))))

```

3.3.3 defun Build the lines from the input for piles

The READLOOP calls `preparseReadLine` which returns a pair of the form

```
(number . string)
```

```
[preparseReadLine p85]
[preparse-echo p88]
[fincomblock p526]
[parsepiles p84]
[preparse1 doSystemCommand (vol5)]
[escaped p525]
[indent-pos p526]
[make-full-cvec p??]
[maxindex p??]
[preparse1 strposl (vol5)]
[is-console p527]
[spad-reader p??]
[$echolinestack p72]
[$byConstructors p599]
[$skipme p??]
[$constructorsSeen p599]
[$preparse-last-line p72]
[$preparse-last-line p72]
[$index p72]
[$index p72]
[$linelist p72]
[$in-stream p??]
```

— **defun preparse1** —

```
(defun preparse1 (linelist)
  (labels (
    (isSystemCommand (line)
      (and (> (length line) 0) (eq (char line 0) #\ ) )))
    (executeSystemCommand (line)
      (catch 'spad_reader (|doSystemCommand| (subseq line 1))))
  )
  (prog (($linelist linelist) $echolinestack num line i l psloc
    instring pcount comsym strsym oparsym cparsym n ncomsym tmp1
    (sloc -1) continue (parenlev 0) ncomblock lines locs nums functor)
    (declare (special $linelist $echolinestack |$byConstructors| $skipme
      |$constructorsSeen| $preparse-last-line $index in-stream))
  READLOOP
    (setq tmp1 (preparseReadLine linelist))
    (setq num (car tmp1))
    (setq line (cdr tmp1))
    (unless (stringp line)
```

```

(preparse-echo linelist)
(cond
  ((null lines) (return nil))
  (ncomblock (fincomblock nil nums locs ncomblock nil)))
(return
  (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines))))))
(when (and (null lines) (isSystemCommand line))
  (preparse-echo linelist)
  (setq $preparse-last-line nil) ;don't reread this line
  (executeSystemCommand line)
  (go READLOOP))
(setq l (length line))
; if we get a null line, read the next line
(when (eq l 0) (go READLOOP))
; otherwise we have to parse this line
(setq psloc sloc)
(setq i 0)
(setq instring nil)
(setq pcount 0)
STRLOOP ;; handle things that need ignoring, quoting, or grouping
; are we in a comment, quoting, or grouping situation?
(setq strsym (or (position #" line :start i ) 1))
(setq comsym (or (search "--" line :start2 i ) 1))
(setq ncomsym (or (search "++" line :start2 i ) 1))
(setq oparsym (or (position #\ ( line :start i ) 1))
(setq cparsym (or (position #\ ) line :start i ) 1))
(setq n (min strsym comsym ncomsym oparsym cparsym))
(cond
  ; nope, we found no comment, quoting, or grouping
  ((= n 1) (go NOCOMS))
  ((escaped line n))
  ; scan until we hit the end of the string
  ((= n strsym) (setq instring (not instring)))
  ; we are in a string, just continue looping
  (instring)
  ;; handle -- comments by ignoring them
  ((= n comsym)
   (setq line (subseq line 0 n))
   (go NOCOMS)) ; discard trailing comment
  ;; handle ++ comments by chunking them together
  ((= n ncomsym)
   (setq sloc (indent-pos line))
   (cond
     ((= sloc n)
      (when (and ncomblock (not (= n (car ncomblock))))
        (fincomblock num nums locs ncomblock linelist)
        (setq ncomblock nil))
      (setq ncomblock (cons n (cons line (ifcdr ncomblock))))
      (setq line ""))
     (t

```

```

    (push (strconc (make-full-cvec n " ") (substring line n ())) $linelist)
    (setq $index (1- $index))
    (setq line (subseq line 0 n))))
  (go NOCOMS))
; know how deep we are into parens
((= n oparsym) (setq pcount (1+ pcount)))
((= n cparsym) (setq pcount (1- pcount)))
(setq i (1+ n))
(go STRLOOP)
NOCOMS
; remember the indentation level
(setq sloc (indent-pos line))
(setq line (string-right-trim " " line))
(when (null sloc)
  (setq sloc psloc)
  (go READLOOP))
; handle line that ends in a continuation character
(cond
  ((eq (elt line (maxindex line)) #\_)
   (setq continue t)
   (setq line (subseq line (maxindex line))))
  ((setq continue nil)))
; test for skipping constructors
(when (and (null lines) (= sloc 0))
  (if (and |$byConstructors|
        (null (search "==" line))
        (not
         (member
          (setq functor
                (intern (substring line 0 (strpos1 ": (" line 0 nil))))
          |$byConstructors|)))
      (setq $skipme 't)
      (progn
        (push functor |$constructorsSeen|)
        (setq $skipme nil))))
; is this thing followed by ++ comments?
(when (and lines (eql sloc 0))
  (when (and ncomblock (not (zerop (car ncomblock))))
    (fincomblock num nums locs ncomblock linelist))
  (when (not (is-console in-stream))
    (setq $preparse-last-line (nreverse $echolinestack)))
  (return
   (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines)))))
(when (> parenlev 0)
  (push nil locs)
  (setq sloc psloc)
  (go REREAD))
(when ncomblock
  (fincomblock num nums locs ncomblock linelist)
  (setq ncomblock ()))

```

```

      (push sloc locs)
REREAD
      (preparse-echo linelist)
      (push line lines)
      (push num nums)
      (setq parenlev (+ parenlev pcount))
      (when (and (is-console in-stream) (not continue))
        (setq $preparse-last-line nil)
        (return
         (pair (nreverse nums) (parsepiles (nreverse locs) (nreverse lines))))))
      (go READLOOP))))

```

3.3.4 defun parsepiles

Add parens and semis to lines to aid parsing. [add-parens-and-semis-to-line p84]

— defun parsepiles —

```

(defun parsepiles (locs lines)
  (mapl #'add-parens-and-semis-to-line
        (nconc lines '(" ")) (nconc locs '(nil)))
  lines)

```

3.3.5 defun add-parens-and-semis-to-line

The line to be worked on is (CAR SLINES). It's indentation is (CAR SLOCS). There is a notion of current indentation. Then:

- Add open paren to beginning of following line if following line's indentation is greater than current, and add close paren to end of last succeeding line with following line's indentation.
- Add semicolon to end of line if following line's indentation is the same.
- If the entire line consists of the single keyword then or else, leave it alone."

```

[infxtok p527]
[drop p525]
[addclose p524]
[nonblankloc p528]

```

— defun add-parens-and-semis-to-line —

```

(defun add-parens-and-semis-to-line (slines slocs)
  (let ((start-column (car slocs)))
    (when (and start-column (> start-column 0))
      (let ((count 0) (i 0))
        (seq
         (mapl #'(lambda (next-lines nlocs)
                   (let ((next-line (car next-lines)) (next-column (car nlocs)))
                     (incf i)
                     (when next-column
                      (setq next-column (abs next-column))
                      (when (< next-column start-column) (exit nil))
                      (cond
                       ((and (eq next-column start-column)
                            (rplaca nlocs (- (car nlocs)))
                            (not (infixtok next-line)))
                        (setq next-lines (drop (1- i) slines))
                        (rplaca next-lines (addclose (car next-lines) #\;))
                        (setq count (1+ count)))))))
                    (cdr slines) (cdr slocs)))
         (when (> count 0)
          (setf (char (car slines) (1- (nonblankloc (car slines)))) #\()
          (setq slines (drop (1- i) slines))
          (rplaca slines (addclose (car slines) #\) ))))))))

```

3.3.6 defun preparsedReadLine

```

[dcq p??]
[preparsedReadLine1 p87]
[initial-substring p607]
[string2BootTree p??]
[storeblanks p607]
[skip-to-endif p528]
[preparsedReadLine p85]
[$*eof* p??]

```

— defun preparsedReadLine —

```

(defun preparsedReadLine (x)
  (let (line ind tmp1)
    (declare (special *eof*))
    (setq tmp1 (preparsedReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
     ((not (stringp line)) (cons ind line))

```

```

((zerop (size line)) (cons ind line))
((char= (elt line 0) #\ )
 (cond
  ((initial-substring ")if" line)
   (if (eval (|string2BootTree| (storeblanks line 3)))
       (preparseReadLine x)
       (skip-ifblock x)))
  ((initial-substring ")elseif" line) (skip-to-endif x))
  ((initial-substring ")else" line) (skip-to-endif x))
  ((initial-substring ")endif" line) (preparseReadLine x))
  ((initial-substring ")fin" line)
   (setq *eof* t)
   (cons ind nil))))
(cons ind line))

```

3.3.7 defun skip-ifblock

```

[preparseReadLine1 p87]
[skip-ifblock p86]
[initial-substring p607]
[string2BootTree p??]
[storeblanks p607]

```

— defun skip-ifblock —

```

(defun skip-ifblock (x)
  (let (line ind tmp1)
    (setq tmp1 (preparseReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
     ((not (stringp line))
      (cons ind line))
     ((zerop (size line))
      (skip-ifblock x))
     ((char= (elt line 0) #\ )
      (cond
       ((initial-substring ")if" line)
        (cond
         ((eval (|string2BootTree| (storeblanks line 3)))
          (preparseReadLine X))
         (t (skip-ifblock x))))
       ((initial-substring ")elseif" line)
        (cond
         ((eval (|string2BootTree| (storeblanks line 7)))

```

```

      (preparseReadLine X))
    (t (skip-ifblock x))))
  ((initial-substring ")else" line)
    (preparseReadLine x))
  ((initial-substring ")endif" line)
    (preparseReadLine x))
  ((initial-substring ")fin" line)
    (cons ind nil))))
  (t (skip-ifblock x))))

```

3.3.8 defun preparseReadLine1

```

[get-a-line p608]
[expand-tabs p??]
[maxindex p??]
[strconc p??]
[preparseReadLine1 p87]
[$linelist p72]
[$preparse-last-line p72]
[$index p72]
[$EchoLineStack p??]

```

— defun preparseReadLine1 —

```

(defun preparseReadLine1 ()
  (labels (
    (accumulateLinesWithTrailingEscape (line)
      (let (ind)
        (declare (special $preparse-last-line))
        (if (and (> (setq ind (maxindex line)) -1) (char= (elt line ind) #\_))
            (setq $preparse-last-line
              (strconc (substring line 0 ind) (cdr (preparseReadLine1))))
            line))))
    (let (line)
      (declare (special $linelist $preparse-last-line $index $EchoLineStack))
      (setq line
        (if $linelist
            (pop $linelist)
            (expand-tabs (get-a-line in-stream))))
      (setq $preparse-last-line line)
      (if (stringp line)
          (progn
            (incf $index) ;; $index is the current line number
            (setq line (string-right-trim " " line))
            (push (copy-seq line) $EchoLineStack)

```

```
(cons $index (accumulateLinesWithTrailingEscape line)))
(cons $index line))))
```

3.4 I/O Handling

3.4.1 defun preparse-echo

```
[Echo-Meta p??]
[$EchoLineStack p??]
```

— defun preparse-echo —

```
(defun preparse-echo (linelist)
  (declare (special $EchoLineStack Echo-Meta) (ignore linelist))
  (if Echo-Meta
    (dolist (x (reverse $EchoLineStack))
      (format out-stream "~&;~A~%" x)))
  (setq $EchoLineStack ()))
```

3.4.2 Parsing stack

3.4.3 defstruct \$stack

— initvars —

```
(defstruct stack "A stack"
  (store nil) ; contents of the stack
  (size 0) ; number of elements in Store
  (top nil) ; first element of Store
  (updated nil) ; whether something has been pushed on the stack
  ) ; since this flag was last set to NIL
```

3.4.4 defun stack-load

[\$stack p88]

— defun stack-load —

```
(defun stack-load (list stack)
  (setf (stack-store stack) list)
  (setf (stack-size stack) (length list))
  (setf (stack-top stack) (car list)))
```

—————

3.4.5 defun stack-clear

[\$stack p88]

— defun stack-clear —

```
(defun stack-clear (stack)
  (setf (stack-store stack) nil)
  (setf (stack-size stack) 0)
  (setf (stack-top stack) nil)
  (setf (stack-updated stack) nil))
```

—————

3.4.6 defmacro stack-/-empty

[\$stack p88]

— defmacro stack-/-empty —

```
(defmacro stack-/-empty (stack) '(> (stack-size ,stack) 0))
```

—————

3.4.7 defun stack-push

[\$stack p88]

— defun stack-push —

```
(defun stack-push (x stack)
  (push x (stack-store stack))
  (setf (stack-top stack) x)
  (setf (stack-updated stack) t)
  (incf (stack-size stack))
  x)
```

3.4.8 defun stack-pop

[\$stack p88]

— defun stack-pop —

```
(defun stack-pop (stack)
  (let ((y (pop (stack-store stack))))
    (decf (stack-size stack))
    (setf (stack-top stack)
          (if (stack-/empty stack) (car (stack-store stack))
              y)))
```

3.4.9 Parsing token

3.4.10 defstruct \$token

A token is a Symbol with a Type. The type is either NUMBER, IDENTIFIER or SPECIAL-CHAR. NonBlank is true if the token is not preceded by a blank.

— initvars —

```
(defstruct token
  (symbol nil)
  (type nil)
  (nonblank t))
```

3.4.11 defvar \$prior-token

[\$token p90]

— initvars —

```
(defvar prior-token (make-token) "What did I see last")
```

3.4.12 defvar \$nonblank

— initvars —

```
(defvar nonblank t "Is there no blank in front of the current token.")
```

3.4.13 defvar \$current-token

Token at head of input stream. [\$token p90]

— initvars —

```
(defvar current-token (make-token))
```

3.4.14 defvar \$next-token

[\$token p90]

— initvars —

```
(defvar next-token (make-token) "Next token in input stream.")
```

3.4.15 defvar \$valid-tokens

[\$token p90]

— initvars —

```
(defvar valid-tokens 0 "Number of tokens in buffer (0, 1 or 2)")
```

3.4.16 defun token-install

[\$token p90]

— defun token-install —

```
(defun token-install (symbol type token &optional (nonblank t))
  (setf (token-symbol token) symbol)
  (setf (token-type token) type)
  (setf (token-nonblank token) nonblank)
  token)
```

3.4.17 defun token-print

[\$token p90]

— defun token-print —

```
(defun token-print (token)
  (format out-stream "(token (symbol ~S) (type ~S))~%"
    (token-symbol token) (token-type token)))
```

3.4.18 Parsing reduction**3.4.19 defstruct \$reduction**

A reduction of a rule is any S-Expression the rule chooses to stack.

— initvars —

```
(defstruct (reduction (:type list))
  (rule nil) ; Name of rule
  (value nil))
```

Chapter 4

Parse Transformers

4.1 Direct called parse routines

4.1.1 defun parseTransform

```
[msubst p??]  
[parseTran p93]  
[$defOp p??]
```

— defun parseTransform —

```
(defun |parseTransform| (x)  
  (let (|$defOp|)  
    (declare (special |$defOp|))  
    (setq |$defOp| nil)  
    (setq x (msubst '$ '% x)) ; for new compiler compatibility  
    (|parseTran| x)))
```

—————

4.1.2 defun parseTran

```
[parseAtom p94]  
[parseConstruct p95]  
[parseTran p93]  
[parseTranList p95]  
[getl p??]  
[$op p??]
```

— defun parseTran —

```

(defun |parseTran| (x)
  (labels (
    (g (op)
      (let (tmp1 tmp2 x)
        (seq
          (if (and (consp op) (eq (qfirst op) '|elt|))
              (progn
                (setq tmp1 (qrest op))
                (and (consp tmp1)
                    (progn
                     (setq op (qfirst tmp1))
                     (setq tmp2 (qrest tmp1))
                     (and (consp tmp2)
                         (eq (qrest tmp2) nil)
                         (progn (setq x (qfirst tmp2)) t))))))
              (exit (g x)))
          (exit op))))))
  (let (|$op| arg1 u r fn)
    (declare (special |$op|))
    (setq |$op| nil)
    (if (atom x)
        (|parseAtom| x)
        (progn
          (setq |$op| (car x))
          (setq arg1 (cdr x))
          (setq u (g |$op|))
          (cond
            ((eq u '|construct|)
             (setq r (|parseConstruct| arg1))
             (if (and (consp |$op|) (eq (qfirst |$op|) '|elt|))
                 (cons (|parseTran| |$op|) (cdr r))
                 r))
            ((and (atom u) (setq fn (get1 u '|parseTran|)))
             (funcall fn arg1))
            (t (cons (|parseTran| |$op|) (|parseTranList| arg1)))))))

```

4.1.3 defun parseAtom

```

[parseLeave p121]
[$NoValue p??]

```

— defun parseAtom —

```

(defun |parseAtom| (x)
  (declare (special |$NoValue|))

```

```
(if (eq x '|break|)
    (|parseLeave| (list '$NoValue|))
    x))
```

4.1.4 defun parseTranList

```
[parseTran p93]
[parseTranList p95]
```

— defun parseTranList —

```
(defun |parseTranList| (x)
  (if (atom x)
      (|parseTran| x)
      (cons (|parseTran| (car x)) (|parseTranList| (cdr x)))))
```

4.1.5 defplist parseConstruct

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| '|parseTran|) '|parseConstruct|))
```

4.1.6 defun parseConstruct

```
[parseTranList p95]
[$insideConstructIfTrue p??]
```

— defun parseConstruct —

```
(defun |parseConstruct| (u)
  (let (|$insideConstructIfTrue| x)
    (declare (special |$insideConstructIfTrue|))
    (setq |$insideConstructIfTrue| t)
    (setq x (|parseTranList| u))
    (cons '|construct| x)))
```

4.2 Indirect called parse routines

In the `parseTran` function there is the code:

```
((and (atom u) (setq fn (get1 u '|parseTran|)))
  (funcall fn arg1))
```

The functions in this section are called through the symbol-plist of the symbol being parsed. The original list read:

<code>and</code>	<code>parseAnd</code>
<code>@</code>	<code>parseAtSign</code>
<code>CATEGORY</code>	<code>parseCategory</code>
<code>::</code>	<code>parseCoerce</code>
<code>\:</code>	<code>parseColon</code>
<code>construct</code>	<code>parseConstruct</code>
<code>DEF</code>	<code>parseDEF</code>
<code>\$<=</code>	<code>parseDollarLessEqual</code>
<code>\$></code>	<code>parseDollarGreaterThan</code>
<code>\$>=</code>	<code>parseDollarGreaterEqual</code>
<code>\$^=</code>	<code>parseDollarNotEqual</code>
<code>eqv</code>	<code>parseEquivalence</code>
<code>exit</code>	<code>parseExit</code>
<code>></code>	<code>parseGreaterThan</code>
<code>>=</code>	<code>parseGreaterEqual</code>
<code>has</code>	<code>parseHas</code>
<code>IF</code>	<code>parseIf</code>
<code>implies</code>	<code>parseImplies</code>
<code>IN</code>	<code>parseIn</code>
<code>INBY</code>	<code>parseInBy</code>
<code>is</code>	<code>parseIs</code>
<code>isnt</code>	<code>parseIsnt</code>
<code>Join</code>	<code>parseJoin</code>
<code>leave</code>	<code>parseLeave</code>
<code>;;control-H</code>	<code>parseLeftArrow</code>
<code><=</code>	<code>parseLessEqual</code>
<code>LET</code>	<code>parseLET</code>
<code>LETD</code>	<code>parseLETD</code>
<code>MDEF</code>	<code>parseMDEF</code>
<code>^</code>	<code>parseNot</code>
<code>not</code>	<code>parseNot</code>
<code>^=</code>	<code>parseNotEqual</code>
<code>or</code>	<code>parseOr</code>
<code>pretend</code>	<code>parsePretend</code>
<code>return</code>	<code>parseReturn</code>
<code>SEGMENT</code>	<code>parseSegment</code>

```

SEQ          parseSeq
;;control-V  parseUpArrow
VCONS       parseVCONS
where        parseWhere

```

4.2.1 defplist parseAnd

— postvars —

```

(eval-when (eval load)
  (setf (get '|and| '|parseTran|) '|parseAnd|))

```

4.2.2 defun parseAnd

```

[parseTran p93]
[parseAnd p97]
[parseTranList p95]
[parseIf p114]
[$InteractiveMode p??]

```

— defun parseAnd —

```

(defun |parseAnd| (arg)
  (cond
    (|$InteractiveMode| (cons '|and| (|parseTranList| arg)))
    ((null arg) '|true|)
    ((null (cdr arg)) (car arg))
    (t
     (|parseIf|
      (list (|parseTran| (car arg)) (|parseAnd| (cdr arg)) '|false| )))))

```

4.2.3 defplist parseAtSign

— postvars —

```

(eval-when (eval load)
  (setf (get '@ '|parseTran|) '|parseAtSign|))

```

4.2.4 defun parseAtSign

```
[parseTran p93]
[parseType p98]
[$InteractiveMode p??]
```

— defun parseAtSign —

```
(defun |parseAtSign| (arg)
  (declare (special |$InteractiveMode|))
  (if |$InteractiveMode|
    (list '@ (|parseTran| (first arg)) (|parseTran| (|parseType| (second arg))))
    (list '@ (|parseTran| (first arg)) (|parseTran| (second arg)))))
```

4.2.5 defun parseType

```
[msubst p??]
[parseTran p93]
```

— defun parseType —

```
(defun |parseType| (x)
  (declare (special |$EmptyMode| |$quadSymbol|))
  (setq x (msubst |$EmptyMode| |$quadSymbol| x))
  (if (and (consp x) (eq (qfirst x) '|typeOf|)
        (consp (qrest x)) (eq (qcddr x) nil))
    (list '|typeOf| (|parseTran| (qsecond x)))
    x))
```

4.2.6 defplist parseCategory

— postvars —

```
(eval-when (eval load)
  (setf (get 'category '|parseTran|) '|parseCategory|))
```

4.2.7 defun parseCategory

```
[parseTranList p95]
[parseDropAssertions p99]
[contained p??]
```

— defun parseCategory —

```
(defun |parseCategory| (arg)
  (let (z key)
    (setq z (|parseTranList| (|parseDropAssertions| arg)))
    (setq key (if (contained '$ z) '|domain| '|package|))
    (cons 'category (cons key z))))
```

—————

4.2.8 defun parseDropAssertions

```
[parseDropAssertions p99]
```

— defun parseDropAssertions —

```
(defun |parseDropAssertions| (x)
  (cond
    ((not (consp x)) x)
    ((and (consp (qfirst x)) (eq (qcaar x) 'if)
          (consp (qcdar x))
          (eq (qcadar x) '|asserted|))
     (|parseDropAssertions| (qrest x)))
    (t (cons (qfirst x) (|parseDropAssertions| (qrest x))))))
```

—————

4.2.9 defpllist parseCoerce

— postvars —

```
(eval-when (eval load)
  (setf (get '|::| '|parseTran|) '|parseCoerce|))
```

—————

4.2.10 defun parseCoerce

```
[parseType p98]
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseCoerce —

```
(defun |parseCoerce| (arg)
  (if |$InteractiveMode|
    (list '|::|
      (|parseTran| (first arg)) (|parseTran| (|parseType| (second arg))))
    (list '|::| (|parseTran| (first arg)) (|parseTran| (second arg)))))
```

—————

4.2.11 defplist parseColon

— postvars —

```
(eval-when (eval load)
  (setf (get '|:| '|parseTran|) '|parseColon|))
```

—————

4.2.12 defun parseColon

```
[parseTran p93]
[parseType p98]
[$InteractiveMode p??]
[$insideConstructIfTrue p??]
```

— defun parseColon —

```
(defun |parseColon| (arg)
  (declare (special |$insideConstructIfTrue|))
  (cond
    ((and (consp arg) (eq (qrest arg) nil))
     (list '|:| (|parseTran| (first arg))))
    ((and (consp arg) (consp (qrest arg)) (eq (qcddr arg) nil))
     (if |$InteractiveMode|
       (if |$insideConstructIfTrue|
         (list 'tag (|parseTran| (first arg))
```

```

(|parseTran| (second arg)))
(list '|:| (|parseTran| (first arg))
      (|parseTran| (|parseType| (second arg)))))
(list '|:| (|parseTran| (first arg))
      (|parseTran| (second arg)))))

```

4.2.13 defplist parseDEF

— postvars —

```

(eval-when (eval load)
  (setf (get 'def '|parseTran|) '|parseDEF|))

```

4.2.14 defun parseDEF

```

[setDefOp p394]
[parseLhs p102]
[parseTranList p95]
[parseTranCheckForRecord p517]
[opFf p??]
[$lhs p??]

```

— defun parseDEF —

```

(defun |parseDEF| (arg)
  (let (|$lhs| tList specialList body)
    (declare (special |$lhs|))
    (setq |$lhs| (first arg))
    (setq tList (second arg))
    (setq specialList (third arg))
    (setq body (fourth arg))
    (|setDefOp| |$lhs|)
    (list 'def (|parseLhs| |$lhs|)
          (|parseTranList| tList)
          (|parseTranList| specialList)
          (|parseTranCheckForRecord| body (|opOf| |$lhs|)))))

```

4.2.15 defun parseLhs

[parseTran p93]
[transIs p102]

— **defun parseLhs** —

```
(defun |parseLhs| (x)
  (let (result)
    (cond
      ((atom x) (|parseTran| x))
      ((atom (car x))
       (cons (|parseTran| (car x))
              (dolist (y (cdr x) (nreverse result))
                (push (|transIs| (|parseTran| y)) result))))
      (t (|parseTran| x)))))
```

4.2.16 defun transIs

[isListConstructor p103]
[transIs1 p102]

— **defun transIs** —

```
(defun |transIs| (u)
  (if (|isListConstructor| u)
      (cons '|construct| (|transIs1| u))
      u))
```

4.2.17 defun transIs1

[qcar p??]
[qcdr p??]
[nreverse0 p??]
[transIs p102]
[transIs1 p102]

— **defun transIs1** —

```
(defun |transIs1| (u)
```

```

(let (x h v tmp3)
  (cond
    ((and (consp u) (eq (qfirst u) '|construct|))
      (dolist (x (qrest u) (nreverse0 tmp3))
        (push (|transIs| x) tmp3)))
    ((and (consp u) (eq (qfirst u) '|append|) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (setq x (qsecond u))
      (setq h (list '|:| (|transIs| x)))
      (setq v (|transIs1| (qthird u)))
      (cond
        ((and (consp v) (eq (qfirst v) '|:|)
          (consp (qrest v)) (eq (qcddr v) nil))
          (list h (qsecond v)))
        ((eq v '|nil|) (car (cdr h)))
        ((atom v) (list h (list '|:| v)))
        (t (cons h v))))
    ((and (consp u) (eq (qfirst u) '|cons|) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (setq h (|transIs| (qsecond u)))
      (setq v (|transIs1| (qthird u)))
      (cond
        ((and (consp v) (eq (qfirst v) '|:|) (consp (qrest v))
          (eq (qcddr v) nil))
          (cons h (list (qsecond v))))
        ((eq v '|nil|) (cons h nil))
        ((atom v) (list h (list '|:| v)))
        (t (cons h v))))
    (t u))))

```

4.2.18 defun isListConstructor

[member p??]

— defun isListConstructor —

```

(defun |isListConstructor| (u)
  (and (consp u) (|member| (qfirst u) '(|construct| |append| |cons|))))

```

4.2.19 defplist parseDollarGreaterthan

— postvars —

```
(eval-when (eval load)
  (setf (get '|$>| ' |parseTran|) '|parseDollarGreaterthan|))
```

4.2.20 defun parseDollarGreaterThan

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarGreaterThan —

```
(defun |parseDollarGreaterthan| (arg)
  (declare (special |$op|))
  (list (msubst '$< '$> |$op|)
        (|parseTran| (second arg))
        (|parseTran| (first arg))))
```

4.2.21 defplist parseDollarGreaterEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|$>=| ' |parseTran|) '|parseDollarGreaterEqual|))
```

4.2.22 defun parseDollarGreaterEqual

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarGreaterEqual —

```
(defun |parseDollarGreaterEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$< '$>= |$op|) arg))))
```

— postvars —

```
(eval-when (eval load)
  (setf (get '$<=| '|parseTran|) '|parseDollarLessEqual|))
```

4.2.23 defun parseDollarLessEqual

```
[msubst p??]
[parseTran p93]
[$op p??]
```

— defun parseDollarLessEqual —

```
(defun |parseDollarLessEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$> '$<= |$op|) arg))))
```

4.2.24 deflist parseDollarNotEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '$^=| '|parseTran|) '|parseDollarNotEqual|))
```

4.2.25 defun parseDollarNotEqual

```
[parseTran p93]
[msubst p??]
```

`[$op p??]`

— `defun parseDollarNotEqual` —

```
(defun |parseDollarNotEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '$= '$~= |$op|) arg))))
```

—————

4.2.26 `defplist parseEquivalence`

— `postvars` —

```
(eval-when (eval load)
  (setf (get '|eqv| '|parseTran|) '|parseEquivalence|))
```

—————

4.2.27 `defun parseEquivalence`

`[parseIf p114]`

— `defun parseEquivalence` —

```
(defun |parseEquivalence| (arg)
  (|parseIf|
   (list (first arg) (second arg)
         (|parseIf| (cons (second arg) '(|false| |true|))))))
```

—————

4.2.28 `defplist parseExit`

— `postvars` —

```
(eval-when (eval load)
  (setf (get '|exit| '|parseTran|) '|parseExit|))
```

—————

4.2.29 defun parseExit

```
[parseTran p93]
[moan p??]
```

— defun parseExit —

```
(defun |parseExit| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (if b
      (cond
        ((null (integerp a))
         (moan "first arg " a " for exit must be integer")
         (list '|exit| 1 a ))
        (t
         (cons '|exit| (cons a b))))
      (list '|exit| 1 a ))))
```

4.2.30 defplist parseGreaterEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|>=| '|parseTran|) '|parseGreaterEqual|))
```

4.2.31 defun parseGreaterEqual

```
[parseTran p93]
[$op p??]
```

— defun parseGreaterEqual —

```
(defun |parseGreaterEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '<' '>= '|$op|) arg))))
```

4.2.32 defplist parseGreaterThan

— postvars —

```
(eval-when (eval load)
  (setf (get '|>' '|parseTran|) '|parseGreaterThan|))
```

—————

4.2.33 defun parseGreaterThan

```
[parseTran p93]
[$op p??]
```

— defun parseGreaterThan —

```
(defun |parseGreaterThan| (arg)
  (declare (special |$op|))
  (list (msubst '<' '>' |$op|)
        (|parseTran| (second arg)) (|parseTran| (first arg))))
```

—————

4.2.34 defplist parseHas

— postvars —

```
(eval-when (eval load)
  (setf (get '|has| '|parseTran|) '|parseHas|))
```

—————

4.2.35 defun parseHas

```
[unabbrevAndLoad p??]
[qcar p??]
[qcdr p??]
[getdatabase p??]
[opOf p??]
[makeNonAtomic p??]
```

```
[parseHasRhs p110]
[member p??]
[parseType p98]
[nreverse0 p??]
[$InteractiveMode p??]
[$CategoryFrame p??]
```

— defun parseHas —

```
(defun |parseHas| (arg)
  (labels (
    (fn (arg)
      (let (tmp4 tmp6 map op kk)
        (declare (special |$InteractiveMode|))
        (when |$InteractiveMode| (setq arg (|unabbrevAndLoad| arg)))
        (cond
          ((and (consp arg) (eq (qfirst arg) '|:|) (consp (qrest arg))
            (consp (qcddr arg)) (eq (qcddr arg) nil)
            (consp (qthird arg))
            (eq (qcaaddr arg) '|Mapping|))
           (setq map (rest (third arg)))
           (setq op (second arg))
           (setq op (if (stringp op) (intern op) op))
           (list (list 'signature op map)))
          ((and (consp arg) (eq (qfirst arg) '|Join|))
           (dolist (z (rest arg) tmp4)
             (setq tmp4 (append tmp4 (fn z))))))
          ((and (consp arg) (eq (qfirst arg) 'category))
           (dolist (z (rest arg) tmp6)
             (setq tmp6 (append tmp6 (fn z))))))
          (t
           (setq kk (getdatabase (|opOf| arg) 'constructorkind))
           (cond
            ((or (eq kk '|domain|) (eq kk '|category|))
             (list (|makeNonAtomic| arg)))
            ((and (consp arg) (eq (qfirst arg) 'attribute))
             (list arg))
            ((and (consp arg) (eq (qfirst arg) 'signature))
             (list arg))
            (|$InteractiveMode|
             (|parseHasRhs| arg))
            (t
             (list (list 'attribute arg)))))))
    (let (tmp1 tmp2 tmp3 x)
      (declare (special |$InteractiveMode| |$CategoryFrame|))
      (setq x (first arg))
      (setq tmp1 (|get| x '|value| |$CategoryFrame|))
      (when |$InteractiveMode|
        (setq x
```

```

      (if (and (consp tmp1) (consp (qrest tmp1)) (consp (qcddr tmp1))
              (eq (qcddr tmp1) nil)
              (|member| (second tmp1)
                        '((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
          (first tmp1)
          (|parseType| x))))
    (setq tmp2
      (dolist (u (fn (second arg)) (nreverse0 tmp3))
        (push (list '|has| x u ) tmp3)))
    (if (and (consp tmp2) (eq (qrest tmp2) nil))
        (qfirst tmp2)
        (cons '|and| tmp2))))

```

4.2.36 defun parseHasRhs

```

[get p??]
[qcar p??]
[qcdr p??]
[member p??]
[abbreviation? p??]
[loadIfNecessary p11]
[unabbrevAndLoad p??]
[$CategoryFrame p??]

```

— defun parseHasRhs —

```

(defun |parseHasRhs| (u)
  (let (tmp1 y)
    (declare (special |$CategoryFrame|))
    (setq tmp1 (|get| u '|value| |$CategoryFrame|))
    (cond
      ((and (consp tmp1) (consp (qrest tmp1))
              (consp (qcddr tmp1)) (eq (qcddr tmp1) nil)
              (|member| (second tmp1)
                        '((|Mode|) (|Domain|) (|SubDomain| (|Domain|)))))
          (second tmp1))
      ((setq y (|abbreviation?| u))
          (if (|loadIfNecessary| y)
              (list (|unabbrevAndLoad| y))
              (list (list 'attribute u))))
      (t (list (list 'attribute u)))))

```

4.2.37 defun loadIfNecessary

```
[loadLibIfNecessary p111]
```

— **defun loadIfNecessary** —

```
(defun |loadIfNecessary| (u)
  (|loadLibIfNecessary| u t))
```

4.2.38 defun loadLibIfNecessary

```
[loadLibIfNecessary p111]
[functionp p??]
[macrop p??]
[getl p??]
[loadLib p??]
[lassoc p??]
[getProplist p??]
[getdatabase p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[throwKeyedMsg p??]
[$CategoryFrame p??]
[$InteractiveMode p??]
```

— **defun loadLibIfNecessary** —

```
(defun |loadLibIfNecessary| (u mustExist)
  (let (value y)
    (declare (special |$CategoryFrame| |$InteractiveMode|))
    (cond
      ((eq u '|$EmptyMode|) u)
      ((null (atom u)) (|loadLibIfNecessary| (car u) mustExist))
      (t
       (setq value
        (cond
          ((or (|functionp| u) (|macrop| u)) u)
          ((getl u 'loaded) u)
          ((|loadLib| u) u)))
        (cond
          ((and (null |$InteractiveMode|)
            (or (null (setq y (|getProplist| u |$CategoryFrame|)))
              (and (null (lassoc '|isFunctor| y))
                (null (lassoc '|isCategory| y))))))
```

```

      (if (setq y (getdatabase u 'constructorkind))
        (if (eq y '|category|)
          (|updateCategoryFrameForCategory| u)
          (|updateCategoryFrameForConstructor| u))
        (|throwKeyedMsg| 's2il0005 (list u))))
      (t value))))))

```

4.2.39 defun updateCategoryFrameForConstructor

```

[getdatabase p??]
[put p??]
[convertOpAlist2compilerInfo p112]
[addModemap p253]
[$CategoryFrame p??]
[$CategoryFrame p??]

```

— defun updateCategoryFrameForConstructor —

```

(defun |updateCategoryFrameForConstructor| (constructor)
  (let (opAlist tmp1 dc sig pred impl)
    (declare (special |$CategoryFrame|))
    (setq opalist (getdatabase constructor 'operationalist))
    (setq tmp1 (getdatabase constructor 'constructormodemap))
    (setq dc (caar tmp1))
    (setq sig (cdar tmp1))
    (setq pred (caadr tmp1))
    (setq impl (cadadr tmp1))
    (setq |$CategoryFrame|
      (|put| constructor '|isFunction|
        (|convertOpAlist2compilerInfo| opalist)
        (|addModemap| constructor dc sig pred impl
          (|put| constructor '|mode| (cons '|Mapping| sig) |$CategoryFrame|))))))

```

4.2.40 defun convertOpAlist2compilerInfo

— defun convertOpAlist2compilerInfo —

```

(defun |convertOpAlist2compilerInfo| (opalist)
  (labels (
    (formatSig (op arg2)

```

```

(let (typelist slot stuff pred impl)
  (setq typelist (car arg2))
  (setq slot (cadr arg2))
  (setq stuff (caddr arg2))
  (setq pred (if stuff (car stuff) t))
  (setq impl (if (cdr stuff) (cadr stuff) 'elt))
  (list (list op typelist) pred (list impl '$ slot))))
(let (data result)
  (setq data
    (loop for item in opalist
      collect
        (loop for sig in (rest item)
          collect (formatSig (car item) sig))))
  (dolist (term data result)
    (setq result (append result term)))))

```

4.2.41 defun updateCategoryFrameForCategory

```

[getdatabase p??]
[put p??]
[addModemap p253]
[$CategoryFrame p??]
[$CategoryFrame p??]

```

— defun updateCategoryFrameForCategory —

```

(defun |updateCategoryFrameForCategory| (category)
  (let (tmp1 dc sig pred impl)
    (declare (special |$CategoryFrame|))
    (setq tmp1 (getdatabase category 'constructormodemap))
    (setq dc (caar tmp1))
    (setq sig (cdar tmp1))
    (setq pred (caadr tmp1))
    (setq impl (cadadr tmp1))
    (setq |$CategoryFrame|
      (|put| category '|isCategory| t
        (|addModemap| category dc sig pred impl |$CategoryFrame|))))

```

4.2.42 defplist parseIf

— postvars —

```
(eval-when (eval load)
  (setf (get 'if '|parseTran|) '|parseIf|))
```

4.2.43 defun parseIf

```
[parseIf,ifTran p114]
[parseTran p93]
```

— defun parseIf —

```
(defun |parseIf| (arg)
  (if (null (and (consp arg) (consp (qrest arg))
                (consp (qcddr arg)) (eq (qcddr arg) nil)))
      arg
      (|parseIf,ifTran|
        (|parseTran| (first arg))
        (|parseTran| (second arg))
        (|parseTran| (third arg)))))
```

4.2.44 defun parseIf,ifTran

```
[parseIf,ifTran p114]
[incExitLevel p??]
[makeSimplePredicateOrNil p518]
[incExitLevel p??]
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseIf,ifTran —

```
(defun |parseIf,ifTran| (pred a b)
  (let (pp z ap bp tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 val s)
    (declare (special |$InteractiveMode|))
    (cond
      ((and (null |$InteractiveMode|) (eq pred '|true|))
       a)
      ((and (null |$InteractiveMode|) (eq pred '|false|))
       b)
      ((and (consp pred) (eq (qfirst pred) '|not|)
            (consp (qrest pred)) (eq (qcddr pred) nil))
       (|parseIf,ifTran| (second pred) b a))
```

```

((and (consp pred) (eq (qfirst pred) 'if)
  (progn
    (setq tmp1 (qrest pred))
    (and (consp tmp1)
      (progn
        (setq pp (qfirst tmp1))
        (setq tmp2 (qrest tmp1))
        (and (consp tmp2)
          (progn
            (setq ap (qfirst tmp2))
            (setq tmp3 (qrest tmp2))
            (and (consp tmp3)
              (eq (qrest tmp3) nil)
              (progn (setq bp (qfirst tmp3)) t)))))))
  (|parseIf,ifTran| pp
    (|parseIf,ifTran| ap (copy a) (copy b))
    (|parseIf,ifTran| bp a b)))
((and (consp pred) (eq (qfirst pred) 'seq)
  (consp (qrest pred)) (progn (setq tmp2 (reverse (qrest pred))) t)
  (and (consp tmp2)
    (consp (qfirst tmp2))
    (eq (qcaar tmp2) '|exit|)
    (progn
      (setq tmp4 (qcddr tmp2))
      (and (consp tmp4)
        (equal (qfirst tmp4) 1)
        (progn
          (setq tmp5 (qrest tmp4))
          (and (consp tmp5)
            (eq (qrest tmp5) nil)
            (progn (setq pp (qfirst tmp5)) t))))
      (progn (setq z (qrest tmp2)) t))
    (progn (setq z (nreverse z)) t))
  (cons 'seq
    (append z
      (list
        (list '|exit| 1 (|parseIf,ifTran| pp
          (|incExitLevel| a)
          (|incExitLevel| b)))))))
((and (consp a) (eq (qfirst a) 'if) (consp (qrest a))
  (equal (qsecond a) pred) (consp (qcddr a))
  (consp (qcdddr a))
  (eq (qcdddr a) nil))
  (list 'if pred (third a) b))
((and (consp b) (eq (qfirst b) 'if)
  (consp (qrest b)) (equal (qsecond b) pred)
  (consp (qcddr b))
  (consp (qcdddr b))
  (eq (qcdddr b) nil))
  (list 'if pred a (fourth b)))

```

```

(progn
  (setq tmp1 (|makeSimplePredicateOrNil| pred))
  (and (consp tmp1) (eq (qfirst tmp1) 'seq)
    (progn
      (setq tmp2 (qrest tmp1))
      (and (and (consp tmp2)
        (progn (setq tmp3 (reverse tmp2)) t))
        (and (consp tmp3)
          (progn
            (setq tmp4 (qfirst tmp3))
            (and (consp tmp4) (eq (qfirst tmp4) '|exit|)
              (progn
                (setq tmp5 (qrest tmp4))
                (and (consp tmp5) (equal (qfirst tmp5) 1)
                  (progn
                    (setq tmp6 (qrest tmp5))
                    (and (consp tmp6) (eq (qrest tmp6) nil)
                      (progn (setq val (qfirst tmp6)) t)))))))
                (progn (setq s (qrest tmp3)) t))))))
      (setq s (nreverse s))
      (|parseTran|
        (cons 'seq
          (append s
            (list (list '|exit| 1 (|incExitLevel| (list 'if val a b))))))))
    (t
      (list 'if pred a b )))))

```

4.2.45 defplist parseImplies

— postvars —

```

(eval-when (eval load)
  (setf (get '|implies| '|parseTran|) '|parseImplies|))

```

4.2.46 defun parseImplies

[parseIf p114]

— defun parseImplies —

```
(defun |parseImplies| (arg)
  (|parseIf| (list (first arg) (second arg) '|true|)))
```

4.2.47 defplist parseIn

— postvars —

```
(eval-when (eval load)
  (setf (get 'in '|parseTran|) '|parseIn|))
```

4.2.48 defun parseIn

```
[parseTran p93]
[postError p364]
```

— defun parseIn —

```
(defun |parseIn| (arg)
  (let (i n)
    (setq i (|parseTran| (first arg)))
    (setq n (|parseTran| (second arg)))
    (cond
      ((and (consp n) (eq (qfirst n) 'segment)
            (consp (qrest n)) (eq (qcddr n) nil))
       (list 'step i (second n) 1))
      ((and (consp n) (eq (qfirst n) '|reverse|)
            (consp (qrest n)) (eq (qcddr n) nil)
            (consp (qsecond n)) (eq (qcaadr n) 'segment)
            (consp (qcdadr n))
            (eq (qcddadr n) nil))
       (|postError| (list " You cannot reverse an infinite sequence." )))
      ((and (consp n) (eq (qfirst n) 'segment)
            (consp (qrest n)) (consp (qcddr n))
            (eq (qcdddr n) nil))
       (if (third n)
           (list 'step i (second n) 1 (third n))
           (list 'step i (second n) 1)))
      ((and (consp n) (eq (qfirst n) '|reverse|)
            (consp (qrest n)) (eq (qcddr n) nil)
            (consp (qsecond n)) (eq (qcaadr n) 'segment)
```

```

      (consp (qcdadr n))
      (consp (qcddadr n))
      (eq (qrest (qcddadr n)) nil))
    (if (third (second n))
        (list 'step i (third (second n)) -1 (second (second n)))
        (|postError| (list " You cannot reverse an infinite sequence."))))
    ((and (consp n) (eq (qfirst n) '|tails|)
          (consp (qrest n)) (eq (qcddr n) nil))
      (list 'on i (second n)))
    (t
      (list 'in i n))))

```

4.2.49 defplist parseInBy

— postvars —

```

(eval-when (eval load)
  (setf (get 'inby '|parseTran|) '|parseInBy|))

```

4.2.50 defun parseInBy

```

[postError p364]
[parseTran p93]
[bright p??]
[parseIn p117]

```

— defun parseInBy —

```

(defun |parseInBy| (arg)
  (let (i n inc u)
    (setq i (first arg))
    (setq n (second arg))
    (setq inc (third arg))
    (setq u (|parseIn| (list i n)))
    (cond
      ((null (and (consp u) (eq (qfirst u) 'step)
                  (consp (qrest u))
                  (consp (qcddr u))
                  (consp (qcdddr u))))
        (|postError|

```

```

      (cons '| You cannot use|
        (append (|bright| "by")
          (list "except for an explicitly indexed sequence."))))))
(t
  (setq inc (|parseTran| inc))
  (cons 'step
    (cons (second u)
      (cons (third u)
        (cons (|parseTran| inc) (cddddr u))))))))))

```

4.2.51 defplist parseIs

— postvars —

```

(eval-when (eval load)
  (setf (get '|is| '|parseTran|) '|parseIs|))

```

4.2.52 defun parseIs

```

[parseTran p93]
[transIs p102]

```

— defun parseIs —

```

(defun |parseIs| (arg)
  (list '|is| (|parseTran| (first arg)) (|transIs| (|parseTran| (second arg)))))

```

4.2.53 defplist parseIsnt

— postvars —

```

(eval-when (eval load)
  (setf (get '|isnt| '|parseTran|) '|parseIsnt|))

```

4.2.54 defun parseIsnt

[parseTran p93]
 [transIs p102]

— defun parseIsnt —

```
(defun |parseIsnt| (arg)
  (list '|isnt|
        (|parseTran| (first arg))
        (|transIs| (|parseTran| (second arg)))))
```

—————

4.2.55 defplist parseJoin

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| '|parseTran|) '|parseJoin|))
```

—————

4.2.56 defun parseJoin

[parseTranList p95]

— defun parseJoin —

```
(defun |parseJoin| (thejoin)
  (labels (
    (fn (arg)
      (cond
        ((null arg)
         nil)
        ((and (consp arg) (consp (qfirst arg)) (eq (qcaar arg) '|Join|))
         (append (cdar arg) (fn (rest arg))))
        (t
         (cons (first arg) (fn (rest arg)))))))
    )
  (cons '|Join| (fn (|parseTranList| thejoin)))))
```

—————

4.2.57 defplist parseLeave

— postvars —

```
(eval-when (eval load)
  (setf (get '|leave| '|parseTran|) '|parseLeave|))
```

4.2.58 defun parseLeave

[parseTran p93]

— defun parseLeave —

```
(defun |parseLeave| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (cond
      (b
       (cond
         ((null (integerp a))
          (moan "first arg " a " for 'leave' must be integer")
          (list '|leave| 1 a))
         (t (cons '|leave| (cons a b))))))
    (t (list '|leave| 1 a)))))
```

4.2.59 defplist parseLessEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|<=| '|parseTran|) '|parseLessEqual|))
```

4.2.60 defun parseLessEqual

```
[parseTran p93]
[$op p??]
```

— defun parseLessEqual —

```
(defun |parseLessEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '>' '<= |$op|) arg)))))
```

—————

4.2.61 defplist parseLET

— postvars —

```
(eval-when (eval load)
  (setf (get 'let '|parseTran|) '|parseLET|))
```

—————

4.2.62 defun parseLET

```
[parseTran p93]
[parseTranCheckForRecord p517]
[opOf p??]
[transIs p102]
```

— defun parseLET —

```
(defun |parseLET| (arg)
  (let (p)
    (setq p
      (list 'let (|parseTran| (first arg))
        (|parseTranCheckForRecord| (second arg) (|opOf| (first arg)))))
    (if (eq (|opOf| (first arg)) '|cons|)
      (list 'let (|transIs| (second p)) (third p))
      p)))
```

—————

4.2.63 defplist parseLETD

— postvars —

```
(eval-when (eval load)
  (setf (get 'letd '|parseTran|) '|parseLETD|))
```

4.2.64 defun parseLETD

```
[parseTran p93]
[parseType p98]
```

— defun parseLETD —

```
(defun |parseLETD| (arg)
  (list 'letd
    (|parseTran| (first arg))
    (|parseTran| (|parseType| (second arg)))))
```

4.2.65 defplist parseMDEF

— postvars —

```
(eval-when (eval load)
  (setf (get 'mdef '|parseTran|) '|parseMDEF|))
```

4.2.66 defun parseMDEF

```
[parseTran p93]
[parseTranList p95]
[parseTranCheckForRecord p517]
[opOf p??]
[$lhs p??]
```

— defun parseMDEF —

```
(defun |parseMDEF| (arg)
  (let (|$lhs|)
    (declare (special |$lhs|))
    (setq |$lhs| (first arg))
    (list 'mdef
      (|parseTran| |$lhs|)
      (|parseTranList| (second arg))
      (|parseTranList| (third arg))
      (|parseTranCheckForRecord| (fourth arg) (|opOf| |$lhs|))))
```

4.2.67 defplist parseNot

— postvars —

```
(eval-when (eval load)
  (setf (get '|not| '|parseTran|) '|parseNot|))
```

4.2.68 defplist parseNot

— postvars —

```
(eval-when (eval load)
  (setf (get '|^| '|parseTran|) '|parseNot|))
```

4.2.69 defun parseNot

```
[parseTran p93]
[$InteractiveMode p??]
```

— defun parseNot —

```
(defun |parseNot| (arg)
  (declare (special |$InteractiveMode|))
  (if |$InteractiveMode|
    (list '|not| (|parseTran| (car arg)))
```

```
(|parseTran| (cons 'if (cons (car arg) '(|false| |true|))))))
```

4.2.70 defplist parseNotEqual

— postvars —

```
(eval-when (eval load)
  (setf (get '|^=' '|parseTran|) '|parseNotEqual|))
```

4.2.71 defun parseNotEqual

```
[parseTran p93]
[msubst p??]
[$op p??]
```

— defun parseNotEqual —

```
(defun |parseNotEqual| (arg)
  (declare (special |$op|))
  (|parseTran| (list '|not| (cons (msubst '= '^= |$op|) arg))))
```

4.2.72 defplist parseOr

— postvars —

```
(eval-when (eval load)
  (setf (get '|or| '|parseTran|) '|parseOr|))
```

4.2.73 defun parseOr

```
[parseTran p93]
[parseTranList p95]
```

[parseIf p114]
 [parseOr p125]

— defun parseOr —

```
(defun |parseOr| (arg)
  (let (x)
    (setq x (|parseTran| (car arg)))
    (cond
      ($InteractiveMode| (cons '|or| (|parseTranList| arg)))
      ((null arg) '|false|)
      ((null (cdr arg)) (car arg))
      ((and (consp x) (eq (qfirst x) '|not|)
            (consp (qrest x)) (eq (qcddr x) nil))
       (|parseIf| (list (second x) (|parseOr| (cdr arg)) '|true|)))
      (t
       (|parseIf| (list x '|true| (|parseOr| (cdr arg))))))))
```

—

4.2.74 defplist parsePretend

— postvars —

```
(eval-when (eval load)
  (setf (get '|pretend| '|parseTran|) '|parsePretend|))
```

—

4.2.75 defun parsePretend

[parseTran p93]
 [parseType p98]

— defun parsePretend —

```
(defun |parsePretend| (arg)
  (if |$InteractiveMode|
    (list '|pretend|
          (|parseTran| (first arg))
          (|parseTran| (|parseType| (second arg)))))
    (list '|pretend|
          (|parseTran| (first arg))
          (|parseTran| (second arg)))))
```

4.2.76 defplist parseReturn

— postvars —

```
(eval-when (eval load)
  (setf (get '|return| '|parseTran|) '|parseReturn|))
```

4.2.77 defun parseReturn

```
[parseTran p93]
[moan p??]
```

— defun parseReturn —

```
(defun |parseReturn| (arg)
  (let (a b)
    (setq a (|parseTran| (car arg)))
    (setq b (|parseTran| (cdr arg)))
    (cond
      (b
       (when (nequal a 1) (moan "multiple-level 'return' not allowed"))
       (cons '|return| (cons 1 b)))
      (t (list '|return| 1 a)))))
```

4.2.78 defplist parseSegment

— postvars —

```
(eval-when (eval load)
  (setf (get '|segment| '|parseTran|) '|parseSegment|))
```

4.2.79 defun parseSegment

[parseTran p93]

— defun parseSegment —

```
(defun |parseSegment| (arg)
  (if (and (consp arg) (consp (qrest arg)) (eq (qcddr arg) nil))
      (if (second arg)
          (list 'segment (|parseTran| (first arg)) (|parseTran| (second arg)))
          (list 'segment (|parseTran| (first arg))))
      (cons 'segment arg)))
```

4.2.80 defplist parseSeq

— postvars —

```
(eval-when (eval load)
  (setf (get 'seq '|parseTran|) '|parseSeq|))
```

4.2.81 defun parseSeq

```
[postError p364]
[transSeq p??]
[mapInto p??]
[last p??]
```

— defun parseSeq —

```
(defun |parseSeq| (arg)
  (let (tmp1)
    (when (consp arg) (setq tmp1 (reverse arg)))
    (if (null (and (consp arg) (consp tmp1)
                  (consp (qfirst tmp1)) (eq (qcaar tmp1) '|exit|))))
        (|postError| (list " Invalid ending to block: " (|last| arg)))
        (|transSeq| (|mapInto| arg '|parseTran|)))))
```

4.2.82 defplist parseVCONS

— postvars —

```
(eval-when (eval load)
  (setf (get 'vcons '|parseTran|) '|parseVCONS|))
```

4.2.83 defun parseVCONS

[parseTranList p95]

— defun parseVCONS —

```
(defun |parseVCONS| (arg)
  (cons 'vector (|parseTranList| arg)))
```

4.2.84 defplist parseWhere

— postvars —

```
(eval-when (eval load)
  (setf (get '|where| '|parseTran|) '|parseWhere|))
```

4.2.85 defun parseWhere

[mapInto p??]

— defun parseWhere —

```
(defun |parseWhere| (arg)
  (cons '|where| (|mapInto| arg '|parseTran|)))
```

Chapter 5

Compile Transformers

5.0.86 `defvar $NoValueMode`

— **initvars** —

```
(defvar |$NoValueMode| ' |NoValueMode|)
```

—————

5.0.87 `defvar $EmptyMode`

`$EmptyMode` is a constant whose value is `$EmptyMode`. It is used by `isPartialMode` to decide if a modemap is partially constructed. If the `$EmptyMode` constant occurs anywhere in the modemap structure at any depth then the modemap is still incomplete. To find this constant the `isPartialMode` function calls `CONTAINED $EmptyMode Y` which will walk the structure `Y` looking for this constant.

— **initvars** —

```
(defvar |$EmptyMode| ' |EmptyMode|)
```

—————

5.1 Routines for handling forms

The functions in this section are called through the symbol-plist of the symbol being parsed.

- `add (p255) compAdd(form mode env) → (form mode env)`

- @ (p352) compAtSign(form mode env) →
- CAPSULE (p258) compCapsule(form mode env) →
- case (p268) compCase(form mode env) →
- Mapping (p270) compCat(form mode env) →
- Record (p270) compCat(form mode env) →
- Union (p270) compCat(form mode env) →
- CATEGORY (p270) compCategory(form mode env) →
- :: (p352) compCoerce(form mode env) →
- : (p275) compColon(form mode env) →
- CONS (p278) compCons(form mode env) →
- construct (p280) compConstruct(form mode env) →
- ListCategory (p282) compConstructorCategory(form mode env) →
- RecordCategory (p282) compConstructorCategory(form mode env) →
- UnionCategory (p282) compConstructorCategory(form mode env) →
- VectorCategory (p282) compConstructorCategory(form mode env) →
- DEF (p282) compDefine(form mode env) →
- elt (p300) compElt(form mode env) →
- exit (p301) compExit(form mode env) →
- has (p302) compHas(pred mode \$e) →
- IF (p304) compIf(form mode env) →
- import (p312) compImport(form mode env) →
- is (p312) compIs(form mode env) →
- Join (p313) compJoin(form mode env) →
- +-> (p315) compLambda(form mode env) →
- leave (p317) compLeave(form mode env) →
- MDEF (p317) compMacro(form mode env) →
- pretend (p318) compPretend →
- QUOTE (p320) compQuote(form mode env) →

- REDUCE (p320) compReduce(form mode env) →
- COLLECT (p323) compRepeatOrCollect(form mode env) →
- REPEAT (p323) compRepeatOrCollect(form mode env) →
- return (p325) compReturn(form mode env) →
- SEQ (p326) compSeq(form mode env) →
- LET (p329) compSetq(form mode env) →
- SETQ (p329) compSetq(form mode env) →
- String (p339) compString(form mode env) →
- SubDomain (p340) compSubDomain(form mode env) →
- SubsetCategory (p342) compSubsetCategory(form mode env) →
- | (p343) compSuchthat(form mode env) →
- VECTOR (p344) compVector(form mode env) →
- where (p345) compWhere(form mode eInit) →

5.2 Functions which handle == statements

5.2.1 defun compDefineAddSignature

```
[hasFullSignature p134]
[assoc p??]
[lassoc p??]
[getProplist p??]
[comp p557]
[$EmptyMode p131]
```

— **defun compDefineAddSignature** —

```
(defun |compDefineAddSignature| (form signature env)
  (let (sig declForm)
    (declare (special |$EmptyMode|))
    (if
      (and (setq sig (|hasFullSignature| (rest form) signature env))
           (null (|assoc| (cons '$ sig)
                          (|lassoc| '$ |modemap| (|getProplist| (car form) env)))))
      (progn
        (setq declForm
          (list '[:|
```

```

      (cons (car form)
            (loop for x in (rest form)
                  for m in (rest sig)
                  collect (list '|:| x m)))
      (car signature)))
    (third (|comp| declForm |$EmptyMode| env)))
  env)))

```

5.2.2 defun hasFullSignature

TPDHERE: test with BASTYPE [get p??]

— defun hasFullSignature —

```

(defun |hasFullSignature| (arg1 signature env)
  (let (target ml u)
    (setq target (first signature))
    (setq ml (rest signature))
    (when target
      (setq u
            (loop for x in arg1 for m in ml
                  collect (or m (|get| x '|mode| env) (return 'failed))))
      (unless (eq u 'failed) (cons target u)))))

```

5.2.3 defun addEmptyCapsuleIfNecessary

[kar p??]
 [\$SpecialDomainNames p??]

— defun addEmptyCapsuleIfNecessary —

```

(defun |addEmptyCapsuleIfNecessary| (target rhs)
  (declare (special |$SpecialDomainNames|) (ignore target))
  (if (member (kar rhs) |$SpecialDomainNames|)
      rhs
      (list '|add| rhs (list 'capsule))))

```

5.2.4 defun getTargetFromRhs

[stackSemanticError p??]
 [getTargetFromRhs p135]
 [compOrCroak p556]

— defun getTargetFromRhs —

```
(defun |getTargetFromRhs| (lhs rhs env)
  (declare (special |$EmptyMode|))
  (cond
    ((and (consp rhs) (eq (qfirst rhs) 'capsule))
     (|stackSemanticError|
      (list "target category of " lhs
            " cannot be determined from definition")
      nil))
    ((and (consp rhs) (eq (qfirst rhs) '|SubDomain|) (consp (qrest rhs)))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (consp rhs) (eq (qfirst rhs) '|add|)
          (consp (qrest rhs)) (consp (qcddr rhs))
          (eq (qcddr rhs) nil)
          (consp (qthird rhs))
          (eq (qcaaddr rhs) 'capsule))
     (|getTargetFromRhs| lhs (second rhs) env))
    ((and (consp rhs) (eq (qfirst rhs) '|Record|))
     (cons '|RecordCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|Union|))
     (cons '|UnionCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|List|))
     (cons '|ListCategory| (rest rhs)))
    ((and (consp rhs) (eq (qfirst rhs) '|Vector|))
     (cons '|VectorCategory| (rest rhs)))
    (t
     (second (|compOrCroak| rhs |$EmptyMode| env)))))
```

—

5.2.5 defun giveFormalParametersValues

[put p??]
 [get p??]

— defun giveFormalParametersValues —

```
(defun |giveFormalParametersValues| (argl env)
  (dolist (x argl)
    (setq env
```

```
(|put| x '|value|
  (list (|genSomeVariable|) (|get| x '|mode| env) nil) env)))
env)
```

5.2.6 defun macroExpandInPlace

[macroExpand p136]

— defun macroExpandInPlace —

```
(defun |macroExpandInPlace| (form env)
  (let (y)
    (setq y (|macroExpand| form env))
    (if (or (atom form) (atom y))
        y
        (progn
          (rplaca form (car y))
          (rplacd form (cdr y))
          form
          ))))
```

5.2.7 defun macroExpand

[macroExpand p136]

[macroExpandList p137]

— defun macroExpand —

```
(defun |macroExpand| (form env)
  (let (u)
    (cond
      ((atom form)
       (if (setq u (|get| form '|macro| env))
           (|macroExpand| u env)
           form))
      ((and (consp form) (eq (qfirst form) 'def)
            (consp (qrest form))
            (consp (qcddr form))
            (consp (qcdddr form))
            (consp (qcddddr form))
            (eq (qrest (qcddddr form)) nil)))
```

```
(list 'def (|macroExpand| (second form) env)
      (|macroExpandList| (third form) env)
      (|macroExpandList| (fourth form) env)
      (|macroExpand| (fifth form) env)))
(t (|macroExpandList| form env))))
```

5.2.8 defun macroExpandList

[macroExpand p136]
[getdatabase p??]

— defun macroExpandList —

```
(defun |macroExpandList| (lst env)
  (let (tmp)
    (if (and (consp lst) (eq (qrest lst) nil)
          (identp (qfirst lst)) (getdatabase (qfirst lst) 'niladic)
          (setq tmp (|get| (qfirst lst) '|macro| env)))
        (|macroExpand| tmp env)
        (loop for x in lst collect (|macroExpand| x env)))))
```

5.2.9 defun compDefineCategory1

[compDefineCategory2 p142]
[makeCategoryPredicates p138]
[compDefine1 p283]
[mkCategoryPackage p139]
[\$insideCategoryPackageIfTrue p??]
[\$EmptyMode p131]
[\$categoryPredicateList p??]
[\$lisplibCategory p??]
[\$bootStrapMode p??]

— defun compDefineCategory1 —

```
(defun |compDefineCategory1| (df mode env prefix fal)
  (let (|$insideCategoryPackageIfTrue| |$categoryPredicateList| form
        sig sc cat body categoryCapsule d tmp1 tmp3)
    (declare (special |$insideCategoryPackageIfTrue| |$EmptyMode|
                      |$categoryPredicateList| |$lisplibCategory|
```

```

                                |$bootStrapMode|))
;; a category is a DEF form with 4 parts:
;; ((DEF (|BasicType|) ((|Category|)) (NIL)
;;   (|add| (CATEGORY |domain| (SIGNATURE = ((|Boolean|) $ $))
;;         (SIGNATURE ~= ((|Boolean|) $ $)))
;;   (CAPSULE (DEF (~= |x| |y|) ((|Boolean|) $ $) (NIL NIL NIL)
;;   (IF (= |x| |y|) |false| |true|))))))
(setq form (second df))
(setq sig (third df))
(setq sc (fourth df))
(setq body (fifth df))
(setq categoryCapsule
  (when (and (consp body) (eq (qfirst body) '|add|)
            (consp (qrest body)) (consp (qcddr body))
            (eq (qcddr body) nil))
    (setq tmp1 (third body))
    (setq body (second body))
    tmp1))
(setq tmp3 (|compDefineCategory2| form sig sc body mode env prefix fal))
(setq d (first tmp3))
(setq mode (second tmp3))
(setq env (third tmp3))
(when (and categoryCapsule (null |$bootStrapMode|))
  (setq |$insideCategoryPackageIfTrue| t)
  (setq |$categoryPredicateList|
    (|makeCategoryPredicates| form |$lisplibCategory|))
  (setq env (third
    (|compDefine1|
      (|mkCategoryPackage| form cat categoryCapsule) |$EmptyMode| env))))
(list d mode env)))

```

5.2.10 defun makeCategoryPredicates

```

[$FormalMapVariableList p250]
[$TriangleVariableList p??]
[$mvl p??]
[$tv1 p??]

```

— defun makeCategoryPredicates —

```

(defun |makeCategoryPredicates| (form u)
  (labels (
    (fn (u pl)
      (declare (special |$tv1| |$mvl|))
      (cond

```

```

((and (consp u) (eq (qfirst u) '|Join|) (consp (qrest u)))
  (fn (car (reverse (qrest u))) pl))
((and (consp u) (eq (qfirst u) '|has|))
  (|insert| (eqsubstlist |$mvl| |$tv1| u) pl))
((and (consp u) (member (qfirst u) '(signature attribute))) pl)
(atom u) pl)
(t (fn1 u pl)))
(fn1 (u pl)
  (dolist (x u) (setq pl (fn x pl)))
  pl))
(declare (special |$FormalMapVariableList| |$mvl| |$tv1|
  |$TriangleVariableList|))
(setq |$tv1| (take (|#| (cdr form)) |$TriangleVariableList|))
(setq |$mvl| (take (|#| (cdr form)) (cdr |$FormalMapVariableList|)))
(fn u nil)))

```

5.2.11 defun mkCategoryPackage

```

[strconc p??]
[pname p??]
[getdatabase p??]
[abbreviationsSpad2Cmd p??]
[JoinInner p??]
[assoc p??]
[sublislis p??]
[msubst p??]
[$options p??]
[$categoryPredicateList p??]
[$e p??]
[$FormalMapVariableList p250]

```

— defun mkCategoryPackage —

```

(defun |mkCategoryPackage| (form cat def)
  (labels (
    (fn (x oplist)
      (cond
        ((atom x) oplist)
        ((and (consp x) (eq (qfirst x) 'def) (consp (qrest x)))
          (cons (second x) oplist))
        (t
          (fn (cdr x) (fn (car x) oplist))))))
    (gn (cat)
      (cond
        ((and (consp cat) (eq (qfirst cat) 'category)) (cddr cat))

```

```

      ((and (consp cat) (eq (qfirst cat) '|Join|)) (gn (|last| (qrest cat))))
      (t nil))))
(let (|$options| op argl packageName packageAbb nameForDollar packageArgl
      capsuleDefAlist explicitCatPart catvec fullCatOpList op1 sig
      catOpList packageCategory nils packageSig)
  (declare (special |$options| |$categoryPredicateList| |$e|
                    |$FormalMapVariableList|))
  (setq op (car form))
  (setq argl (cdr form))
  (setq packageName (intern (strconc (pname op) "&")))
  (setq packageAbb (intern (strconc (getdatabase op 'abbreviation) "-")))
  (setq |$options| nil)
  (|abbreviationsSpad2Cmd| (list '|domain| packageAbb packageName))
  (setq nameForDollar (car (setdifference '(s a b c d e f g h i) argl)))
  (setq packageArgl (cons nameForDollar argl))
  (setq capsuleDefAlist (fn def nil))
  (setq explicitCatPart (gn cat))
  (setq catvec (|eval| (|mkEvaluableCategoryForm| form)))
  (setq fullCatOpList (elt (|JoinInner| (list catvec) |$e|) 1))
  (setq catOpList
    (loop for x in fullCatOpList do
      (setq op1 (caar x))
      (setq sig (cadar x))
      when (|assoc| op1 capsuleDefAlist)
      collect (list 'signature op1 sig)))
  (when catOpList
    (setq packageCategory
      (cons 'category
        (cons '|domain| (sublislis argl |$FormalMapVariableList| catOpList))))
    (setq nils (loop for x in argl collect nil))
    (setq packageSig (cons packageCategory (cons form nils)))
    (setq |$categoryPredicateList|
      (msubst nameForDollar '$ |$categoryPredicateList|))
    (msubst nameForDollar '$
      (list 'def (cons packageName packageArgl)
        packageSig (cons nil nils) def))))))

```

5.2.12 defun mkEvaluableCategoryForm

```

[qcar p??]
[qcdr p??]
[mkEvaluableCategoryForm p140]
[compOrCroak p556]
[getdatabase p??]
[get p??]

```

```

[mkq p??]
[$Category p??]
[$e p??]
[$EmptyMode p131]
[$CategoryFrame p??]
[$Category p??]
[$CategoryNames p??]
[$e p??]

```

— defun mkEvaluableCategoryForm —

```

(defun |mkEvaluableCategoryForm| (c)
  (let (op arg1 tmp1 x m)
    (declare (special |$Category| |$e| |$EmptyMode| |$CategoryFrame|
                      |$CategoryNames|))
    (if (consp c)
        (progn
          (setq op (qfirst c))
          (setq arg1 (qrest c))
          (cond
            ((eq op '|Join|)
             (cons '|Join|
                   (loop for x in arg1
                        collect (|mkEvaluableCategoryForm| x))))
            ((eq op '|DomainSubstitutionMacro|)
             (|mkEvaluableCategoryForm| (cadr arg1)))
            ((eq op '|mkCategory|) c)
            ((member op |$CategoryNames|)
             (setq tmp1 (|compOrCroak| c |$EmptyMode| |$e|))
             (setq x (car tmp1))
             (setq m (cadr tmp1))
             (setq |$e| (caddr tmp1))
             (when (equal m |$Category|) x))
            ((or (eq (getdatabase op 'constructorkind) '|category|)
                  (|get| op '|isCategory| |$CategoryFrame|))
             (cons op
                   (loop for x in arg1
                        collect (mkq x))))
            (t
             (setq tmp1 (|compOrCroak| c |$EmptyMode| |$e|))
             (setq x (car tmp1))
             (setq m (cadr tmp1))
             (setq |$e| (caddr tmp1))
             (when (equal m |$Category|) x))))
      (mkq c))))

```

5.2.13 defun compDefineCategory2

```

[addBinding p??]
[getArgumentModeOrMoan p156]
[giveFormalParametersValues p135]
[take p??]
[sublis p??]
[compMakeDeclaration p593]
[nequal p??]
[opOf p??]
[optFunctorBody p??]
[compOrCroak p556]
[mkConstructor p170]
[compile p145]
[lisplibWrite p182]
[removeZeroOne p??]
[mkq p??]
[evalAndRwriteLispForm p169]
[eval p??]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[augLisplibModemapsFromCategory p157]
[$prefix p??]
[$formalArgList p??]
[$definition p??]
[$form p??]
[$op p??]
[$extraParms p??]
[$lisplibCategory p??]
[$FormalMapVariableList p250]
[$libFile p??]
[$TriangleVariableList p??]
[$lisplib p??]
[$formalArgList p??]
[$insideCategoryIfTrue p??]
[$top-level p??]
[$definition p??]
[$form p??]
[$op p??]
[$extraParms p??]
[$functionStats p??]
[$functorStats p??]
[$frontier p??]
[$getDomainCode p??]
[$addForm p??]

```

```

[$lisplibAbbreviation p??]
[$functorForm p??]
[$lisplibAncestors p??]
[$lisplibCategory p??]
[$lisplibParents p??]
[$lisplibModemap p??]
[$lisplibKind p??]
[$lisplibForm p??]
[$domainShell p??]

```

— **defun compDefineCategory2** —

```

(defun |compDefineCategory2|
  (form signature specialCases body mode env |$prefix| |$formalArgList|)
  (declare (special |$prefix| |$formalArgList|) (ignore specialCases))
  (let (|$insideCategoryIfTrue| $TOP_LEVEL |$definition| |$form| |$op|
        |$extraParms| |$functionStats| |$functorStats| |$frontier|
        |$getDomainCode| |$addForm| argl sargl aList signaturep opp formp
        formalBody formals actuals g fun pairlis parSignature parForm modemap)
    (declare (special |$insideCategoryIfTrue| $top_level |$definition|
                      |$form| |$op| |$extraParms| |$functionStats|
                      |$functorStats| |$frontier| |$getDomainCode|
                      |$addForm| |$lisplibAbbreviation| |$functorForm|
                      |$lisplibAncestors| |$lisplibCategory|
                      |$FormalMapVariableList| |$lisplibParents|
                      |$lisplibModemap| |$lisplibKind| |$lisplibForm|
                      $lisplib |$domainShell| |$libFile|
                      |$TriangleVariableList|))
      ; 1. bind global variables
      (setq |$insideCategoryIfTrue| t)
      (setq $top_level nil)
      (setq |$definition| nil)
      (setq |$form| nil)
      (setq |$op| nil)
      (setq |$extraParms| nil)
      ; 1.1 augment e to add declaration $: <form>
      (setq |$definition| form)
      (setq |$op| (car |$definition|))
      (setq argl (cdr |$definition|))
      (setq env (|addBinding| '$ (list (cons '|mode| |$definition|)) env))
      ; 2. obtain signature
      (setq signaturep
        (cons (car signature)
              (loop for a in argl
                    collect (|getArgumentModeOrMoan| a |$definition| env))))
      (setq env (|giveFormalParametersValues| argl env))
      ; 3. replace arguments by $1,..., substitute into body,
      ;    and introduce declarations into environment
      (setq sargl (take (|#| argl) |$TriangleVariableList|))

```

```

(setq |$form| (cons |$op| sargl))
(setq |$functorForm| |$form|)
(setq |$formalArgList| (append sargl |$formalArgList|))
(setq aList (loop for a in argl for sa in sargl collect (cons a sa)))
(setq formalBody (sublis aList body))
(setq signaturep (sublis aList signaturep))
; Begin lines for category default definitions
(setq |$functionStats| (list 0 0))
(setq |$functorStats| (list 0 0))
(setq |$frontier| 0)
(setq |$getDomainCode| nil)
(setq |$addForm| nil)
(loop for x in sargl for r in (rest signaturep)
  do (setq env (third (|compMakeDeclaration| (list '[:| x r] mode env)))))
; 4. compile body in environment of %type declarations for arguments
(setq opp |$op|)
(when (and (nequal (|opOf| formalBody) '|Join|)
           (nequal (|opOf| formalBody) '|mkCategory|))
  (setq formalBody (list '|Join| formalBody)))
(setq body
  (|optFunctorBody| (car (|compOrCroak| formalBody (car signaturep) env)))))
(when |$extraParms|
  (setq actuals nil)
  (setq formals nil)
  (loop for u in |$extraParms| do
    (setq formals (cons (car u) formals))
    (setq actuals (cons (mkq (cdr u)) actuals))))
(setq body
  (list '|sublisV| (list 'pair (list 'quote formals) (cons 'list actuals))
    body)))
; always subst for args after extraparms
(when argl
  (setq body
    (list '|sublisV|
      (list 'pair
        (list 'quote sargl)
        (cons 'list (loop for u in sargl collect (list '|devaluate| u))))
        body)))
  (setq body
    (list 'prog1 (list 'let (setq g (gensym)) body)
      (list 'setelt g 0 (|mkConstructor| |$form|))))
  (setq fun (|compile| (list opp (list 'lam sargl body)))))
; 5. give operator a 'modemap property
(setq pairlis
  (loop for a in argl for v in |$FormalMapVariableList|
    collect (cons a v)))
(setq parSignature (sublis pairlis signaturep))
(setq parForm (sublis pairlis form))
(|lisplibWrite| "compilerInfo"
  (|removeZeroOne|

```

```

      (list 'setq '|$CategoryFrame|
        (list '|put| (list 'quote opp) ''|isCategory| t
          (list '|addModemap| (mkq opp) (mkq parForm)
            (mkq parSignature) t (mkq fun) '|$CategoryFrame|))))
      |$libFile|)
(unless sarg1
  (|evalAndRwriteLispForm| 'niladic
    '(setf (get ',opp 'niladic) t)))
;; 6 put modemaps into InteractiveModemapFrame
(setq |$domainShell| (|eval| (cons opp (mapcar 'mkq sargl))))
(setq |$lisplibCategory| formalBody)
(when $lisplib
  (setq |$lisplibForm| form)
  (setq |$lisplibKind| '|category|)
  (setq modemap (list (cons parForm parSignature) (list t opp)))
  (setq |$lisplibModemap| modemap)
  (setq |$lisplibParents|
    (|getParentsFor| |$op| |$FormalMapVariableList| |$lisplibCategory|))
  (setq |$lisplibAncestors| (|computeAncestorsOf| |$form| nil))
  (setq |$lisplibAbbreviation| (|constructor?| |$op|))
  (setq formp (cons opp sargl))
  (|augLisplibModemapsFromCategory| formp formalBody signaturep))
(list fun '(|Category|) env)))

```

5.2.14 defun compile

```

[member p??]
[getmode p??]
[qcar p??]
[qcdr p??]
[get p??]
[modeEqual p357]
[userError p??]
[encodeItem p150]
[strconc p??]
[nequal p??]
[kar p??]
[encodeFunctionName p148]
[splitEncodedFunctionName p149]
[sayBrightly p??]
[optimizeFunctionDef p208]
[putInLocalDomainReferences p154]
[constructMacro p151]
[spadCompileOrSetq p151]

```

```

[elapsedTime p??]
[addStats p??]
[printStats p??]
[$functionStats p??]
[$macroIfTrue p??]
[$doNotCompileJustPrint p??]
[$insideCapsuleFunctionIfTrue p??]
[$saveableItems p??]
[$lisplibItemsAlreadyThere p??]
[$splitUpItemsAlreadyThere p??]
[$lisplib p??]
[$compileOnlyCertainItems p??]
[$functorForm p??]
[$signatureOfForm p??]
[$suffix p??]
[$prefix p??]
[$signatureOfForm p??]
[$e p??]
[$functionStats p??]
[$savableItems p??]
[$suffix p??]

```

— defun compile —

```

(defun |compile| (u)
  (labels (
    (isLocalFunction (op)
      (let (tmp1)
        (declare (special |$e| |$formalArgList|))
        (and (null (|member| op |$formalArgList|))
          (progn
            (setq tmp1 (|getmode| op |$e|))
            (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)))))))
    (let (op lamExpr DC sig sel opexport opmodes opp parts s tt unew
      optimizedBody stuffToCompile result functionStats)
      (declare (special |$functionStats| |$macroIfTrue| |$doNotCompileJustPrint|
        |$insideCapsuleFunctionIfTrue| |$saveableItems| |$e|
        |$lisplibItemsAlreadyThere| |$splitUpItemsAlreadyThere|
        |$compileOnlyCertainItems| $LISPLIB |$suffix|
        |$signatureOfForm| |$functorForm| |$prefix|
        |$savableItems|))
        (setq op (first u))
        (setq lamExpr (second u))
        (when |$suffix|
          (setq |$suffix| (1+ |$suffix|))
          (setq opp
            (progn
              (setq opexport nil)

```

```

(setq opmodes
(loop for item in (|get| op '|modemap| |$e|)
do
  (setq dc (caar item))
  (setq sig (cdar item))
  (setq sel (cadadr item))
  when (and (eq dc '$)
            (setq opexport t)
            (let ((result t))
              (loop for x in sig for y in |$signatureOfForm|
                do (setq result (|modeEqual| x y)))
              result))
    collect sel))
(cond
((isLocalFunction op)
 (when opexport
  (|userError| (list '|%b| op '|%d| " is local and exported"))
  (intern (strconc (|encodeItem| |$prefix|) ";" (|encodeItem| op))))
(t
 (|encodeFunctionName| op |$functorForm| |$signatureOfForm|
  '|;| |$suffix|))))
(setq u (list opp lamExpr))
(when (and $lisplib |$compileOnlyCertainItems|)
 (setq parts (|splitEncodedFunctionName| (elt u 0) '|;|))
 (cond
 ((eq parts '|inner|)
  (setq |$savableItems| (cons (elt u 0) |$savableItems|)))
 (t
  (setq unew nil)
  (loop for item in |$splitUpItemsAlreadyThere|
  do
    (setq s (first item))
    (setq tt (second item))
    (when
     (and (equal (elt parts 0) (elt s 0))
          (equal (elt parts 1) (elt s 1))
          (equal (elt parts 2) (elt s 2)))
      (setq unew tt)))
  (cond
  ((null unew)
   (|sayBrightly| (list " Error: Item did not previously exist"))
   (|sayBrightly| (cons " Item not saved: " (|bright| (elt u 0))))
   (|sayBrightly|
    (list " What's there is: " |$lisplibItemsAlreadyThere|))
   nil)
  (t
   (|sayBrightly| (list " Renaming " (elt u 0) " as " unew))
   (setq u (cons unew (cdr u)))
   (setq |$savableItems| (cons unew |$saveableItems|))))))
(setq optimizedBody (|optimizeFunctionDef| u))

```

```

(setq stuffToCompile
  (if |$insideCapsuleFunctionIfTrue|
    (|putInLocalDomainReferences| optimizedBody)
    optimizedBody))
(cond
  ((eq |$doNotCompileJustPrint| t)
    (prettyprint stuffToCompile)
    opp)
  (|$macroIfTrue| (|constructMacro| stuffToCompile))
  (t
    (setq result (|spadCompileOrSetq| stuffToCompile))
    (setq functionStats (list 0 (|elapsedTime|)))
    (setq |$functionStats| (|addStats| |$functionStats| functionStats))
    (|printStats| functionStats)
    result))))

```

5.2.15 defun encodeFunctionName

Code for encoding function names inside package or domain [msubst p??]

```

[mkRepetitionAssoc p149]
[encodeItem p150]
[stringimage p??]
[internal p??]
[getAbbreviation p285]
[length p??]
[$lisplib p??]
[$lisplibSignatureAlist p??]
[$lisplibSignatureAlist p??]

```

— defun encodeFunctionName —

```

(defun |encodeFunctionName| (fun package signature sep count)
  (let (packageName arglist signaturep reducedSig n x encodedSig encodedName)
    (declare (special |$lisplibSignatureAlist| $lisplib))
    (setq packageName (car package))
    (setq arglist (cdr package))
    (setq signaturep (msubst '$ package signature))
    (setq reducedSig
      (|mkRepetitionAssoc| (append (cdr signaturep) (list (car signaturep)))))
    (setq encodedSig
      (let ((result ""))
        (loop for item in reducedSig
          do
            (setq n (car item))
            (setq x (cdr item))

```

```

      (setq result
        (strconc result
          (if (eql n 1)
              (|encodeItem| x)
              (strconc (stringimage n) (|encodeItem| x))))))
      result))
(setq encodedName
  (internl (|getAbbreviation| packageName (|#| arglist))
    '|;| (|encodeItem| fun) '|;| encodedSig sep (stringimage count)))
(when $lisplib
  (setq |$lisplibSignatureAlist|
    (cons (cons encodedName signaturep) |$lisplibSignatureAlist|)))
encodedName))

```

5.2.16 defun mkRepititionAssoc

```

[qcar p??]
[qcdr p??]

```

— defun mkRepititionAssoc —

```

(defun |mkRepititionAssoc| (z)
  (labels (
    (mkRepfun (z n)
      (cond
        ((null z) nil)
        ((and (consp z) (eq (qrest z) nil) (list (cons n (qfirst z)))))
        ((and (consp z) (consp (qrest z)) (equal (qsecond z) (qfirst z)))
          (mkRepfun (cdr z) (1+ n)))
        (t (cons (cons n (car z)) (mkRepfun (cdr z) 1))))))
    (mkRepfun z 1)))

```

5.2.17 defun splitEncodedFunctionName

```

[stringimage p??]
[strpos p??]

```

— defun splitEncodedFunctionName —

```

(defun |splitEncodedFunctionName| (encodedName sep)
  (let (sep0 p1 p2 p3 s1 s2 s3 s4)

```

```

; sep0 is the separator used in "encodeFunctionName".
(setq sep0 ";")
(unless (stringp encodedName) (setq encodedName (stringimage encodedName)))
(cond
  ((null (setq p1 (strpos sep0 encodedName 0 "*"))) nil)
  ; This is picked up in compile for inner functions in partial compilation
  ((null (setq p2 (strpos sep0 encodedName (1+ p1) "*"))) '|inner|)
  ((null (setq p3 (strpos sep encodedName (1+ p2) "*"))) nil)
  (t
   (setq s1 (substring encodedName 0 p1))
   (setq s2 (substring encodedName (1+ p1) (- p2 p1 1)))
   (setq s3 (substring encodedName (1+ p2) (- p3 p2 1)))
   (setq s4 (substring encodedName (1+ p3) nil))
   (list s1 s2 s3 s4))))

```

5.2.18 defun encodeItem

```

[getCaps p150]
[identp p??]
[qcar p??]
[pname p??]
[stringimage p??]

```

— defun encodeItem —

```

(defun |encodeItem| (x)
  (cond
    ((consp x) (|getCaps| (qfirst x)))
    ((identp x) (pname x))
    (t (stringimage x))))

```

5.2.19 defun getCaps

```

[stringimage p??]
[maxindex p??]
[l-case p??]
[strconc p??]

```

— defun getCaps —

```

(defun |getCaps| (x)

```

```

(let (s c clist tmp1)
  (setq s (stringimage x))
  (setq clist
    (loop for i from 0 to (maxindex s)
      when (upper-case-p (setq c (elt s i)))
      collect c))
  (cond
    ((null clist) "_")
    (t
     (setq tmp1
       (cons (first clist) (loop for u in (rest clist) collect (l-case u))))
     (let ((result ""))
       (loop for u in tmp1
         do (setq result (strconc result u)))
       result)))))

```

5.2.20 defun constructMacro

constructMacro (form is [nam,[lam,vl,body]]) [stackSemanticError p??]
 [identp p??]

— defun constructMacro —

```

(defun |constructMacro| (form)
  (let (vl body)
    (setq vl (cadadr form))
    (setq body (car (cddadr form)))
    (cond
      ((null (let ((result t))
        (loop for x in vl
          do (setq result (and result (atom x))))
        result))
       (|stackSemanticError| (list '|illegal parameters for macro: | vl) nil))
      (t
       (list 'xlam (loop for x in vl when (identp x) collect x) body)))))

```

5.2.21 defun spadCompileOrSetq

[qcar p??]
 [qcdr p??]
 [contained p??]

```

[sayBrightly p??]
[bright p??]
[LAM,EVALANDFILEACTQ p??]
[mkq p??]
[comp p557]
[compileConstructor p153]
[$insideCapsuleFunctionIfTrue p??]

```

— **defun spadCompileOrSetq** —

```

(defun |spadCompileOrSetq| (form)
  (let (nam lam vl body namp tmp1 e vlp macform)
    (declare (special |$insideCapsuleFunctionIfTrue|))
    (setq nam (car form))
    (setq lam (caadr form))
    (setq vl (cadadr form))
    (setq body (car (cddadr form)))
    (cond
      ((and (consp vl) (progn (setq tmp1 (reverse vl)) t)
        (consp tmp1)
        (progn
          (setq e (qfirst tmp1))
          (setq vlp (qrest tmp1))
          t)
        (progn (setq vlp (nreverse vlp)) t)
        (consp body)
        (progn (setq namp (qfirst body)) t)
        (equal (qrest body) vlp))
      (|LAM,EVALANDFILEACTQ|
       (list 'put (mkq nam) (mkq '|SPADreplace|) (mkq namp)))
      (|sayBrightly|
       (cons " " (append (|bright| nam)
        (cons "is replaced by" (|bright| namp))))))
    ((and (or (atom body)
      (let ((result t))
        (loop for x in body
          do (setq result (and result (atom x))))
        result))
      (consp vl)
      (progn (setq tmp1 (reverse vl)) t)
      (consp tmp1)
      (progn
        (setq e (qfirst tmp1))
        (setq vlp (qrest tmp1))
        t)
      (progn (setq vlp (nreverse vlp)) t)
      (null (contained e body)))
      (setq macform (list 'xlam vlp body))
      (|LAM,EVALANDFILEACTQ|

```

```

      (list 'put (mkq nam) (mkq '|SPADreplace|) (mkq macform)))
    (|sayBrightly| (cons "      " (append (|bright| nam)
      (cons "is replaced by" (|bright| body))))))
    (t nil))
  (if |$insideCapsuleFunctionIfTrue|
    (car (comp (list form)))
    (|compileConstructor| form)))

```

5.2.22 defun compileConstructor

```

[compileConstructor1 p153]
[clearClams p??]

```

— defun compileConstructor —

```

(defun |compileConstructor| (form)
  (let (u)
    (setq u (|compileConstructor1| form))
    (|clearClams|
     u))

```

5.2.23 defun compileConstructor1

```

[getdatabase p??]
[compAndDefine p??]
[comp p557]
[clearConstructorCache p??]
[$mutableDomain p??]
[$ConstructorCache p??]
[$clamList p??]
[$clamList p??]

```

— defun compileConstructor1 —

```

(defun |compileConstructor1| (form)
  (let (|$clamList| fn key vl body1 lambdaOrSlam compForm u)
    (declare (special |$clamList| |$ConstructorCache| |$mutableDomain|))
    (setq fn (car form))
    (setq key (caadr form))
    (setq vl (cadadr form))

```

```

(setq body1 (cddadr form))
(setq |$clamList| nil)
(setq lambdaOrSlam
  (cond
    ((eq (getdatabase fn 'constructorkind) '|category|) 'spadslam)
    (|$mutableDomain| 'lambda)
  (t
    (setq |$clamList|
      (cons (list fn '|$ConstructorCache| '|domainEqualList| '|count|)
        |$clamList|)))
    'lambda)))
(setq compForm (list (list fn (cons lambdaOrSlam (cons vl body1)))))
(if (eq (getdatabase fn 'constructorkind) '|category|)
  (setq u (|compAndDefine| compForm))
  (setq u (comp compForm)))
(|clearConstructorCache| fn)
(car u))

```

5.2.24 defun putInLocalDomainReferences

[NRTputInTail p154]
 [\$QuickCode p??]
 [\$elt p300]

— defun putInLocalDomainReferences —

```

(defun |putInLocalDomainReferences| (def)
  (let (|$elt| opName lam varl body)
    (declare (special |$elt| |$QuickCode|))
    (setq opName (car def))
    (setq lam (caadr def))
    (setq varl (cadadr def))
    (setq body (car (cddadr def)))
    (setq |$elt| (if |$QuickCode| 'qrefelt 'elt))
    (|NRTputInTail| (cddadr def))
    def))

```

5.2.25 defun NRTputInTail

[lassoc p??]
 [NRTassocIndex p336]

```
[rplaca p??]
[NRTputInHead p155]
[$elt p300]
[$devalueateList p??]
```

— **defun NRTputInTail** —

```
(defun |NRTputInTail| (x)
  (let (u k)
    (declare (special |$elt| |$devalueateList|))
    (maplist #'(lambda (y)
      (cond
        ((atom (setq u (car y)))
         (cond
           ((or (eq u '$) (lassoc u |$devalueateList|))
            nil)
           ((setq k (|NRTassocIndex| u))
            (cond
              ; u atomic means that the slot will always contain a vector
              ((atom u) (rplaca y (list |$elt| '$ k)))
              ; this reference must check that slot is a vector
              (t (rplaca y (list 'spadcheckelt '$ k))))
            (t nil))))
        (t (|NRTputInHead| u))))
      x)
    x))
```

—————

5.2.26 defun NRTputInHead

```
[NRTputInTail p154]
[NRTassocIndex p336]
[NRTputInHead p155]
[lastnode p??]
[keyedSystemError p??]
[$elt p300]
```

— **defun NRTputInHead** —

```
(defun |NRTputInHead| (bod)
  (let (fn clauses dom tmp2 ind k)
    (declare (special |$elt|))
    (cond
      ((atom bod) bod)
      ((and (consp bod) (eq (qcar bod) 'spadcall) (consp (qcdr bod))
        (progn (setq tmp2 (reverse (qcdr bod))) t) (consp tmp2))
```

```

(setq fn (qcar tmp2))
(|NRTputInTail| (cdr bod))
(cond
  ((and (consp fn) (consp (qcdr fn)) (consp (qcdr (qcdr fn)))
        (eq (qcddr fn) nil) (null (eq (qsecond fn) '$))
        (member (qcar fn) '(elt qrefelt const)))
    (when (setq k (|NRTassocIndex| (qsecond fn)))
      (rplaca (lastnode bod) (list |$elt| '$ k)))
    (t (|NRTputInHead| fn) bod)))
((and (consp bod) (eq (qcar bod) 'cond))
 (setq clauses (qcdr bod))
 (loop for cc in clauses do (|NRTputInTail| cc))
 bod)
((and (consp bod) (eq (qcar bod) 'quote)) bod)
((and (consp bod) (eq (qcar bod) 'closedfn)) bod)
((and (consp bod) (eq (qcar bod) 'spadconst) (consp (qcdr bod))
      (consp (qcddr bod)) (eq (qcddr bod) nil))
 (setq dom (qsecond bod))
 (setq ind (qthird bod))
 (rplaca bod |$elt|)
 (cond
   ((eq dom '$) nil)
   ((setq k (|NRTassocIndex| dom))
    (rplaca (lastnode bod) (list |$elt| '$ k))
    bod)
   (t
    (|keyedSystemError| 'S2GE0016
     (list "NRTputInHead" "unexpected SPADCONST form")))))
(t
 (|NRTputInHead| (car bod))
 (|NRTputInTail| (cdr bod) bod))))

```

5.2.27 defun getArgumentModeOrMoan

[getArgumentMode p299]
 [stackSemanticError p??]

— defun getArgumentModeOrMoan —

```

(defun |getArgumentModeOrMoan| (x form env)
  (or (|getArgumentMode| x env)
      (|stackSemanticError|
       (list 'argument | x | of | form | is not declared) nil)))

```

5.2.28 defun augLisplibModemapsFromCategory

```
[sublis p??]
[mkAlistOfExplicitCategoryOps p158]
[isCategoryForm p??]
[lassoc p??]
[member p??]
[mkpf p??]
[interactiveModemapForm p160]
[$lisplibModemapAlist p??]
[$EmptyEnvironment p??]
[$domainShell p??]
[$PatternVariableList p??]
[$lisplibModemapAlist p??]
```

— **defun augLisplibModemapsFromCategory** —

```
(defun |augLisplibModemapsFromCategory| (form body signature)
  (let (argl sl opAlist nonCategorySigAlist domainList catPredList op sig
        pred sel predp modemap)
    (declare (special |$lisplibModemapAlist| |$EmptyEnvironment|
                      |$domainShell| |$PatternVariableList|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq sl
      (cons (cons '$ '*1)
            (loop for a in argl for p in (rest |$PatternVariableList|)
                  collect (cons a p))))
    (setq form (sublis sl form))
    (setq body (sublis sl body))
    (setq signature (sublis sl signature))
    (when (setq opAlist (sublis sl (elt |$domainShell| 1)))
      (setq nonCategorySigAlist
        (|mkAlistOfExplicitCategoryOps| (msubst '*1 '$ body)))
      (setq domainList
        (loop for a in (rest form) for m in (rest signature)
              when (|isCategoryForm| m |$EmptyEnvironment|)
                collect (list a m)))
      (setq catPredList
        (loop for u in (cons (list '*1 form) domainList)
              collect (cons '|ofCategory| u)))
      (loop for entry in opAlist
            when (|member| (cadar entry) (lassoc (caar entry) nonCategorySigAlist))
              do
                (setq op (caar entry))
                (setq sig (cadar entry))
                (setq pred (cadr entry))
                (setq sel (caddr entry))
                (setq predp (mkpf (cons pred catPredList) 'and)))
```

```
(setq modemap (list (cons '*1 sig) (list predp sel)))
(setq |$lisplibModemapAlist|
  (cons (cons op (|interactiveModemapForm| modemap))
        |$lisplibModemapAlist|))))
```

5.2.29 defun mkAlistOfExplicitCategoryOps

```
[qcar p??]
[qcdr p??]
[keyedSystemError p??]
[union p??]
[mkAlistOfExplicitCategoryOps p158]
[flattenSignatureList p159]
[nreverse0 p??]
[remdup p??]
[assocleft p??]
[isCategoryForm p??]
[$e p??]
```

— defun mkAlistOfExplicitCategoryOps —

```
(defun |mkAlistOfExplicitCategoryOps| (target)
  (labels (
    (atomizeOp (op)
      (cond
        ((atom op) op)
        ((and (consp op) (eq (qrest op) nil)) (qfirst op))
        (t (|keyedSystemError| 'S2GE0016
            (list "mkAlistOfExplicitCategoryOps" "bad signature")))))
    (fn (op u)
      (if (and (consp u) (consp (qfirst u)))
          (if (equal (qcar u) op)
              (cons (qcdar u) (fn op (qrest u)))
              (fn op (qrest u))))))
  (let (z tmp1 op sig u opList)
    (declare (special |$e|))
    (when (and (consp target) (eq (qfirst target) '|add|) (consp (qrest target)))
      (setq target (second target)))
    (cond
      ((and (consp target) (eq (qfirst target) '|Join|))
       (setq z (qrest target))
       (PROG (tmp1)
        (RETURN
         (DO ((G167566 z (CDR G167566)) (cat nil))
              ((OR (ATOM G167566) (PROGN (setq cat (CAR G167566)) nil))
```

```

      tmp1)
      (setq tmp1 (|union| tmp1 (|mkAlistOfExplicitCategoryOps| cat))))))
((and (consp target) (eq (qfirst target) 'category)
  (progn
    (setq tmp1 (qrest target))
    (and (consp tmp1)
      (progn (setq z (qrest tmp1)) t))))
  (setq z (|flattenSignatureList| (cons 'progn z)))
  (setq u
    (prog (G167577)
      (return
        (do ((G167583 z (cdr G167583)) (x nil))
          ((or (atom G167583)) (nreverse0 G167577))
          (setq x (car G167583))
          (cond
            ((and (consp x) (eq (qfirst x) 'signature) (consp (qrest x))
              (consp (qcddr x)))
              (setq op (qsecond x))
              (setq sig (qthird x))
              (setq G167577 (cons (cons (atomizeOp op) sig) G167577)))))))
    (setq opList (remdup (assocleft u)))
    (prog (G167593)
      (return
        (do ((G167598 opList (cdr G167598)) (x nil))
          ((or (atom G167598)) (nreverse0 G167593))
          (setq x (car G167598))
          (setq G167593 (cons (cons x (fn x u)) G167593))))))
    ((|isCategoryForm| target |$e|) nil)
    (t
      (|keyedSystemError| 'S2GE0016
        (list "mkAlistOfExplicitCategoryOps" "bad signature"))))))

```

5.2.30 defun flattenSignatureList

```

[qcar p??]
[qcdr p??]
[flattenSignatureList p159]

```

— defun flattenSignatureList —

```

(defun |flattenSignatureList| (x)
  (let (zz)
    (cond
      ((atom x) nil)
      ((and (consp x) (eq (qfirst x) 'signature)) (list x))

```

```

((and (consp x) (eq (qfirst x) 'if) (consp (qrest x))
      (consp (qcddr x)) (consp (qcdddr x))
      (eq (qcdddr x) nil))
 (append (|flattenSignatureList| (third x))
         (|flattenSignatureList| (fourth x))))
((and (consp x) (eq (qfirst x) 'progn))
 (loop for x in (qrest x)
       do
         (if (and (consp x) (eq (qfirst x) 'signature))
             (setq zz (cons x zz))
             (setq zz (append (|flattenSignatureList| x) zz))))
 zz)
(t nil)))

```

5.2.31 defun interactiveModemapForm

Create modemap form for use by the interpreter. This function replaces all specific domains mentioned in the modemap with pattern variables, and predicates [qcar p??]

```

[qcdr p??]
[nequal p??]
[replaceVars p161]
[modemapPattern p169]
[substVars p168]
[fixUpPredicate p161]
[$PatternVariableList p??]
[$FormalMapVariableList p250]

```

— defun interactiveModemapForm —

```

(defun |interactiveModemapForm| (mm)
  (labels (
    (fn (x)
      (if (and (consp x) (consp (qrest x))
              (consp (qcddr x)) (eq (qcdddr x) nil)
              (nequal (qfirst x) '|isFreeFunction|)
              (atom (qthird x)))
          (list (first x) (second x) (list (third x))
                x)))
    (let (pattern dc sig mmpat patternAlist partial patvars
          domainPredicateList tmp1 pred dependList cond)
      (declare (special |$PatternVariableList| |$FormalMapVariableList|))
      (setq mm
            (|replaceVars| (copy mm) |$PatternVariableList| |$FormalMapVariableList|))
      (setq pattern (car mm))
      (setq dc (caar mm))

```

```

(setq sig (cdar mm))
(setq pred (cadr mm))
(setq pred
  (prog ()
    (return
      (do ((x pred (cdr x)) (result nil))
        ((atom x) (nreverse0 result))
        (setq result (cons (fn (car x)) result)))))))
(setq tmp1 (|modemapPattern| pattern sig))
(setq mmpat (car tmp1))
(setq patternAlist (cadr tmp1))
(setq partial (caddr tmp1))
(setq patvars (caddr tmp1))
(setq tmp1 (|substVars| pred patternAlist patvars))
(setq pred (car tmp1))
(setq domainPredicateList (cadr tmp1))
(setq tmp1 (|fixUpPredicate| pred domainPredicateList partial (cdr mmpat)))
(setq pred (car tmp1))
(setq dependList (cdr tmp1))
(setq cond (car pred))
(list mmpat cond))))

```

5.2.32 defun replaceVars

Replace every identifier in oldvars with the corresponding identifier in newvars in the expression x [msubst p??]

— defun replaceVars —

```

(defun |replaceVars| (x oldvars newvars)
  (loop for old in oldvars for new in newvars
    do (setq x (msubst new old x)))
  x)

```

5.2.33 defun fixUpPredicate

```

[qcar p??]
[qcdr p??]
[length p??]
[orderPredicateItems p162]
[moveORsOutside p167]

```

— defun fixUpPredicate —

```
(defun |fixUpPredicate| (predClause domainPreds partial sig)
  (let (predicate fn skip predicates tmp1 dependList pred)
    (setq predicate (car predClause))
    (setq fn (cadr predClause))
    (setq skip (caddr predClause))
    (cond
      ((eq (car predicate) 'and)
       (setq predicates (append domainPreds (cdr predicate))))
      ((nequal predicate (mkq t))
       (setq predicates (cons predicate domainPreds)))
      (t
       (setq predicates (or domainPreds (list predicate)))))
    (cond
      (> (|#| predicates) 1)
      (setq pred (cons 'and predicates))
      (setq tmp1 (|orderPredicateItems| pred sig skip))
      (setq pred (car tmp1))
      (setq dependList (cdr tmp1))
      tmp1)
      (t
       (setq pred (|orderPredicateItems| (car predicates) sig skip))
       (setq dependList
        (when (and (consp pred) (eq (qfirst pred) '|isDomain|)
                   (consp (qrest pred)) (consp (qcddr pred))
                   (eq (qcdddr pred) nil)
                   (consp (qthird pred))
                   (eq (qcdaddr pred) nil))
              (list (second pred)))))
       (setq pred (|moveORsOutside| pred))
       (when partial (setq pred (cons '|partial| pred)))
       (cons (cons pred (cons fn skip)) dependList)))
```

—————

5.2.34 defun orderPredicateItems

```
[qcar p??]
[qcdr p??]
[signatureTran p163]
[orderPredTran p163]
```

— defun orderPredicateItems —

```
(defun |orderPredicateItems| (pred1 sig skip)
```

```
(let (pred)
  (setq pred (|signatureTran| pred1))
  (if (and (consp pred) (eq (qfirst pred) 'and))
      (|orderPredTran| (qrest pred) sig skip)
      pred)))
```

5.2.35 defun signatureTran

```
[signatureTran p163]
[isCategoryForm p??]
[$e p??]
```

— defun signatureTran —

```
(defun |signatureTran| (pred)
  (declare (special |$e|))
  (cond
    ((atom pred) pred)
    ((and (consp pred) (eq (qfirst pred) '|has|) (CONSP (qrest pred))
          (consp (qcddr pred))
          (eq (qcdddr pred) nil)
          (|isCategoryForm| (third pred) |$e|))
     (list '|ofCategory| (second pred) (third pred)))
    (t
     (loop for p in pred
           collect (|signatureTran| p))))))
```

5.2.36 defun orderPredTran

```
[qcar p??]
[qcdr p??]
[member p??]
[delete p??]
[unionq p??]
[listOfPatternIds p??]
[intersectionq p??]
[setdifference p??]
[insertWOC p??]
[isDomainSubst p166]
```

— defun orderPredTran —

```

(defun |orderPredTran| (oldList sig skip)
  (let (lastDependList somethingDone lastPreds indepvl depvl dependList
        noldList x ids fullDependList newList answer)
    ; --(1) make two kinds of predicates appear last:
    ; ----- (op *target ..) when *target does not appear later in sig
    ; ----- (isDomain *1 ..)
    (SEQ
      (loop for pred in oldList
        do (cond
          ((or (and (consp pred) (consp (qrest pred))
                    (consp (qcddr pred))
                    (eq (qcddr pred) nil)
                    (member (qfirst pred) '(|isDomain| |ofCategory|))
                    (equal (qsecond pred) (car sig))
                    (null (|member| (qsecond pred) (cdr sig))))
              (and (null skip) (consp pred) (eq (qfirst pred) '(|isDomain|
                    (consp (qrest pred)) (consp (qcddr pred))
                    (eq (qcddr pred) nil)
                    (equal (qsecond pred) '*1))))
              (setq oldList (|delete| pred oldList))
              (setq lastPreds (cons pred lastPreds))))))
      ; --(2a) lastDependList=list of all variables that lastPred forms depend upon
      (setq lastDependList
        (let (result)
          (loop for x in lastPreds
            do (setq result (unionq result (|listOfPatternIds| x))))
          result))
      ; --(2b) dependList=list of all variables that isDom/ofCat forms depend upon
      (setq dependList
        (let (result)
          (loop for x in oldList
            do (when
              (and (consp x)
                   (or (eq (qfirst x) '(|isDomain|) (eq (qfirst x) '(|ofCategory|))
                       (consp (qrest x)) (consp (qcddr x))
                       (eq (qcddr x) nil))
                   (setq result (unionq result (|listOfPatternIds| (third x))))))
              result))
      ; --(3a) newList= list of ofCat/isDom entries that don't depend on
      (loop for x in oldList
        do
          (cond
            ((and (consp x)
                  (or (eq (qfirst x) '(|ofCategory|) (eq (qfirst x) '(|isDomain|))
                      (consp (qrest x)) (consp (qcddr x))
                      (eq (qcddr x) nil))
                  (setq indepvl (|listOfPatternIds| (second x)))
                  (setq depvl (|listOfPatternIds| (third x))))
            (t
             (setq indepvl (|listOfPatternIds| x))

```

```

        (setq depvl nil)))
    (when
      (and (null (intersectionq indepvl dependList))
            (intersectionq indepvl lastDependList))
        (setq somethingDone t)
        (setq lastPreds (append lastPreds (list x)))
        (setq oldList (delete x oldList))))
; --(3b) newList= list of ofCat/isDom entries that don't depend on
(loop while oldList do
  (loop for x in oldList do
    (cond
      ((and (consp x)
            (or (eq (qfirst x) '|ofCategory|) (eq (qfirst x) '|isDomain|))
            (consp (qrest x))
            (consp (qcddr x)) (eq (qcdddr x) nil))
            (setq indepvl (|listOfPatternIds| (second x)))
            (setq depvl (|listOfPatternIds| (third x))))
      (t
       (setq indepvl (|listOfPatternIds| x))
       (setq depvl nil)))
      (when (null (intersectionq indepvl dependList))
        (setq dependList (SETDIFFERENCE dependList depvl))
        (setq newList (APPEND newList (list x))))))
; --(4) noldList= what is left over
(cond
  ((equal (setq noldList (setdifference oldList newList)) oldList)
   (setq newList (APPEND newList oldList))
   (return nil))
  (t
   (setq oldList noldList))))
(loop for pred in newList do
  (when
    (and (consp pred)
         (or (eq (qfirst pred) '|isDomain|) (eq (qfirst x) '|ofCategory|))
         (consp (qrest pred))
         (consp (qcddr pred))
         (eq (qcdddr pred) nil))
        (setq ids (|listOfPatternIds| (third pred)))
        (when
          (let (result)
            (loop for id in ids do
              (setq result (and result (|member| id fullDependList))))
            result)
          (setq fullDependList (|insertWOC| (second pred) fullDependList))
          (setq fullDependList (unionq fullDependList ids))))
        (setq newList (append newList lastPreds))
        (setq newList (|isDomainSubst| newList))
        (setq answer
         (cons (cons 'and newList) (intersectionq fullDependList sig))))))

```

5.2.37 defun isDomainSubst

— defun isDomainSubst —

```

(defun |isDomainSubst| (u)
  (labels (
    (findSub (x alist)
      (cond
        ((null alist) nil)
        ((and (consp alist) (consp (qfirst alist))
              (eq (qcaar alist) '|isDomain|)
              (consp (qcddar alist))
              (consp (qcdddr alist))
              (eq (qcdddr alist) nil)
              (equal x (cadar alist))))
          (caddar alist))
        (t (findSub x (cdr alist))))))
    (fn (x alist)
      (let (s)
        (declare (special |$PatternVariableList|))
        (if (atom x)
          (if
            (and (identp x)
                 (member x |$PatternVariableList|)
                 (setq s (findSub x alist)))
            s
            x)
          (cons (car x)
                (loop for y in (cdr x)
                      collect (fn y alist))))))
      (let (head tail nhead)
        (if (consp u)
          (progn
            (setq head (qfirst u))
            (setq tail (qrest u))
            (setq nhead
              (cond
                ((and (consp head) (eq (qfirst head) '|isDomain|)
                      (consp (qrest head)) (consp (qcddr head))
                      (eq (qcdddr head) nil))
                 (list '|isDomain| (second head)
                       (fn (third head) tail)))
                (t head)))
            (cons nhead (|isDomainSubst| (cdr u))))
          u))))

```

5.2.38 defun moveORsOutside

[moveORsOutside p167]

— defun moveORsOutside —

```

(defun |moveORsOutside| (p)
  (let (q x)
    (cond
      ((and (consp p) (eq (qfirst p) 'and))
        (setq q
              (prog (G167169)
                    (return
                     (do ((G167174 (cdr p) (cdr G167174)) (|r| nil))
                         ((or (atom G167174)) (nreverse0 G167169))
                         (setq |r| (CAR G167174))
                         (setq G167169 (cons (|moveORsOutside| |r|) G167169)))))))
      (t
       (cond
         ((setq x
              (let (tmp1)
                (loop for r in q
                      when (and (consp r) (eq (qfirst r) 'or))
                      do (setq tmp1 (or tmp1 r)))
                tmp1))
          (|moveORsOutside|
           (cons 'or
                 (let (tmp1)
                   (loop for tt in (cdr x)
                         do (setq tmp1 (cons (cons 'and (msubst tt x q)) tmp1)))
                   (nreverse0 tmp1))))))
         (t (cons 'and q))))
    (t p))))

;(defun |moveORsOutside| (p)
;  (let (q s x tmp1)
;    (cond
;      ((and (consp p) (eq (qfirst p) 'and))
;       (setq q (loop for r in (qrest p) collect (|moveORsOutside| r)))
;       (setq tmp1
;             (loop for r in q
;                   when (and (consp r) (eq (qrest r) 'or))
;                   collect r))
;       (setq x (mapcar #'(lambda (a b) (or a b)) tmp1))
;       (if x
;         (|moveORsOutside|
;          (cons 'or
;                (loop for tt in (cdr x)

```

```

;      collect (cons 'and (msubst tt x q))))
;      (cons 'and q)))
;      ('t p))))

```

5.2.39 defun substVars

Make pattern variable substitutions. [msubst p??]

[nsubst p??]

[contained p??]

[\$FormalMapVariableList p250]

— defun substVars —

```

(defun |substVars| (pred patternAlist patternVarList)
  (let (patVar value everything replacementVar domainPredicates)
    (declare (special |$FormalMapVariableList|))
    (setq domainPredicates NIL)
    (maplist
     #'(lambda (x)
         (setq patVar (caar x))
         (setq value (cdar x))
         (setq pred (msubst patVar value pred))
         (setq patternAlist (|nsubst| patVar value patternAlist))
         (setq domainPredicates (msubst patVar value domainPredicates))
         (unless (member value |$FormalMapVariableList|)
           (setq domainPredicates
                (cons (list 'isDomain| patVar value) domainPredicates))))
      patternAlist)
    (setq everything (list pred patternAlist domainPredicates))
    (dolist (|var| |$FormalMapVariableList|)
      (cond
       ((contained |var| everything)
        (setq replacementVar (car patternVarList))
        (setq patternVarList (cdr patternVarList))
        (setq pred (msubst replacementVar |var| pred))
        (setq domainPredicates
             (msubst replacementVar |var| domainPredicates))))
      (list pred domainPredicates)))

```

5.2.40 defun modemapPattern

```
[qcar p??]
[qcdr p??]
[rassoc p??]
[$PatternVariableList p??]
```

— defun modemapPattern —

```
(defun |modemapPattern| (mmPattern sig)
  (let (partial patvar patvars mmpat patternAlist)
    (declare (special |$PatternVariableList|))
    (setq patternAlist nil)
    (setq mmpat nil)
    (setq patvars |$PatternVariableList|)
    (setq partial nil)
    (maplist
     #'(lambda (xTails)
        (let ((x (car xTails)))
          (when (and (consp x) (eq (qfirst x) '|Union|)
                     (consp (qrest x)) (consp (qcddr x))
                     (eq (qcdddr x) nil)
                     (equal (third x) "failed")
                     (equal xTails sig))
            (setq x (second x))
            (setq partial t))
          (setq patvar (|rassoc| x patternAlist))
          (cond
           ((null (null patvar))
            (setq mmpat (cons patvar mmpat)))
           (t
            (setq patvar (car patvars))
            (setq patvars (cdr patvars))
            (setq mmpat (cons patvar mmpat))
            (setq patternAlist (cons (cons patvar x) patternAlist))))))
     mmPattern)
    (list (nreverse mmpat) patternAlist partial patvars)))
```

—————

5.2.41 defun evalAndRwriteLispForm

```
[eval p??]
[rwriteLispForm p170]
```

— defun evalAndRwriteLispForm —

```
(defun |evalAndRwriteLispForm| (key form)
  (|eval| form)
  (|rwriteLispForm| key form))
```

5.2.42 defun rwriteLispForm

```
[|libFile| p??]
[|lisplib| p??]
```

— defun rwriteLispForm —

```
(defun |rwriteLispForm| (key form)
  (declare (special |libFile| |lisplib|))
  (when $lisplib
    (|rwrite| key form |libFile|)
    (|LAM,FILEACTQ| key form)))
```

5.2.43 defun mkConstructor

```
[mkConstructor p170]
```

— defun mkConstructor —

```
(defun |mkConstructor| (form)
  (cond
    ((atom form) (list '|devaluate| form))
    ((null (rest form)) (list 'quote (list (first form))))
    (t
     (cons 'list
           (cons (mkq (first form))
                 (loop for x in (rest form) collect (|mkConstructor| x)))))))
```

5.2.44 defun compDefineCategory

```
[compDefineLisplib p171]
[compDefineCategory1 p137]
[$domainShell p??]
```

```

[$lisplibCategory p??]
[$lisplib p??]
[$insideFunctorIfTrue p??]

```

— **defun compDefineCategory** —

```

(defun |compDefineCategory| (df mode env prefix fal)
  (let (|$domainShell| |$lisplibCategory|)
    (declare (special |$domainShell| |$lisplibCategory| $lisplib
                     |$insideFunctorIfTrue|))
    (setq |$domainShell| nil) ; holds the category of the object being compiled
    (setq |$lisplibCategory| nil)
    (if (and (null |$insideFunctorIfTrue|) $lisplib)
        (|compDefineLisplib| df mode env prefix fal ' |compDefineCategory1|)
        (|compDefineCategory1| df mode env prefix fal))))

```

5.2.45 defun compDefineLisplib

```

[sayMSG p??]
[fillerSpaces p??]
[getConstructorAbbreviation p??]
[compileDocumentation p174]
[bright p??]
[finalizeLisplib p176]
[rshut p??]
[lisplibDoRename p174]
[filep p??]
[rpackfile p??]
[unloadOneConstructor p173]
[localdatabase p??]
[getdatabase p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[$compileDocumentation p174]
[$filep p??]
[$spadLibFT p??]
[$algebraOutputStream p??]
[$newConlist p??]
[$lisplibKind p??]
[$lisplib p??]
[$op p??]
[$lisplibParents p??]
[$lisplibPredicates p??]

```

```

[$lisplibCategoriesExtended p??]
[$lisplibForm p??]
[$lisplibKind p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibModemap p??]
[$lisplibModemapAlist p??]
[$lisplibSlot1 p??]
[$lisplibOperationAlist p??]
[$lisplibSuperDomain p??]
[$libFile p??]
[$lisplibVariableAlist p??]
[$lisplibCategory p??]
[$newConlist p??]

```

— **defun compDefineLisplib** —

```

(defun |compDefineLisplib| (df m env prefix fal fn)
  (let ($LISPLIB |$op| |$lisplibAttributes| |$lisplibPredicates|
        |$lisplibCategoriesExtended| |$lisplibForm| |$lisplibKind|
        |$lisplibAbbreviation| |$lisplibParents| |$lisplibAncestors|
        |$lisplibModemap| |$lisplibModemapAlist| |$lisplibSlot1|
        |$lisplibOperationAlist| |$lisplibSuperDomain| |$libFile|
        |$lisplibVariableAlist| |$lisplibCategory| op libname res ok filearg)
    (declare (special $lisplib |$op| |$lisplibAttributes| |$newConlist|
                    |$lisplibPredicates| |$lisplibCategoriesExtended| | |
                    |$lisplibForm| |$lisplibKind| |$algebraOutputStream|
                    |$lisplibAbbreviation| |$lisplibParents| |$spadLibFT|
                    |$lisplibAncestors| |$lisplibModemap| $filep
                    |$lisplibModemapAlist| |$lisplibSlot1|
                    |$lisplibOperationAlist| |$lisplibSuperDomain|
                    |$libFile| |$lisplibVariableAlist|
                    |$lisplibCategory| |$compileDocumentation|))
      (when (eq (car df) 'def) (car df))
      (setq op (caadr df))
      (|sayMSG| (|fillerSpaces| 72 "-"))
      (setq $lisplib t)
      (setq |$op| op)
      (setq |$lisplibAttributes| nil)
      (setq |$lisplibPredicates| nil)
      (setq |$lisplibCategoriesExtended| nil)
      (setq |$lisplibForm| nil)
      (setq |$lisplibKind| nil)
      (setq |$lisplibAbbreviation| nil)
      (setq |$lisplibParents| nil)
      (setq |$lisplibAncestors| nil)
      (setq |$lisplibModemap| nil)
      (setq |$lisplibModemapAlist| nil)

```

```

(setq |$lisplibSlot1| nil)
(setq |$lisplibOperationAlist| nil)
(setq |$lisplibSuperDomain| nil)
(setq |$libFile| nil)
(setq |$lisplibVariableAlist| nil)
(setq |$lisplibCategory| nil)
(setq libname (|getConstructorAbbreviation| op))
(cond
  ((and (boundp '|$compileDocumentation|) |$compileDocumentation|)
    (|compileDocumentation| libname))
  (t
    (|sayMSG| (cons "    initializing " (cons |$spadLibFT|
      (append (|bright| libname) (cons "for" (|bright| op)))))))
    (|initializeLisplib| libname)
    (|sayMSG|
      (cons "    compiling into " (cons |$spadLibFT| (|bright| libname))))
    (setq ok nil)
    (unwind-protect
      (progn
        (setq res (funcall fn df m env prefix fal))
        (|sayMSG| (cons "    finalizing " (cons |$spadLibFT| (|bright| libname))))
        (|finalizeLisplib| libname)
        (setq ok t))
        (rshut |$libFile|))
      (when ok (|lisplibDoRename| libname))
      (setq filearg ($filep libname |$spadLibFT| 'a))
      (rpackfile filearg)
      (fresh-line |$algebraOutputStream|)
      (|sayMSG| (|fillerSpaces| 72 "-"))
      (|unloadOneConstructor| op libname)
      (localdatabase (list (getdatabase op 'abbreviation)) nil)
      (setq |$newConlist| (cons op |$newConlist|))
      (when (eq |$lisplibKind| '|category|)
        (|updateCategoryFrameForCategory| op)
        (|updateCategoryFrameForConstructor| op))
      res))))

```

5.2.46 defun unloadOneConstructor

```

[remprop p??]
[mkAutoLoad p??]

```

— defun unloadOneConstructor —

```

(defun |unloadOneConstructor| (cnam fn)

```

```
(remprop cnam 'loaded)
(setf (symbol-function cnam) (|mkAutoLoad| fn cnam)))
```

5.2.47 defun compileDocumentation

```
[make-input-filename p??]
[rdefiostream p??]
[lisplibWrite p182]
[finalizeDocumentation p466]
[rshut p??]
[rpackfile p??]
[replaceFile p??]
[$fcopy p??]
[$spadLibFT p??]
[$EmptyMode p131]
[$e p??]
```

— defun compileDocumentation —

```
(defun |compileDocumentation| (libName)
  (let (filename stream)
    (declare (special |$e| |$EmptyMode| |$spadLibFT| $fcopy))
    (setq filename (make-input-filename libName |$spadLibFT|))
    ($fcopy filename (cons libname (list 'doclb)))
    (setq stream
      (rdefiostream (cons (list 'file libName 'doclb) (list (cons 'mode 'o))))))
    (|lisplibWrite| "documentation" (|finalizeDocumentation|) stream)
    (rshut stream)
    (rpackfile (list libName 'doclb))
    (replaceFile (list libName |$spadLibFT|) (list libName 'doclb))
    (list '|dummy| |$EmptyMode| |$e|)))
```

5.2.48 defun lisplibDoRename

```
[replaceFile p??]
[$spadLibFT p??]
```

— defun lisplibDoRename —

```
(defun |lisplibDoRename| (libName)
```

```
(declare (special |$spadLibFT|))
(replaceFile (list libName |$spadLibFT| 'a) (list libName 'errorlib 'a)))
```

5.2.49 defun initializeLisplib

```
[erase p??]
[writeLib1 p176]
[adoptions p??]
[pathnameTypeId p??]
[LAM,FILEACTQ p??]
[$erase p??]
[$libFile p??]
[$libFile p??]
[$lisplibForm p??]
[$lisplibModemap p??]
[$lisplibKind p??]
[$lisplibModemapAlist p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibOpAlist p??]
[$lisplibOperationAlist p??]
[$lisplibSuperDomain p??]
[$lisplibVariableAlist p??]
[$lisplibSignatureAlist p??]
[/editfile p??]
[/major-version p??]
[errors p??]
```

— defun initializeLisplib —

```
(defun |initializeLisplib| (libName)
  (declare (special $erase |$libFile| |$lisplibForm|
                    |$lisplibModemap| |$lisplibKind| |$lisplibModemapAlist|
                    |$lisplibAbbreviation| |$lisplibAncestors|
                    |$lisplibOpAlist| |$lisplibOperationAlist|
                    |$lisplibSuperDomain| |$lisplibVariableAlist| errors
                    |$lisplibSignatureAlist| /editfile /major-version errors))
  ($erase libName 'errorlib 'a)
  (setq errors 0)
  (setq |$libFile| (|writeLib1| libname 'errorlib 'a))
  (adoptions 'file |$libFile|)
  (setq |$lisplibForm| nil)
  (setq |$lisplibModemap| nil)
  (setq |$lisplibKind| nil)
```

```

(setq |$lisplibModemapAlist| nil)
(setq |$lisplibAbbreviation| nil)
(setq |$lisplibAncestors| nil)
(setq |$lisplibOpAlist| nil)
(setq |$lisplibOperationAlist| nil)
(setq |$lisplibSuperDomain| nil)
(setq |$lisplibVariableAlist| nil)
(setq |$lisplibSignatureAlist| nil)
(when (eq (|pathnameTypeId| /editfile) 'spad)
  (|LAM,FILEACTQ| 'version (list '/versioncheck /major-version))))

```

5.2.50 defun writeLib1

[rdefiostream p??]

— defun writeLib1 —

```

(defun |writeLib1| (fn ft fm)
  (rdefiostream (cons (list 'file fn ft fm) (list '(mode . output))))

```

5.2.51 defun finalizeLisplib

```

[lisplibWrite p182]
[removeZeroOne p??]
[namestring p??]
[getConstructorOpsAndAtts p178]
[NRTgenInitialAttributeAlist p??]
[mergeSignatureAndLocalVarAlists p182]
[finalizeDocumentation p466]
[profileWrite p??]
[sayMSG p??]
[$lisplibForm p??]
[$libFile p??]
[$lisplibKind p??]
[$lisplibModemap p??]
[$lisplibCategory p??]
[$/editfile p??]
[$lisplibModemapAlist p??]
[$lisplibForm p??]
[$lisplibModemap p??]

```

```

[$FormalMapVariableList p250]
[$lisplibSuperDomain p??]
[$lisplibSignatureAlist p??]
[$lisplibVariableAlist p??]
[$lisplibAttributes p??]
[$lisplibPredicates p??]
[$lisplibAbbreviation p??]
[$lisplibParents p??]
[$lisplibAncestors p??]
[$lisplibSlot1 p??]
[$profileCompiler p??]
[$spadLibFT p??]
[$lisplibCategory p??]
[$pairlis p??]
[$NRTslot1PredicateList p??]

```

— defun finalizeLisplib —

```

(defun |finalizeLisplib| (libName)
  (let (|$pairlis| |$NRTslot1PredicateList| kind opsAndAtts)
    (declare (special |$pairlis| |$NRTslot1PredicateList| |$spadLibFT|
                      |$lisplibForm| |$profileCompiler| |$libFile|
                      |$lisplibSlot1| |$lisplibAncestors| |$lisplibParents|
                      |$lisplibAbbreviation| |$lisplibPredicates|
                      |$lisplibAttributes| |$lisplibVariableAlist|
                      |$lisplibSignatureAlist| |$lisplibSuperDomain|
                      |$FormalMapVariableList| |$lisplibModemap|
                      |$lisplibModemapAlist| /editfile |$lisplibCategory|
                      |$lisplibKind| errors))
      (|lisplibWrite| "constructorForm"
        (|removeZeroOne| |$lisplibForm|) |$libFile|)
      (|lisplibWrite| "constructorKind"
        (setq kind (|removeZeroOne| |$lisplibKind|)) |$libFile|)
      (|lisplibWrite| "constructorModemap"
        (|removeZeroOne| |$lisplibModemap|) |$libFile|)
      (setq |$lisplibCategory| (or |$lisplibCategory| (cadar |$lisplibModemap|)))
      (|lisplibWrite| "constructorCategory" |$lisplibCategory| |$libFile|)
      (|lisplibWrite| "sourceFile" (|namestring| /editfile) |$libFile|)
      (|lisplibWrite| "modemaps"
        (|removeZeroOne| |$lisplibModemapAlist|) |$libFile|)
      (setq opsAndAtts
        (|getConstructorOpsAndAtts| |$lisplibForm| kind |$lisplibModemap|))
      (|lisplibWrite| "operationAlist"
        (|removeZeroOne| (car opsAndAtts)) |$libFile|)
      (when (eq kind '|category|)
        (setq |$pairlis|
          (loop for a in (rest |$lisplibForm|)
                for v in |$FormalMapVariableList|

```

```

      collect (cons a v)))
    (setq |$NRTslot1PredicateList| nil)
    (|NRTgenInitialAttributeAlist| (cdr opsAndAtts)))
  (|lisplibWrite| "superDomain"
    (|removeZeroOne| |$lisplibSuperDomain|) |$libFile|)
  (|lisplibWrite| "signaturesAndLocals"
    (|removeZeroOne|
      (|mergeSignatureAndLocalVarAlists| |$lisplibSignatureAlist|
                                           |$lisplibVariableAlist|))
    |$libFile|)
  (|lisplibWrite| "attributes"
    (|removeZeroOne| |$lisplibAttributes|) |$libFile|)
  (|lisplibWrite| "predicates"
    (|removeZeroOne| |$lisplibPredicates|) |$libFile|)
  (|lisplibWrite| "abbreviation" |$lisplibAbbreviation| |$libFile|)
  (|lisplibWrite| "parents" (|removeZeroOne| |$lisplibParents|) |$libFile|)
  (|lisplibWrite| "ancestors" (|removeZeroOne| |$lisplibAncestors|) |$libFile|)
  (|lisplibWrite| "documentation" (|finalizeDocumentation|) |$libFile|)
  (|lisplibWrite| "slot1Info" (|removeZeroOne| |$lisplibSlot1|) |$libFile|)
  (when |$profileCompiler| (|profileWrite|))
  (when (and |$lisplibForm| (null (cdr |$lisplibForm|))))
    (setf (get (car |$lisplibForm|) 'niladic) t))
  (unless (eq errors 0)
    (|sayMSG| (list "      Errors in processing " kind " " libName ":")
      (|sayMSG| (list "      not replacing " |$spadLibFT| " for" libName)))))

```

5.2.52 defun getConstructorOpsAndAtts

[getCategoryOpsAndAtts p178]
 [getFunctorOpsAndAtts p181]

— defun getConstructorOpsAndAtts —

```

(defun |getConstructorOpsAndAtts| (form kind modemap)
  (if (eq kind '|category|)
    (|getCategoryOpsAndAtts| form)
    (|getFunctorOpsAndAtts| form modemap)))

```

5.2.53 defun getCategoryOpsAndAtts

[transformOperationAlist p179]
 [getSlotFromCategoryForm p179]

[getSlotFromCategoryForm p179]

— defun getCategoryOpsAndAtts —

```
(defun |getCategoryOpsAndAtts| (catForm)
  (cons (|transformOperationAlist| (|getSlotFromCategoryForm| catForm 1))
        (|getSlotFromCategoryForm| catForm 2)))
```

—————

5.2.54 defun getSlotFromCategoryForm

[eval p??]
 [take p??]
 [systemErrorHere p??]
 [\$FormalMapVariableList p250]

— defun getSlotFromCategoryForm —

```
(defun |getSlotFromCategoryForm| (opargs index)
  (let (op arg1 u)
    (declare (special |$FormalMapVariableList|))
    (setq op (first opargs))
    (setq arg1 (rest opargs))
    (setq u
      (|eval| (cons op (mapcar 'mkq (take (|#| arg1) |$FormalMapVariableList|)))))
    (if (null (vecp u))
        (|systemErrorHere| "getSlotFromCategoryForm")
        (elt u index))))
```

—————

5.2.55 defun transformOperationAlist

This transforms the operationAlist which is written out onto LISPLIBs. The original form of this list is a list of items of the form:

```
((<op> <signature>) (<condition> (ELT $ n)))
```

The new form is an op-Alist which has entries

```
(<op> . signature-Alist)
```

where signature-Alist has entries

```
(<signature> . item)
```

where item has form

```
(<slotNumber> <condition> <kind>)
```

```
where <kind> =
  NIL => function
  CONST => constant ... and others
```

```
[member p??]
[keyedSystemError p??]
[assoc p??]
[lassq p??]
[insertAlist p??]
[$functionLocations p??]
```

— **defun transformOperationAlist** —

```
(defun |transformOperationAlist| (operationAlist)
  (let (op sig condition implementation eltEtc impOp kind u n signatureItem
        itemList newAlist)
    (declare (special |$functionLocations|))
    (setq newAlist nil)
    (dolist (item operationAlist)
      (setq op (caar item))
      (setq sig (cadar item))
      (setq condition (cadr item))
      (setq implementation (caddr item))
      (setq kind
        (cond
          ((and (consp implementation) (consp (qrest implementation))
                (consp (qcddr implementation))
                (eq (qcdddr implementation) nil)
                (progn (setq n (qthird implementation)) t)
                (|member| (setq eltEtc (qfirst implementation)) '(const elt)))
            eltEtc)
          ((consp implementation)
            (setq impOp (qfirst implementation))
            (cond
              ((eq impOp 'xlam) implementation)
              ((|member| impOp '(const |Subsumed|)) impOp)
              (t (|keyedSystemError| 's2il0025 (list impOp))))))
            ((eq implementation '|mkRecord|) '|mkRecord|)
            (t (|keyedSystemError| 's2il0025 (list implementation))))))
      (when (setq u (|assoc| (list op sig) |$functionLocations|))
        (setq n (cons n (cdr u))))
      (setq signatureItem
```

```

      (if (eq kind 'elt)
        (if (eq condition t)
          (list sig n)
          (list sig n condition))
        (list sig n condition kind)))
      (setq itemList (cons signatureItem (lassq op newAlist)))
      (setq newAlist (|insertAlist| op itemList newAlist))
      newAlist))

```

5.2.56 defun getFunctorOpsAndAtts

[transformOperationAlist p179]
 [getSlotFromFunctor p181]

— defun getFunctorOpsAndAtts —

```

(defun |getFunctorOpsAndAtts| (form modemap)
  (cons (|transformOperationAlist| (|getSlotFromFunctor| form 1 modemap))
        (|getSlotFromFunctor| form 2 modemap)))

```

5.2.57 defun getSlotFromFunctor

[compMakeCategoryObject p182]
 [systemErrorHere p??]
 [\$e p??]
 [\$lisplibOperationAlist p??]

— defun getSlotFromFunctor —

```

(defun |getSlotFromFunctor| (arg1 slot arg2)
  (declare (ignore arg1))
  (let (tt)
    (declare (special |$e| |$lisplibOperationAlist|))
    (cond
      ((eq slot 1) |$lisplibOperationAlist|)
      (t
       (setq tt (or (|compMakeCategoryObject| (cadar arg2) |$e|)
                    (|systemErrorHere| "getSlotFromFunctor"))))
      (elt (car tt) slot))))

```

5.2.58 defun compMakeCategoryObject

```
[isCategoryForm p??]
[mkEvalableCategoryForm p140]
[$e p??]
[$Category p??]
```

— defun compMakeCategoryObject —

```
(defun |compMakeCategoryObject| (c |$e|)
  (declare (special |$e|))
  (let (u)
    (declare (special |$Category|))
    (cond
      ((null (|isCategoryForm| c |$e|)) nil)
      ((setq u (|mkEvalableCategoryForm| c)) (list (|eval| u) |$Category| |$e|))
      (t nil))))
```

—————

5.2.59 defun mergeSignatureAndLocalVarAlists

```
[lassoc p??]
```

— defun mergeSignatureAndLocalVarAlists —

```
(defun |mergeSignatureAndLocalVarAlists| (signatureAlist localVarAlist)
  (loop for item in signatureAlist
    collect
      (cons (first item)
            (cons (rest item)
                  (lassoc (first item) localVarAlist)))))
```

—————

5.2.60 defun lisplibWrite

```
[rwrite128 p??]
[$lisplib p??]
```

— defun lisplibWrite —

```
(defun |lisplibWrite| (prop val filename)
  (declare (special $lisplib))
```

```
(when $lisplib (|rwrite| prop val filename)))
```

5.2.61 defun compDefineFunctor

```
[compDefineLisplib p171]
[compDefineFunctor1 p183]
[$domainShell p??]
[$profileCompiler p??]
[$lisplib p??]
[$profileAlist p??]
```

— defun compDefineFunctor —

```
(defun |compDefineFunctor| (df mode env prefix fal)
  (let (|$domainShell| |$profileCompiler| |$profileAlist|)
    (declare (special |$domainShell| |$profileCompiler| $lisplib |$profileAlist|))
    (setq |$domainShell| nil)
    (setq |$profileCompiler| t)
    (setq |$profileAlist| nil)
    (if $lisplib
      (|compDefineLisplib| df mode env prefix fal '|compDefineFunctor1|)
      (|compDefineFunctor1| df mode env prefix fal))))
```

5.2.62 defun compDefineFunctor1

```
[isCategoryPackageName p190]
[getArgumentModeOrMoan p156]
[getModemap p239]
[giveFormalParametersValues p135]
[compMakeCategoryObject p182]
[sayBrightly p??]
[pp p??]
[strconc p??]
[pname p??]
[disallowNilAttribute p195]
[remdup p??]
[NRTgenInitialAttributeAlist p??]
[NRTgetLocalIndex p192]
[compMakeDeclaration p593]
[qcar p??]
```

[qcdr p??]
[augModemapsFromCategoryRep p251]
[augModemapsFromCategory p245]
[sublis p??]
[maxindex p??]
[makeFunctorArgumentParameters p198]
[compFunctorBody p195]
[reportOnFunctorCompilation p197]
[compile p145]
[augmentLisplibModemapsFromFunctor p193]
[reportOnFunctorCompilation p197]
[getParentsFor p??]
[computeAncestorsOf p??]
[constructor? p??]
[nequal p??]
[NRTmakeSlot1Info p??]
[isCategoryPackageName p190]
[lisplibWrite p182]
[mkq p??]
[getdatabase p??]
[NRTgetLookupFunction p191]
[simpBool p??]
[removeZeroOne p??]
[evalAndRwriteLispForm p169]
[\$lisplib p??]
[\$top-level p??]
[\$bootStrapMode p??]
[\$CategoryFrame p??]
[\$CheckVectorList p??]
[\$FormalMapVariableList p250]
[\$LocalDomainAlist p??]
[\$NRTaddForm p??]
[\$NRTaddList p??]
[\$NRTattributeAlist p??]
[\$NRTbase p??]
[\$NRTdeltaLength p??]
[\$NRTdeltaListComp p??]
[\$NRTdeltaList p??]
[\$NRTdomainFormList p??]
[\$NRTloadTimeAlist p??]
[\$NRTslot1Info p??]
[\$NRTslot1PredicateList p??]
[\$Representation p??]
[\$addForm p??]
[\$attributesName p??]
[\$byteAddress p??]

```

[$byteVec p??]
[$compileOnlyCertainItems p??]
[$condAlist p??]
[$domainShell p??]
[$form p??]
[$functionLocations p??]
[$functionStats p??]
[$functorForm p??]
[$functorLocalParameters p??]
[$functorStats p??]
[$functorSpecialCases p??]
[$functorTarget p??]
[$functorsUsed p??]
[$genFVar p??]
[$genSDVar p??]
[$getDomainCode p??]
[$goGetList p??]
[$insideCategoryPackageIfTrue p??]
[$insideFunctorIfTrue p??]
[$isOpPackageName p??]
[$libFile p??]
[$lisplibAbbreviation p??]
[$lisplibAncestors p??]
[$lisplibCategoriesExtended p??]
[$lisplibCategory p??]
[$lisplibForm p??]
[$lisplibKind p??]
[$lisplibMissingFunctions p??]
[$lisplibModemap p??]
[$lisplibOperationAlist p??]
[$lisplibParents p??]
[$lisplibSlot1 p??]
[$lookupFunction p??]
[$myFunctorBody p??]
[$mutableDomain p??]
[$mutableDomains p??]
[$op p??]
[$pairlis p??]
[$QuickCode p??]
[$setelt p??]
[$signature p??]
[$template p??]
[$uncondAlist p??]
[$viewNames p??]
[$lisplibFunctionLocations p??]

```

— defun compDefineFunctor1 —

```

(defun |compDefineFunctor1| (df mode |$e| |$prefix| |$formalArgList|)
  (declare (special |$e| |$prefix| |$formalArgList|))
  (labels (
    (FindRep (cb)
      (loop while cb do
        (when (atom cb) (return nil))
        (when (and (consp cb) (consp (qfirst cb)) (eq (qcaar cb) 'let)
          (consp (qcddar cb)) (eq (qcadar cb) '|Rep|)
          (consp (qcddar cb))))
        (return (caddar cb)))
      (pop cb))))
  (let (|$addForm| |$viewNames| |$functionStats| |$functorStats|
        |$form| |$op| |$signature| |$functorTarget|
        |$Representation| |$LocalDomainAlist| |$functorForm|
        |$functorLocalParameters| |$CheckVectorList|
        |$getDomainCode| |$insideFunctorIfTrue| |$functorsUsed|
        |$setelt| $TOP_LEVEL |$genFVar| |$genSDVar|
        |$mutableDomain| |$attributesName| |$goGetList|
        |$condAlist| |$uncondAlist| |$NRTslot1PredicateList|
        |$NRTattributeAlist| |$NRTslot1Info| |$NRTbase|
        |$NRTaddForm| |$NRTdeltaList| |$NRTdeltaListComp|
        |$NRTaddList| |$NRTdeltaLength| |$NRTloadTimeAlist|
        |$NRTdomainFormList| |$template| |$functionLocations|
        |$isOpPackageName| |$lookupFunction| |$byteAddress|
        |$byteVec| form signature body originale argl signaturep target ds
        attributeList parSignature parForm
        argPars opp rettype tt bodyp lamOrSlam fun
        operationAlist modemap libFn tmp1)
    (declare (special $lisplib $top_level $bootstrapMode |$CategoryFrame|
                      |$CheckVectorList| |$FormalMapVariableList| | | | |
                      |$LocalDomainAlist| |$NRTaddForm| |$NRTaddList|
                      |$NRTattributeAlist| |$NRTbase| |$NRTdeltaLength|
                      |$NRTdeltaListComp| |$NRTdeltaList| |$NRTdomainFormList|
                      |$NRTloadTimeAlist| |$NRTslot1Info| |$NRTslot1PredicateList|
                      |$Representation| |$addForm| |$attributesName|
                      |$byteAddress| |$byteVec| |$compileOnlyCertainItems|
                      |$condAlist| |$domainShell| |$form| |$functionLocations|
                      |$functionStats| |$functorForm| |$functorLocalParameters|
                      |$functorStats| |$functorSpecialCases| |$functorTarget|
                      |$functorsUsed| |$genFVar| |$genSDVar| |$getDomainCode|
                      |$goGetList| |$insideCategoryPackageIfTrue|
                      |$insideFunctorIfTrue| |$isOpPackageName| |$libFile|
                      |$lisplibAbbreviation| |$lisplibAncestors|
                      |$lisplibCategoriesExtended| |$lisplibCategory|
                      |$lisplibForm| |$lisplibKind| |$lisplibMissingFunctions|
                      |$lisplibModemap| |$lisplibOperationAlist| |$lisplibParents|
                      |$lisplibSlot1| |$lookupFunction| |$myFunctorBody|
                      |$mutableDomain| |$mutableDomains| |$op| |$pairlis|

```

```

      |$QuickCode| |$setelt| |$signature| |$template|
      |$uncondAlist| |$viewNames| |$lisplibFunctionLocations|))
(setq form (second df))
(setq signature (third df))
(setq |$functorSpecialCases| (fourth df))
(setq body (fifth df))
(setq |$addForm| nil)
(setq |$viewNames| nil)
(setq |$functionStats| (list 0 0))
(setq |$functorStats| (list 0 0))
(setq |$form| nil)
(setq |$op| nil)
(setq |$signature| nil)
(setq |$functorTarget| nil)
(setq |$Representation| nil)
(setq |$LocalDomainAlist| nil)
(setq |$functorForm| nil)
(setq |$functorLocalParameters| nil)
(setq |$myFunctorBody| body)
(setq |$CheckVectorList| nil)
(setq |$getDomainCode| nil)
(setq |$insideFunctorIfTrue| t)
(setq |$functorsUsed| nil)
(setq |$setelt| (if |$QuickCode| 'qsetrefv 'setelt))
(setq $top_level nil)
(setq |$genFVar| 0)
(setq |$genSDVar| 0)
(setq originale |$e|)
(setq |$op| (first form))
(setq argl (rest form))
(setq |$formalArgList| (append argl |$formalArgList|))
(setq |$pairlis|
  (loop for a in argl for v in |$FormalMapVariableList|
    collect (cons a v)))
(setq |$mutableDomain|
  (OR (|isCategoryPackageName| |$op|)
    (COND
      ((boundp '|$mutableDomains|)
       (member |$op| |$mutableDomains|))
      ('T NIL))))
(setq signaturep
  (cons (car signature)
    (loop for a in argl collect (|getArgumentModeOrMoan| a form |$e|))))
(setq |$form| (cons |$op| argl))
(setq |$functorForm| |$form|)
(unless (car signaturep)
  (setq signaturep (cdar (|getModemap| |$form| |$e|))))
(setq target (first signaturep))
(setq |$functorTarget| target)
(setq |$e| (|giveFormalParametersValues| argl |$e|))

```

```

(setq tmp1 (|compMakeCategoryObject| target |$e|))
(if tmp1
  (progn
    (setq ds (first tmp1))
    (setq |$e| (third tmp1))
    (setq |$domainShell| (copy-seq ds))
    (setq |$attributesName| (intern (strconc (pname |$op|) ";attributes")))
    (setq attributeList (|disallowNilAttribute| (elt ds 2)))
    (setq |$goGetList| nil)
    (setq |$condAlist| nil)
    (setq |$uncondAlist| nil)
    (setq |$NRTslot1PredicateList|
      (remdup (loop for x in attributeList collect (second x))))
    (setq |$NRTattributeAlist| (|NRTgenInitialAttributeAlist| attributeList))
    (setq |$NRTslot1Info| nil)
    (setq |$NRTbase| 6)
    (setq |$NRTaddForm| nil)
    (setq |$NRTdeltaList| nil)
    (setq |$NRTdeltaListComp| nil)
    (setq |$NRTaddList| nil)
    (setq |$NRTdeltaLength| 0)
    (setq |$NRTloadTimeAlist| nil)
    (setq |$NRTdomainFormList| nil)
    (setq |$template| nil)
    (setq |$functionLocations| nil)
    (loop for x in arg1 do (|NRTgetLocalIndex| x))
    (setq |$e|
      (third (|compMakeDeclaration| (list '(:| '$ target) mode |$e|)))
      (unless |$insideCategoryPackageIfTrue|
        (if
          (and (consp body) (eq (qfirst body) '|add|)
            (consp (qrest body))
            (consp (qsecond body))
            (consp (qcddr body))
            (eq (qcdddr body) nil)
            (consp (qthird body))
            (eq (qcaaddr body) '|capsule|)
            (member (qcaadr body) '(|List| |Vector|))
            (equal (FindRep (qcdaddr body)) (second body)))
          (setq |$e| (|augModemapsFromCategoryRep| '$
            (second body) (cdaddr body) target |$e|))
          (setq |$e| (|augModemapsFromCategory| '$ '$ target |$e|))))
    (setq |$signature| signaturep)
    (setq operationAlist (sublis |$pairlis| (elt |$domainShell| 1)))
    (setq parSignature (sublis |$pairlis| signaturep))
    (setq parForm (sublis |$pairlis| form))
    (setq argPars (|makeFunctorArgumentParameters| arg1
      (cdr signaturep) (car signaturep)))
    (setq |$functorLocalParameters| arg1)
    (setq opp |$op|)

```

```

(setq rettype (CAR signaturep))
(setq tt (|compFunctorBody| body rettype |$e| parForm))
(cond
  (|$compileOnlyCertainItems|
    (|reportOnFunctorCompilation|)
    (list nil (cons '|Mapping| signaturep) originale))
  (t
    (setq bodyp (first tt))
    (setq lamOrSlam (if |$mutableDomain| 'lam 'spadslam))
    (setq fun
      (|compile| (sublis |$pairlis| (list opp (list lamOrSlam argl bodyp)))))
    (setq operationAlist (sublis |$pairlis| |$lisplibOperationAlist|))
    (cond
      ($lisplib
        (|augmentLisplibModemapsFromFunctor| parForm
          operationAlist parSignature)))
    (|reportOnFunctorCompilation|)
    (cond
      ($lisplib
        (setq modemap (list (cons parForm parSignature) (list t opp)))
        (setq |$lisplibModemap| modemap)
        (setq |$lisplibCategory| (cadar modemap))
        (setq |$lisplibParents|
          (|getParentsFor| |$op| |$FormalMapVariableList| |$lisplibCategory|))
        (setq |$lisplibAncestors| (|computeAncestorsOf| |$form| NIL))
        (setq |$lisplibAbbreviation| (|constructor?| |$op|)))
      (setq |$insideFunctorIfTrue| NIL)
      (cond
        ($lisplib
          (setq |$lisplibKind|
            (if (and (consp |$functorTarget|)
              (eq (qfirst |$functorTarget|) 'category)
              (consp (qrest |$functorTarget|))
              (nequal (qsecond |$functorTarget|) '|domain|))
              '|package|
              '|domain|))
            (setq |$lisplibForm| form)
            (cond
              ((null |$bootStrapMode|)
                (setq |$NRTslot1Info| (|NRTmakeSlot1Info|))
                (setq |$isOpPackageName| (|isCategoryPackageName| |$op|))
                (when |$isOpPackageName|
                  (|lisplibWrite| "slot1DataBase"
                    (list '|updateSlot1DataBase| (mkq |$NRTslot1Info|)
                      |$libFile|))
                  (setq |$lisplibFunctionLocations|
                    (sublis |$pairlis| |$functionLocations|))
                  (setq |$lisplibCategoriesExtended|
                    (sublis |$pairlis| |$lisplibCategoriesExtended|))
                  (setq libFn (getdatabase opp 'abbreviation))

```

```

(setq |$lookupFunction|
  (|NRTgetLookupFunction| |$functorForm|
    (cadar |$lisplibModemap|) |$NRtaddForm|))
(setq |$byteAddress| 0)
(setq |$byteVec| NIL)
(setq |$NRTslot1PredicateList|
  (loop for x in |$NRTslot1PredicateList|
    collect (|simpBool| x)))
(|rewriteLispForm| 'loadTimeStuff|
  '(setf (get , (mkq |$op|) '|infovec|) , (|getInfovecCode|))))
(setq |$lisplibSlot1| |$NRTslot1Info|)
(setq |$lisplibOperationAlist| operationAlist)
(setq |$lisplibMissingFunctions| |$CheckVectorList|))
(|lisplibWrite| "compilerInfo"
  (|removeZeroOne|
    (list 'setq '|$CategoryFrame|
      (list '|put| (list 'quote opp) '|isFunctor|
        (list 'quote operationAlist)
        (list '|addModemap|
          (list 'quote opp)
          (list 'quote parForm)
          (list 'quote parSignature)
          t
          (list 'quote opp)
          (list '|put| (list 'quote opp) '|model|
            (list 'quote (cons '|Mapping| parSignature))
            '|$CategoryFrame|))))))
      |$libFile|)
    (unless arg1
      (|evalAndRwriteLispForm| 'niladic
        '(setf (get ',opp 'niladic) t)))
      (list fun (cons '|Mapping| signaturep) originale))))
  (progn
    (|sayBrightly| "  cannot produce category object:")
    (|pp| target)
    nil))))

```

5.2.63 defun isCategoryPackageName

```

[pname p??]
[maxindex p??]
[char p??]

```

— defun isCategoryPackageName —

```
(defun |isCategoryPackageName| (nam)
```

```
(let (p)
  (setq p (pname (lop0f| nam)))
  (equal (elt p (maxindex p)) (|char| '&))))
```

5.2.64 defun NRTgetLookupFunction

Compute the lookup function (complete or incomplete) [sublis p??]

```
[NRTextendsCategory1 p??]
[getExportCategory p??]
[sayBrightly p??]
[sayBrightlyNT p??]
[bright p??]
[form2String p??]
[$why p??]
[$why p??]
[$pairlis p??]
```

— defun NRTgetLookupFunction —

```
(defun |NRTgetLookupFunction| (domform exCategory addForm)
  (let (|$why| extends u msg v)
    (declare (special |$why| |$pairlis|))
    (setq domform (sublis |$pairlis| domform))
    (setq addForm (sublis |$pairlis| addForm))
    (setq |$why| nil)
    (cond
      ((atom addForm) '|lookupComplete|)
      (t
       (setq extends
         (|NRTextendsCategory1| domform exCategory (|getExportCategory| addForm)))
       (cond
         ((null extends)
          (setq u (car |$why|))
          (setq msg (cadr |$why|))
          (setq v (cddr |$why|))
          (|sayBrightly|
           "-----non extending category-----")
          (|sayBrightlyNT|
           (cons ".."
             (append (|bright| (|form2String| domform)) (list '|of cat|))))
          (print u)
          (|sayBrightlyNT| (|bright| msg))
          (if v (print (car v)) (terpri))))
       '|lookupIncomplete|
```

```
'|lookupComplete|))))))
```

5.2.65 defun NRTgetLocalIndex

```
[NRTassocIndex p336]
[NRTaddInner p??]
[compOrCroak p556]
[rplaca p??]
[$NRTaddForm p??]
[$formalArgList p??]
[$NRTdeltaList p??]
[$NRTdeltaListComp p??]
[$NRTdeltaLength p??]
[$NRTbase p??]
[$EmptyMode p131]
[$e p??]
```

— defun NRTgetLocalIndex —

```
(defun |NRTgetLocalIndex| (item)
  (let (k value saveNRTdeltaListComp saveIndex compEntry)
    (declare (special |$e| |$EmptyMode| |$NRTdeltaLength| |$NRTbase|
                      |$NRTdeltaListComp| |$NRTdeltaList| |$formalArgList|
                      |$NRTaddForm|))
    (cond
      ((setq k (|NRTassocIndex| item)) k)
      ((equal item |$NRTaddForm|) 5)
      ((eq item '$) 0)
      ((eq item '$$) 2)
      (t
       (when (member item |$formalArgList|) (setq value item))
       (cond
         ((and (atom item) (null (member item '($ $$))) (null value))
          (setq |$NRTdeltaList|
                (cons (cons '|domain| (cons (|NRTaddInner| item) value))
                      |$NRTdeltaList|))
          (setq |$NRTdeltaListComp| (cons item |$NRTdeltaListComp|))
          (setq |$NRTdeltaLength| (1+ |$NRTdeltaLength|))
          (1- (+ |$NRTbase| |$NRTdeltaLength|)))
         (t
          (setq |$NRTdeltaList|
                (cons (cons '|domain| (cons (|NRTaddInner| item) value))
                      |$NRTdeltaList|))
          (setq saveNRTdeltaListComp
                (setq |$NRTdeltaListComp| (cons nil |$NRTdeltaListComp|))))
```

```

(setq saveIndex (+ |$NRTbase| |$NRTdeltaLength|))
(setq |$NRTdeltaLength| (1+ |$NRTdeltaLength|))
(setq compEntry (car (|compOrCroak| item |$EmptyMode| |$e|)))
(rplaca saveNRTdeltaListComp compEntry)
saveIndex))))))

```

5.2.66 defun augmentLisplibModemapsFromFunctor

```

[formal2Pattern p194]
[mkAlistOfExplicitCategoryOps p158]
[allLASSOCs p194]
[member p??]
[msubst p??]
[mkDatabasePred p195]
[mkpf p??]
[listOfPatternIds p??]
[interactiveModemapForm p160]
[$lisplibModemapAlist p??]
[$PatternVariableList p??]
[$e p??]
[$lisplibModemapAlist p??]
[$e p??]

```

— defun augmentLisplibModemapsFromFunctor —

```

(defun |augmentLisplibModemapsFromFunctor| (form opAlist signature)
  (let (argl nonCategorySigAlist op pred sel predList sig predp z skip modemap)
    (declare (special |$lisplibModemapAlist| |$PatternVariableList| |$e|))
    (setq form (|formal2Pattern| form))
    (setq argl (cdr form))
    (setq opAlist (|formal2Pattern| opAlist))
    (setq signature (|formal2Pattern| signature))
    ; We are going to be EVALing categories containing these pattern variables
    (loop for u in form for v in signature
      do (when (member u |$PatternVariableList|)
          (setq |$e| (|put| u '|mode| v |$e|))))
    (when
      (setq nonCategorySigAlist (|mkAlistOfExplicitCategoryOps| (CAR signature)))
      (loop for entry in opAlist
        do
          (setq op (caar entry))
          (setq sig (cadar entry))
          (setq pred (cadr entry))
          (setq sel (caddr entry))

```

```

(when
  (let (result)
    (loop for catSig in (|allLASSOCs| op nonCategorySigAlist)
      do (setq result (or result (|member| sig catSig))))
    result)
  (setq skip (when (and argl (contained '$ (cdr sig))) 'skip))
  (setq sel (msubst form '$ sel))
  (setq predList
    (loop for a in argl for m in (rest signature)
      when (|member| a |$PatternVariableList|)
      collect (list a m)))
  (setq sig (msubst form '$ sig))
  (setq predp
    (mkpf
      (cons pred (loop for y in predList collect (|mkDatabasePred| y)))
      'and))
  (setq z (|listOfPatternIds| predList))
  (when (some #'(lambda (u) (null (member u z))) argl)
    (|sayMSG| (list "cannot handle modemap for " op "by pattern match"))
    (setq skip 'skip))
  (setq modemap (list (cons form sig) (cons predp (cons sel skip))))
  (setq |$lisplibModemapAlist|
    (cons
      (cons op (|interactiveModemapForm| modemap))
      |$lisplibModemapAlist|))))))

```

5.2.67 defun allLASSOCs

— defun allLASSOCs —

```

(defun |allLASSOCs| (op alist)
  (loop for value in alist
    when (equal (car value) op)
    collect value))

```

5.2.68 defun formal2Pattern

```

[sublis p??]
[pairList p??]
[$PatternVariableList p??]

```

— defun formal2Pattern —

```
(defun |formal2Pattern| (x)
  (declare (special |$PatternVariableList|))
  (sublis (|pairList| |$FormalMapVariableList| (cdr |$PatternVariableList|)) x))
```

—————

5.2.69 defun mkDatabasePred

```
[isCategoryForm p??]
[$e p??]
```

— defun mkDatabasePred —

```
(defun |mkDatabasePred| (arg)
  (let (a z)
    (declare (special |$e|))
    (setq a (car arg))
    (setq z (cadr arg))
    (if (|isCategoryForm| z |$e|)
        (list '|ofCategory| a z)
        (list '|ofType| a z))))
```

—————

5.2.70 defun disallowNilAttribute

— defun disallowNilAttribute —

```
(defun |disallowNilAttribute| (x)
  (loop for y in x when (and (car y) (nequal (car y) '|nil|))
    collect y))
```

—————

5.2.71 defun compFunctorBody

```
[bootStrapError p196]
[compOrCroak p556]
[/editfile p??]
```

```

[$NRTaddForm p??]
[$functorForm p??]
[$bootStrapMode p??]

```

— **defun compFunctorBody** —

```

(defun |compFunctorBody| (form mode env parForm)
  (declare (ignore parForm))
  (let (tt)
    (declare (special |$NRTaddForm| |$functorForm| |$bootStrapMode| /editfile))
    (if |$bootStrapMode|
      (list (|bootStrapError| |$functorForm| /editfile) mode env)
      (progn
        (setq tt (|compOrCroak| form mode env))
        (if (and (consp form) (member (qfirst form) '(|add| capsule)))
            tt
            (progn
              (setq |$NRTaddForm|
                (if (and (consp form) (eq (qfirst form) '|SubDomain|)
                    (consp (qrest form)) (consp (qcddr form))
                    (eq (qcdddr form) nil))
                  (qsecond form)
                  form))
              tt))))))

```

—————

5.2.72 defun bootStrapError

```

[mkq p??]
[namestring p??]
[mkDomainConstructor p??]

```

— **defun bootStrapError** —

```

(defun |bootStrapError| (functorForm sourceFile)
  (list 'cond
    (list '|$bootStrapMode|
      (list 'vector (|mkDomainConstructor| functorForm) nil nil nil nil nil))
    (list ''t
      (list '|systemError|
        (list 'list ''|%b| (MKQ (CAR functorForm)) ''|%d| "from" ''|%b|
          (mkq (|namestring| sourceFile)) ''|%d| "needs to be compiled")))))

```

—————

5.2.73 defun reportOnFunctorCompilation

```
[displayMissingFunctions p197]
[sayBrightly p??]
[displaySemanticErrors p??]
[displayWarnings p??]
[addStats p??]
[normalizeStatAndStringify p??]
[$op p??]
[$functorStats p??]
[$functionStats p??]
[$warningStack p??]
[$semanticErrorStack p??]
```

— **defun reportOnFunctorCompilation** —

```
(defun |reportOnFunctorCompilation| ()
  (declare (special |$op| |$functorStats| |$functionStats|
                    |$warningStack| |$semanticErrorStack|))
  (|displayMissingFunctions|)
  (when |$semanticErrorStack| (|sayBrightly| " "))
  (|displaySemanticErrors|)
  (when |$warningStack| (|sayBrightly| " "))
  (|displayWarnings|)
  (setq |$functorStats| (|addStats| |$functorStats| |$functionStats|))
  (|sayBrightly|
   (cons '|%1|
         (append (|bright| " Cumulative Statistics for Constructor"
                          (list |$op|))))))
  (|sayBrightly|
   (cons " Time:"
         (append (|bright| (|normalizeStatAndStringify| (second |$functorStats|))
                          (list "seconds")))))
  (|sayBrightly| " ")
  '|done|)
```

5.2.74 defun displayMissingFunctions

```
[member p??]
[getmode p??]
[sayBrightly p??]
[bright p??]
[formatUnabbreviatedSig p??]
[$env p??]
```

```
[$formalArgList p??]
[$CheckVectorList p??]
```

— **defun displayMissingFunctions** —

```
(defun |displayMissingFunctions| ()
  (let (i loc exp)
    (declare (special |$env| |$formalArgList| |$CheckVectorList|))
    (unless |$CheckVectorList|
      (setq loc nil)
      (setq exp nil)
      (loop for cvl in |$CheckVectorList| do
        (unless (cdr cvl)
          (if (and (null (|member| (caar cvl) |$formalArgList|))
                  (consp (|getmode| (caar cvl) |$env|))
                  (eq (qfirst (|getmode| (caar cvl) |$env|)) '|Mapping|))
              (push (list (caar cvl) (cadar cvl)) loc)
              (push (list (caar cvl) (cadar cvl)) exp))))
      (when loc
        (|sayBrightly| (cons '|%l| (|bright| " Missing Local Functions:"))))
        (setq i 0)
        (loop for item in loc do
          (|sayBrightly|
            (cons "          [" (cons (incf i) (cons "]"
              (append (|bright| (first item))
                (cons '|: | (|formatUnabbreviatedSig| (second item))))))))))
        (when exp
          (|sayBrightly| (cons '|%l| (|bright| " Missing Exported Functions:"))))
          (setq i 0)
          (loop for item in exp do
            (|sayBrightly|
              (cons "          [" (cons (incf i) (cons "]"
                (append (|bright| (first item))
                  (cons '|: | (|formatUnabbreviatedSig| (second item)))))))))))))
```

—————

5.2.75 defun makeFunctorArgumentParameters

```
[assq p??]
[msubst p??]
[isCategoryForm p??]
[qcar p??]
[qcdr p??]
[genDomainViewList0 p200]
[union p??]
[$ConditionalOperators p??]
```

```
[$alternateViewList p??]
[$forceAdd p??]
```

— defun makeFunctorArgumentParameters —

```
(defun |makeFunctorArgumentParameters| (argl sigl target)
  (labels (
    (augmentSig (s ss)
      (let (u)
        (declare (special |$ConditionalOperators|))
        (if ss
          (progn
            (loop for u in ss do (push (rest u) |$ConditionalOperators|))
            (if (and (consp s) (eq (qfirst s) '|Join|))
              (progn
                (if (setq u (assq 'category ss))
                  (msubst (append u ss) u s)
                  (cons '|Join|
                    (append (rest s) (list (cons 'category (cons '|package| ss)))))))
              (list '|Join| s (cons 'category (cons '|package| ss))))
            s)))
          (fn (a s)
            (declare (special |$CategoryFrame|))
            (if (|isCategoryForm| s |$CategoryFrame|)
              (if (and (consp s) (eq (qfirst s) '|Join|))
                (|genDomainViewList0| a (rest s))
                (list (|genDomainView| a s '|getDomainView|)))
              (list a)))
            (findExtras (a target)
              (cond
                ((and (consp target) (eq (qfirst target) '|Join|))
                  (reduce #'|union|
                    (loop for x in (qrest target)
                      collect (findExtras a x))))
                ((and (consp target) (eq (qfirst target) 'category))
                  (reduce #'|union|
                    (loop for x in (qcddr target)
                      collect (findExtras1 a x))))))
              (findExtras1 (a x)
                (cond
                  ((and (consp x) (or (eq (qfirst x) 'and)) (eq (qfirst x) 'or))
                    (reduce #'|union|
                      (loop for y in (rest x) collect (findExtras1 a y))))
                  ((and (consp x) (eq (qfirst x) 'if)
                    (consp (qrest x)) (consp (qcddr x))
                    (consp (qcdddr x))
                    (eq (qcdddr x) nil))
                    (|union| (findExtrasP a (second x))
                      (|union|
                        (findExtras1 a (third x))
```

```

        (findExtras1 a (fourth x))))))
(findExtrasP (a x)
  (cond
    ((and (consp x) (or (eq (qfirst x) 'and)) (eq (qfirst x) 'or))
      (reduce #'|union|
        (loop for y in (rest x) collect (findExtrasP a y))))
    ((and (consp x) (eq (qfirst x) '|has|)
      (consp (qrest x)) (consp (qcddr x))
      (consp (qcdddr x))
      (eq (qcdddr x) nil))
      (|union| (findExtrasP a (second x))
        (|union|
          (findExtras1 a (third x))
          (findExtras1 a (fourth x))))))
    ((and (consp x) (eq (qfirst x) '|has|)
      (consp (qrest x)) (equal (qsecond x) a)
      (consp (qcddr x))
      (eq (qcdddr x) nil)
      (consp (qthird x))
      (eq (qcaaddr x) 'signature))
      (list (third x))))))
)
(let (|$alternateViewList| |$forceAdd| |$ConditionalOperators|)
  (declare (special |$alternateViewList| |$forceAdd| |$ConditionalOperators|))
  (setq |$alternateViewList| nil)
  (setq |$forceAdd| t)
  (setq |$ConditionalOperators| nil)
  (mapcar #'reduce
    (loop for a in argl for s in sigl do
      (fn a (augmentSig s (findExtras a target))))))
)

```

5.2.76 defun genDomainViewList0

[getDomainViewList p??]

— defun genDomainViewList0 —

```

(defun |genDomainViewList0| (id catlist)
  (|genDomainViewList| id catlist t))

```

5.2.77 defun genDomainViewList

```
[qcdr p??]
[isCategoryForm p??]
[genDomainView p201]
[genDomainViewList p201]
[$EmptyEnvironment p??]
```

— **defun genDomainViewList** —

```
(defun |genDomainViewList| (id catlist firsttime)
  (declare (special |$EmptyEnvironment|) (ignore firsttime))
  (cond
    ((null catlist) nil)
    ((and (consp catlist) (eq (qrest catlist) nil)
      (null (|isCategoryForm| (first catlist) |$EmptyEnvironment|)))
      nil)
    (t
     (cons
      (|genDomainView| id (first catlist) '|getDomainView|)
      (|genDomainViewList| id (rest catlist) nil))))))
```

—————

5.2.78 defun genDomainView

```
[genDomainOps p202]
[qcar p??]
[qcdr p??]
[augModemapsFromCategory p245]
[mkDomainConstructor p??]
[member p??]
[$e p??]
[$getDomainCode p??]
```

— **defun genDomainView** —

```
(defun |genDomainView| (name c viewSelector)
  (let (code cd)
    (declare (special |$getDomainCode| |$e|))
    (cond
      ((and (consp c) (eq (qfirst c) 'category) (consp (qrest c)))
       (|genDomainOps| name name c))
      (t
       (setq code
        (if (and (consp c) (eq (qfirst c) '|SubsetCategory|)
```

```

      (consp (qrest c)) (consp (qcddr c))
      (eq (qcddr c) nil))
    (second c)
    c))
  (setq |$e| (|augModemapsFromCategory| name nil c |$e|))
  (setq cd
    (list 'let name (list viewSelector name (|mkDomainConstructor| code))))
  (unless (|member| cd |$getDomainCode|)
    (setq |$getDomainCode| (cons cd |$getDomainCode|)))
  name))))

```

5.2.79 defun genDomainOps

```

[getOperationAlist p250]
[substNames p251]
[mkq p??]
[mkDomainConstructor p??]
[addModemap p253]
[$e p??]
[$ConditionalOperators p??]
[$getDomainCode p??]

```

— defun genDomainOps —

```

(defun |genDomainOps| (viewName dom cat)
  (let (siglist oplist cd i)
    (declare (special |$e| |$ConditionalOperators| |$getDomainCode|))
    (setq oplist (|getOperationAlist| dom dom cat))
    (setq siglist (loop for lst in oplist collect (first lst)))
    (setq oplist (|substNames| dom viewName dom oplist))
    (setq cd
      (list 'let viewName
        (list 'mkOpVec| dom
          (cons 'list
            (loop for opsig in siglist
              collect
                (list 'list (mkq (first opsig))
                  (cons 'list
                    (loop for mode in (rest opsig)
                      collect (|mkDomainConstructor| mode))))))))))
    (setq |$getDomainCode| (cons cd |$getDomainCode|))
    (setq i 0)
    (loop for item in oplist do
      (if (|member| (first item) |$ConditionalOperators|)
        (setq |$e| (|addModemap| (caar item) dom (cadar item) nil)

```

```

      (list 'elt viewName (incf i)) |$e|))
    (setq |$e| (|addModemap| (caar item) dom (cadar item) (second item)
      (list 'elt viewName (incf i)) |$e|))))
  viewName))

```

5.2.80 defun mkOpVec

```

[getPrincipalView p??]
[getOperationAlistFromLisplib p??]
[opOf p??]
[length p??]
[assq p??]
[assoc p??]
[qcar p??]
[qcdr p??]
[sublis p??]
[AssocBarGensym p204]
[msubst p??]
[$FormalMapVariableList p250]
[Undef p??]

```

— defun mkOpVec —

```

(defun |mkOpVec| (dom siglist)
  (let (substargs oplist ops u noplist i tmp1)
    (declare (special |$FormalMapVariableList| |Undef|))
    (setq dom (|getPrincipalView| dom))
    (setq substargs
      (cons (cons '$ (elt dom 0))
        (loop for a in |$FormalMapVariableList| for x in (rest (elt dom 0))
          collect (cons a x))))
    (setq oplist (|getOperationAlistFromLisplib| (|opOf| (elt dom 0))))
    (setq ops (make-array (|#| siglist)))
    (setq i -1)
    (loop for opSig in siglist do
      (incf i)
      (setq u (assq (first opSig) oplist))
      (setq tmp1 (|assoc| (second opSig) u))
      (cond
        ((and (consp tmp1) (consp (qrest tmp1))
          (consp (qcddr tmp1)) (consp (qcdddr tmp1))
          (eq (qcdddr tmp1) nil)
          (eq (qfourth tmp1) 'elt))
          (setelt ops i (elt dom (second tmp1)))))

```

```

(t
  (setq noplist (sublis substargs u))
  (setq tmp1
    (|AssocBarGensym| (msubst (elt dom 0) '$ (second opSig)) noplist))
  (cond
    ((and (consp tmp1) (consp (qrest tmp1)) (consp (qcddr tmp1))
      (consp (qcdddr tmp1))
      (eq (qcdddr tmp1) nil)
      (eq (qfourth tmp1) 'elt))
      (setelt ops i (elt dom (second tmp1))))
    (t
      (setelt ops i (cons |Undef| (cons (list (elt dom 0) i) opSig))))))
ops))

```

5.2.81 defun AssocBarGensym

[EqualBarGensym p228]

— defun AssocBarGensym —

```

(defun |AssocBarGensym| (key z)
  (loop for x in z
    do (when (and (consp x) (|EqualBarGensym| key (car x))) (return x))))

```

5.2.82 defun compDefWhereClause

```

[qcar p??]
[qcdr p??]
[getmode p??]
[userError p??]
[concat p??]
[lassoc p??]
[pairList p??]
[union p??]
[listOfIdentifiersIn p??]
[delete p??]
[orderByDependency p207]
[assocleft p??]
[assocright p??]
[comp p557]

```

```
[$sigAlist p??]
[$predAlist p??]
```

— defun compDefWhereClause —

```
(defun |compDefWhereClause| (arg mode env)
  (labels (
    (transformType (x)
      (declare (special |$sigAlist|))
      (cond
        ((atom x) x)
        ((and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
              (consp (qcddr x)) (eq (qcdddr x) nil))
          (setq |$sigAlist|
            (cons (cons (second x) (transformType (third x)))
              |$sigAlist|))
          x)
        ((and (consp x) (eq (qfirst x) '|Record|)) x)
        (t
         (cons (first x)
              (loop for y in (rest x)
                    collect (transformType y))))))
    (removeSuchthat (x)
      (declare (special |$predAlist|))
      (if (and (consp x) (eq (qfirst x) '|\\|') (consp (qrest x))
              (consp (qcddr x)) (eq (qcdddr x) nil))
          (progn
            (setq |$predAlist| (cons (cons (second x) (third x)) |$predAlist|)
              (second x))
            x))
          (fetchType (a x env form)
            (if x
              x
              (or (|getmodel| a env)
                  (|userError| (|concat|
                    "There is no mode for argument" a "of function" (first form))))))
          (addSuchthat (x y)
            (let (p)
              (declare (special |$predAlist|))
              (if (setq p (lassoc x |$predAlist|)) (list '|\\| y p) y)))
          )
    (let (|$sigAlist| |$predAlist| form signature specialCases body sigList
          argList argSigAlist argDepAlist varList whereList formxx signaturex
          deform formx)
      (declare (special |$sigAlist| |$predAlist|))
      ; form is lhs (f a1 ... an) of definition; body is rhs;
      ; signature is (t0 t1 ... tn) where t0= target type, ti=type of ai, i > 0;
      ; specialCases is (NIL l1 ... ln) where li is list of special cases
      ; which can be given for each ti
      ;
```

```

; removes declarative and assignment information from form and
; signature, placing it in list L, replacing form by ("where",form',:L),
; signature by a list of NILs (signifying declarations are in e)
(setq form (second arg))
(setq signature (third arg))
(setq specialCases (fourth arg))
(setq body (fifth arg))
(setq |$sigAlist| nil)
(setq |$predAlist| nil)
; 1. create sigList= list of all signatures which have embedded
; declarations moved into global variable $sigAlist
(setq sigList
  (loop for a in (rest form) for x in (rest signature)
    collect (transformType (fetchType a x env form))))
; 2. replace each argument of the form (| x p) by x, recording
; the given predicate in global variable $predAlist
(setq argList
  (loop for a in (rest form)
    collect (removeSuchthat a)))
(setq argSigAlist (append |$sigAlist| (|pairList| argList sigList)))
(setq argDepAlist
  (loop for pear in argSigAlist
    collect
      (cons (car pear)
        (|union| (|listOfIdentifiersIn| (cdr pear))
          (|delete| (car pear)
            (|listOfIdentifiersIn| (lassoc (car pear) |$predAlist|)))))))
; 3. obtain a list of parameter identifiers (x1 .. xn) ordered so that
; the type of xi is independent of xj if i < j
(setq varList
  (|orderByDependency| (assocleft argDepAlist) (assocright argDepAlist)))
; 4. construct a WhereList which declares and/or defines the xi's in
; the order constructed in step 3
(setq whereList
  (loop for x in varList
    collect (addSuchthat x (list '|| x (lassoc x argSigAlist)))))
(setq formxx (cons (car form) argList))
(setq signaturex
  (cons (car signature)
    (loop for x in (rest signature) collect nil)))
(setq defform (list 'def formxx signaturex specialCases body))
(setq formx (cons '|where| (cons defform whereList)))
; 5. compile new ('DEF,("where",form',:WhereList),:.) where
; all argument parameters of form' are bound/declared in WhereList
(|comp| formx mode env)))

```

5.2.83 defun orderByDependency

```
[say p??]
[userError p??]
[intersection p??]
[member p??]
[remdup p??]
```

— defun orderByDependency —

```
(defun |orderByDependency| (vl dl)
  (let (selfDependents fatalError newl orderedVarList vlp dlp)
    (setq selfDependents
      (loop for v in vl for d in dl
        when (member v d)
        collect v))
    (loop for v in vl for d in dl
      when (member v d)
      do (say v "depends on itself")
        (setq fatalError t))
    (cond
      (fatalError (|userError| "Parameter specification error"))
      (t
        (loop until (null vl) do
          (setq newl
            (loop for v in vl for d in dl
              when (null (|intersection| d vl))
              collect v))
          (if (null newl)
            (setq vl nil) ; force loop exit
            (progn
              (setq orderedVarList (append newl orderedVarList))
              (setq vlp (setdifference vl newl))
              (setq dlp
                (loop for x in vl for d in dl
                  when (|member| x vlp)
                  collect (setdifference d newl)))
              (setq vl vlp)
              (setq dl dlp))))
          (when (and newl orderedVarList) (remdup (nreverse orderedVarList)))))))
```

—————

5.3 Code optimization routines

5.3.1 defun optimizeFunctionDef

```
[qcar p??]
[qcdr p??]
[rplac p??]
[sayBrightlyI p??]
[optimize p209]
[pp p??]
[bright p??]
[$reportOptimization p??]
```

— defun optimizeFunctionDef —

```
(defun |optimizeFunctionDef| (def)
  (labels (
    (fn (x g)
      (cond
        ((and (consp x) (eq (qfirst x) 'throw) (consp (qrest x))
          (equal (qsecond x) g))
         (|rplac| (car x) 'return)
         (|rplac| (cdr x)
          (replaceThrowByReturn (qcddr x) g)))
        ((atom x) nil)
        (t
         (replaceThrowByReturn (car x) g)
         (replaceThrowByReturn (cdr x) g))))
    (replaceThrowByReturn (x g)
      (fn x g)
        x)
    (removeTopLevelCatch (body)
      (if (and (consp body) (eq (qfirst body) 'catch) (consp (qrest body))
        (consp (qcddr body)) (eq (qcdddr body) nil))
        (removeTopLevelCatch
          (replaceThrowByReturn
            (qthird body) (qsecond body)))
        body)))
    (let (defp name slamOrLam args body bodyp)
      (declare (special |$reportOptimization|)
        (when |$reportOptimization|
          (|sayBrightlyI| (|bright| "Original LISP code:"))
          (|pp| def))
        (setq defp (|optimize| (copy def)))
        (when |$reportOptimization|
          (|sayBrightlyI| (|bright| "Optimized LISP code:"))
          (|pp| defp)
          (|sayBrightlyI| (|bright| "Final LISP code:")))))
```

```

(setq name (car defp))
(setq slam0rLam (caadr defp))
(setq args (cadadr defp))
(setq body (car (cddadr defp)))
(setq bodyp (removeTopLevelCatch body))
(list name (list slam0rLam args bodyp))))

```

5.3.2 defun optimize

```

[qcar p??]
[qcdr p??]
[optimize p209]
[say p??]
[prettyprint p??]
[rplac p??]
[optIF2COND p212]
[getl p??]
[subrname p212]

```

— defun optimize —

```

(defun |optimize| (x)
  (labels (
    (opt (x)
      (let (argl body a y op)
        (cond
          ((atom x) nil)
          ((eq (setq y (car x)) 'quote) nil)
          ((eq y 'closedfn) nil)
          ((and (consp y) (consp (qfirst y)) (eq (qcaar y) 'xlam)
               (consp (qcdar y)) (consp (qcddar y))
               (eq (qcdddar y) nil))
           (setq argl (qcadar y))
           (setq body (qcaddar y))
           (setq a (qrest y))
           (|optimize| (cdr x))
           (cond
             ((eq argl '|ignore|) (rplac (car x) body))
             (t
              (when (null (<= (length argl) (length a)))
                (say "length mismatch in XLAM expression")
                (prettyprint y))
              (rplac (car x)
                    (|optimize|
                     (|optXLAMCond|

```

```

(sublis (|pairList| argl a) body))))))
((atom y)
 (|optimize| (cdr x))
 (cond
  ((eq y '|true|) (rplac (car x) ''T))
  ((eq y '|false|) (rplac (car x) nil))))
((eq (car y) 'if)
 (rplac (car x) (|optIF2COND| y))
 (setq y (car x))
 (when (setq op (get1 (|subrname| (car y)) 'optimize))
  (|optimize| (cdr x))
  (rplac (car x) (funcall op (|optimize| (car x))))))
((setq op (get1 (|subrname| (car y)) 'optimize))
 (|optimize| (cdr x))
 (rplac (car x) (funcall op (|optimize| (car x))))))
(t
 (rplac (car x) (|optimize| (car x)))
 (|optimize| (cdr x))))))
(opt x)
x))

```

5.3.3 defun optXLAMCond

```

[optCONDtail p211]
[optPredicateIfTrue p211]
[optXLAMCond p210]
[qcar p??]
[qcdr p??]
[rplac p??]

```

— defun optXLAMCond —

```

(defun |optXLAMCond| (x)
 (cond
  ((and (consp x) (eq (qfirst x) 'cond) (consp (qrest x))
        (consp (qsecond x)) (consp (qcddadr x))
        (eq (qcddadr x) nil))
   (if (|optPredicateIfTrue| (qcaadr x))
       (qcadadr x)
       (cons 'cond (cons (qsecond x) (|optCONDtail| (qcddr x)))))
  ((atom x) x)
  (t
   (rplac (car x) (|optXLAMCond| (car x)))
   (rplac (cdr x) (|optXLAMCond| (cdr x)))
   x)))

```

5.3.4 defun optCONDtail

[optCONDtail p211]
 [\$true p??]

— defun optCONDtail —

```
(defun |optCONDtail| (z)
  (declare (special |$true|))
  (when z
    (cond
      ((|optPredicateIfTrue| (caar z)) (list (list |$true| (cadar z))))
      ((null (cdr z)) (list (car z) (list |$true| (list '|CondError|))))
      (t (cons (car z) (|optCONDtail| (cdr z)))))))
```

5.3.5 defvar \$BasicPredicates

If these predicates are found in an expression the code optimizer routine `optPredicateIfTrue` then `optXLAM` will replace the call with the argument. This is used for predicates that test the type of their argument so that, for instance, a call to `integerp` on an integer will be replaced by that integer if it is true. This represents a simple kind of compile-time type evaluation.

— initvars —

```
(defvar |$BasicPredicates| '(integerp stringp floatp))
```

5.3.6 defun optPredicateIfTrue

[\$BasicPredicates p211]

— defun optPredicateIfTrue —

```
(defun |optPredicateIfTrue| (p)
  (declare (special |$BasicPredicates|))
  (cond
```

```

((and (consp p) (eq (qfirst p) 'quote)) T)
((and (consp p) (consp (qrest p)) (eq (qcddr p) nil)
  (member (qfirst p) |$BasicPredicates|) (funcall (qfirst p) (qsecond p)))
  t)
(t nil)))

```

5.3.7 defun optIF2COND

```

[optIF2COND p212]
[$true p??]

```

— defun optIF2COND —

```

(defun |optIF2COND| (arg)
  (let (a b c)
    (declare (special |$true|))
    (setq a (cadr arg))
    (setq b (caddr arg))
    (setq c (caddr arg))
    (cond
      ((eq b '|noBranch|) (list 'cond (list (list 'null a ) c)))
      ((eq c '|noBranch|) (list 'cond (list a b)))
      ((and (consp c) (eq (qfirst c) 'if))
        (cons 'cond (cons (list a b) (cdr (|optIF2COND| c)))))
      ((and (consp c) (eq (qfirst c) 'cond))
        (cons 'cond (cons (list a b) (qrest c)))))
    (t
      (list 'cond (list a b) (list |$true| c))))))

```

5.3.8 defun subrname

```

[identp p??]
[compiled-function-p p??]
[mbpip p??]
[bpname p??]

```

— defun subrname —

```

(defun |subrname| (u)
  (cond
    ((identp u) u)

```

```
((or (compiled-function-p u) (mbpip u)) (bpiname u))
(t nil)))
```

5.3.9 Special case optimizers

Optimization functions are called through the OPTIMIZE property on the symbol property list. The current list is:

call	optCall
seq	optSEQ
eq	optEQ
minus	optMINUS
qsminus	optQSMINUS
-	opt-
lessp	optLESSP
spadcall	optSPADCALL
	optSuchthat
catch	optCatch
cond	optCond
mkRecord	optMkRecord
recordelt	optRECORDELT
setrecordelt	optSETRECORDELT
recordcopy	optRECORDCOPY

Be aware that there are case-sensitivity issues. When found in the s-expression, each symbol in the left column will call a custom optimization routine in the right column. The optimization routines are below. Note that each routine has a special chunk in postvars using eval-when to set the property list at load time.

These optimizations are done destructively. That is, they modify the function in-place using rplac.

Not all of the optimization routines are called through the property list. Some are called only from other optimization routines, e.g. optPackageCall.

5.3.10 defplist optCall

— postvars —

```
(eval-when (eval load)
  (setf (get '|call| 'optimize) '|optCall|))
```

5.3.11 defun Optimize “call” expressions

```
[optimize p209]
[rplac p??]
[optPackageCall p215]
[optCallSpecially p215]
[systemErrorHere p??]
[$QuickCode p??]
[$bootStrapMode p??]
```

— defun optCall —

```
(defun |optCall| (x)
  (let (u tmp1 fn a name q r n w)
    (declare (special |$QuickCode| |$bootStrapMode|))
    (setq u (cdr x))
    (setq x (|optimize| (list u)))
    (cond
      ((atom (car x)) (car x))
      (t
       (setq tmp1 (car x))
       (setq fn (car tmp1))
       (setq a (cdr tmp1))
       (cond
         ((atom fn) (rplac (cdr x) a) (rplac (car x) fn))
         ((and (consp fn) (eq (qfirst fn) 'pac)) (|optPackageCall| x fn a))
         ((and (consp fn) (eq (qfirst fn) '|applyFun|)
              (consp (qrest fn)) (eq (qcddr fn) nil))
          (setq name (qsecond fn))
          (rplac (car x) 'spadcall)
          (rplac (cdr x) (append a (cons name nil))))
         x)
       ((and (consp fn) (consp (qrest fn)) (consp (qcddr fn))
            (eq (qcddr fn) nil)
            (member (qfirst fn) '(elt qrefelt const)))
        (setq q (qfirst fn))
        (setq r (qsecond fn))
        (setq n (qthird fn))
        (cond
          ((and (null |$bootStrapMode|) (setq w (|optCallSpecially| q x n r)))
           w)
          ((eq q 'const)
           (list '|spadConstant| r n))
          (t
           (rplac (car x) 'spadcall)
           (when |$QuickCode| (rplaca fn 'qrefelt))
           (rplac (cdr x) (append a (list fn)))
           x)))
       (t (|systemErrorHere| "optCall"))))))))
```

5.3.12 defun optPackageCall

[rplaca p??]
[rplacd p??]

— defun optPackageCall —

```
(defun |optPackageCall| (x arg2 arglist)
  (let (packageVariableOrForm functionName)
    (setq packageVariableOrForm (second arg2))
    (setq functionName (third arg2))
    (rplaca x functionName)
    (rplacd x (append arglist (list packageVariableOrForm)))
    x))
```

5.3.13 defun optCallSpecially

[lassoc p??]
[kar p??]
[get p??]
[opOf p??]
[optSpecialCall p216]
[\$specialCaseKeyList p??]
[\$getDomainCode p??]
[\$optimizableConstructorNames p??]
[\$e p??]

— defun optCallSpecially —

```
(defun |optCallSpecially| (q x n r)
  (declare (ignore q))
  (labels (
    (lookup (a z)
      (let (zp)
        (when z
          (setq zp (car z))
          (setq z (cdr x))
          (if (and (consp zp) (eq (qfirst zp) 'let) (consp (qrest zp))
              (equal (qsecond zp) a) (consp (qcddr zp))))
```

```

      (qthird zp)
      (lookup a z))))))
(let (tmp1 op y prop yy)
(declare (special |$specialCaseKeyList| |$getDomainCode| |$e|
              |$optimizableConstructorNames|))
(cond
  ((setq y (lassoc r |$specialCaseKeyList|))
   (|optSpecialCall| x y n))
  ((member (kar r) |$optimizableConstructorNames|)
   (|optSpecialCall| x r n))
  ((and (setq y (|get| r '|value| |$e|))
        (member (|opOf| (car y)) |$optimizableConstructorNames|))
   (|optSpecialCall| x (car y) n))
  ((and (setq y (lookup r |$getDomainCode|))
        (progn
          (setq tmp1 y)
          (setq op (first tmp1))
          (setq y (second tmp1))
          (setq prop (third tmp1))
          tmp1)
        (setq yy (lassoc y |$specialCaseKeyList|)))
   (|optSpecialCall| x (list op yy prop) n))
  (t nil))))))

```

5.3.14 defun optSpecialCall

```

[optCallEval p218]
[function p??]
[keyedSystemError p??]
[mkq p??]
[getl p??]
[compileTimeBindingOf p217]
[rplac p??]
[optimize p209]
[rplacw p??]
[rplaca p??]
[$QuickCode p??]
[$Undef p??]

```

— defun optSpecialCall —

```

(defun |optSpecialCall| (x y n)
  (let (yval args tmp1 fn a)
    (declare (special |$QuickCode| |Undef|))
    (setq yval (|optCallEval| y))

```

```

(cond
  ((eq (caaar x) 'const)
    (cond
      ((equal (kar (elt yval n)) (|function| |Undef|))
        (|keyedSystemError| 'S2GE0016
          (list "optSpecialCall" "invalid constant"))))
      (t (mkq (elt yval n)))))
  ((setq fn (get1 (|compileTimeBindingOf| (car (elt yval n))) '|SPADreplace|))
    (|rplac| (cdr x) (cdr x))
    (|rplac| (car x) fn)
    (when (and (consp fn) (eq (qfirst fn) 'xlam))
      (setq x (car (|optimize| (list x)))))
    (if (and (consp x) (eq (qfirst x) 'equal) (progn (setq args (qrest x)) t))
      (rplacw x (def-equal args))
      x))
  (t
    (setq tmp1 (car x))
    (setq fn (car tmp1))
    (setq a (cdr tmp1))
    (rplac (car x) 'spadcall)
    (when (|$QuickCode| (rplaca fn 'qrefelt))
      (rplac (cdr x) (append a (list fn)))
      x))))

```

5.3.15 defun compileTimeBindingOf

```

[bpiname p??]
[keyedSystemError p??]
[moan p??]

```

— defun compileTimeBindingOf —

```

(defun |compileTimeBindingOf| (u)
  (let (name)
    (cond
      ((null (setq name (bpiname u)))
        (|keyedSystemError| 'S2000001 (list u)))
      ((eq name '|Undef|)
        (moan "optimiser found unknown function"))
      (t name))))

```

5.3.16 defun optCallEval

```
[qcar p??]
[List p??]
[Integer p??]
[Vector p??]
[PrimitiveArray p??]
[FactoredForm p??]
[Matrix p??]
[eval p??]
```

— defun optCallEval —

```
(defun |optCallEval| (u)
  (cond
    ((and (consp u) (eq (qfirst u) '|List|))
     (|List| (|Integer|)))
    ((and (consp u) (eq (qfirst u) '|Vector|))
     (|Vector| (|Integer|)))
    ((and (consp u) (eq (qfirst u) '|PrimitiveArray|))
     (|PrimitiveArray| (|Integer|)))
    ((and (consp u) (eq (qfirst u) '|FactoredForm|))
     (|FactoredForm| (|Integer|)))
    ((and (consp u) (eq (qfirst u) '|Matrix|))
     (|Matrix| (|Integer|)))
    (t
     (|eval| u))))
```

—————

5.3.17 defplist optSEQ

— postvars —

```
(eval-when (eval load)
  (setf (get 'seq 'optimize) '|optSEQ|))
```

—————

5.3.18 defun optSEQ

— defun optSEQ —

```

(defun |optSEQ| (arg)
  (labels (
    (tryToRemoveSEQ (z)
      (if (and (consp z) (eq (qfirst z) 'seq) (consp (qrest z))
            (eq (qcddr z) nil) (consp (qsecond z))
            (consp (qcaddr z))
            (eq (qcddadr z) nil)
            (member (qcaadr z) '(exit return throw)))
        (qcadadr z)
        z))
    (SEQToCOND (z)
      (let (transform before aft)
        (setq transform
          (loop for x in z
            while
              (and (consp x) (eq (qfirst x) 'cond) (consp (qrest x))
                    (eq (qcddr x) nil) (consp (qsecond x))
                    (consp (qcaddr x))
                    (eq (qcddadr x) nil)
                    (consp (qcadadr x))
                    (eq (qfirst (qcadadr x)) 'exit)
                    (consp (qrest (qcadadr x)))
                    (eq (qcddr (qcadadr x)) nil)))
            collect
              (list (qcaadr x)
                    (qsecond (qcadadr x)))))
        (setq before (take (|#| transform) z))
        (setq aft (|after| z before))
        (cond
          ((null before) (cons 'seq aft))
          ((null aft)
           (cons 'cond (append transform (list '(t (|conderr|))))))
          (t
           (cons 'cond (append transform
                                (list (list 't (|optSEQ| (cons 'seq aft))))))))
        (getRidOfTemps (z)
          (let (g x r)
            (cond
              ((null z) nil)
              ((and (consp z) (consp (qfirst z)) (eq (qcaar z) 'let)
                    (consp (qcdar z)) (consp (qcddar z))
                    (gensymp (qcadar z))
                    (> 2 (|numOfOccurrencesOf| (qcadar z) (qrest z))))
               (setq g (qcadar z))
               (setq x (qcaddar z))
               (setq r (qrest z))
               (getRidOfTemps (msubst x g r)))
              ((eq (car z) '(/throwAway|)
               (getRidOfTemps (cdr z)))
              (t

```

```
(cons (car z) (getRidOfTemps (cdr z)))))))))
(tryToRemoveSEQ (SEQToCOND (getRidOfTemps (cdr arg))))))
```

5.3.19 defplist optEQ

— postvars —

```
(eval-when (eval load)
  (setf (get 'eq 'optimize) '|optEQ|))
```

5.3.20 defun optEQ

— defun optEQ —

```
(defun |optEQ| (u)
  (let (z r)
    (cond
      ((and (consp u) (eq (qfirst u) 'eq) (consp (qrest u))
            (consp (qcddr u)) (eq (qcdddr u) nil))
       (setq z (qsecond u))
       (setq r (qthird u))
       ; That undoes some weird work in Boolean to do with the definition of true
       (if (and (numberp z) (numberp r))
           (list 'quote (eq z r))
           u))
      (t u))))
```

5.3.21 defplist optMINUS

— postvars —

```
(eval-when (eval load)
  (setf (get 'minus 'optimize) '|optMINUS|))
```

5.3.22 defun optMINUS

— defun optMINUS —

```
(defun |optMINUS| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) 'minus) (consp (qrest u))
        (eq (qcddr u) nil))
       (setq v (qsecond u))
       (cond ((numberp v) (- v)) (t u)))
      (t u))))
```

5.3.23 defplist optQSMINUS

— postvars —

```
(eval-when (eval load)
  (setf (get 'qsminus 'optimize) '|optQSMINUS|))
```

5.3.24 defun optQSMINUS

— defun optQSMINUS —

```
(defun |optQSMINUS| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) 'qsminus) (consp (qrest u))
        (eq (qcddr u) nil))
       (setq v (qsecond u))
       (cond ((numberp v) (- v)) (t u)))
      (t u))))
```

5.3.25 defplist opt-

— postvars —

```
(eval-when (eval load)
  (setf (get '- 'optimize) '|opt-|))
```

—————

5.3.26 defun opt-

— defun opt- —

```
(defun |opt-| (u)
  (let (v)
    (cond
      ((and (consp u) (eq (qfirst u) '-') (consp (qrest u))
            (eq (qcddr u) NIL))
       (setq v (qsecond u))
       (cond ((numberp v) (- v)) (t u)))
      (t u))))
```

—————

5.3.27 defplist optLESSP

— postvars —

```
(eval-when (eval load)
  (setf (get 'lessp 'optimize) '|optLESSP|))
```

—————

5.3.28 defun optLESSP

— defun optLESSP —

```
(defun |optLESSP| (u)
  (let (a b)
```

```
(cond
  ((and (consp u) (eq (qfirst u) 'lessp) (consp (qrest u))
        (consp (qcddr u))
        (eq (qcddr u) nil))
    (setq a (qsecond u))
    (setq b (qthird u))
    (if (eql b 0)
        (list 'minusp a)
        (list '> b a)))
  (t u))))
```

5.3.29 defplist optSPADCALL

— postvars —

```
(eval-when (eval load)
  (setf (get 'spadcall 'optimize) '|optSPADCALL|))
```

5.3.30 defun optSPADCALL

[optCall p214]
[\$InteractiveMode p??]

— defun optSPADCALL —

```
(defun |optSPADCALL| (form)
  (let (fun argl tmp1 dom slot)
    (declare (special |$InteractiveMode|))
    (setq argl (cdr form))
    (cond
      ; last arg is function/env, but may be a form
      ((null |$InteractiveMode|) form)
      ((and (consp argl)
            (progn (setq tmp1 (reverse argl)) t)
            (consp tmp1))
       (setq fun (qfirst tmp1))
       (setq argl (qrest tmp1))
       (setq argl (nreverse argl)))
      (cond
        ((and (consp fun)
```

```

(or (eq (qfirst fun) 'elt) (eq (qfirst fun) 'lispelt))
(progn
  (and (consp (qrest fun))
    (progn
      (setq dom (qsecond fun))
      (and (consp (qcddr fun))
        (eq (qcdddr fun) nil)
        (progn
          (setq slot (qthird fun))
          t))))))
  (|optCall| (cons '|call| (cons (list 'elt dom slot) argl))))
(t form)))
(t form)))

```

5.3.31 defplist optSuchthat

— postvars —

```

(eval-when (eval load)
  (setf (get '|\\| 'optimize) '|optSuchthat|))

```

5.3.32 defun optSuchthat

— defun optSuchthat —

```

(defun |optSuchthat| (arg)
  (cons 'suchthat (cdr arg)))

```

5.3.33 defplist optCatch

— postvars —

```

(eval-when (eval load)
  (setf (get 'catch 'optimize) '|optCatch|))

```

5.3.34 defun optCatch

```
[qcar p??]
[qcdr p??]
[rplac p??]
[optimize p209]
[$InteractiveMode p??]
```

— defun optCatch —

```
(defun |optCatch| (x)
  (labels (
    (changeThrowToExit (s g)
      (cond
        ((or (atom s) (member (car s) '(quote seq repeat collect))) nil)
        ((and (consp s) (eq (qfirst s) 'throw) (consp (qrest s))
          (equal (qsecond s) g))
          (|rplac| (car s) 'exit)
          (|rplac| (cdr s) (qcddr s)))
        (t
          (changeThrowToExit (car s) g)
          (changeThrowToExit (cdr s) g))))
    (hasNoThrows (a g)
      (cond
        ((and (consp a) (eq (qfirst a) 'throw) (consp (qrest a))
          (equal (qsecond a) g))
          nil)
        ((atom a) t)
        (t
          (and (hasNoThrows (car a) g)
            (hasNoThrows (cdr a) g))))))
    (changeThrowToGo (s g)
      (let (u)
        (cond
          ((or (atom s) (eq (car s) 'quote)) nil)
          ((and (consp s) (eq (qfirst s) 'throw) (consp (qrest s))
            (equal (qsecond s) g) (consp (qcddr s))
            (eq (qcddr s) nil))
            (setq u (qthird s))
            (changeThrowToGo u g)
            (|rplac| (car s) 'progn)
            (|rplac| (cdr s) (list (list 'let (cadr g) u) (list 'go (cadr g)))))
          (t
            (changeThrowToGo (car s) g)
            (changeThrowToGo (cdr s) g))))))
    (let (g tmp2 u s tmp6 a)
```

```

(declare (special |$InteractiveModel|))
(setq g (cadr x))
(setq a (caddr x))
(cond
  (|$InteractiveModel| x)
  ((atom a) a)
  (t
   (cond
    ((and (consp a) (eq (qfirst a) 'seq) (consp (qrest a))
      (progn (setq tmp2 (reverse (qrest a))) t)
      (consp tmp2) (consp (qfirst tmp2)) (eq (qcaar tmp2) 'throw)
      (consp (qcдар tmp2))
      (equal (qcadar tmp2) g)
      (consp (qcddar tmp2))
      (eq (qcdddar tmp2) nil))
     (setq u (qcdddar tmp2))
     (setq s (qrest tmp2))
     (setq s (nreverse s))
     (changeThrowToExit s g)
     (|rplac| (cdr a) (append s (list (list 'exit u)))))
    (setq tmp6 (|optimize| x))
    (setq a (caddr tmp6))))
   (cond
    ((hasNoThrows a g)
     (|rplac| (car x) (car a))
     (|rplac| (cdr x) (cdr a)))
    (t
     (changeThrowToGo a g)
     (|rplac| (car x) 'seq)
     (|rplac| (cdr x)
      (list (list 'exit a) (cadr g) (list 'exit (cadr g)))))
    x))))

```

5.3.35 defplist optCond

— postvars —

```

(eval-when (eval load)
  (setf (get 'cond 'optimize) '|optCond|))

```

```

      (consp (qthird z))
      (|TruthP| (qcaaddr z)))
  (setq p1 (qcaar z))
  (setq c1 (qcdar z))
  (setq p2 (qcaadr z))
  (setq c2 (qcdadr z))
  (setq c3 (qcdaddr z))
  (cond
   ((|EqualBarGensym| c1 c3)
    (list 'cond
      (cons (list 'or p1 (list 'null p2)) c1) (cons (list 'quote t) c2)))
   ((|EqualBarGensym| c1 c2)
    (list 'cond (cons (list 'or p1 p2) c1) (cons (list 'quote t) c3)))
   (t x)))
  (t
   (do ((y z (cdr y)))
       ((atom y) nil)
       (do ()
          ((null (and (consp y) (consp (qfirst y)) (consp (qcdar y))
                     (eq (qcddar y) nil) (consp (qrest y))
                     (consp (qsecond y)) (consp (qcdadr y))
                     (eq (qcddadr y) nil)
                     (|EqualBarGensym| (qcadar y)
                                       (qcadadr y)))))
          nil)
        (setq a (list 'or (qcaar y) (qcaadr y)))
        (rplac (car (car y)) a)
        (rplac (cdr y) (qcddr y))))
   x))))

```

5.3.37 defun EqualBarGensym

```

[gensymp p??]
[$GensymAssoc p??]
[$GensymAssoc p??]

```

— defun EqualBarGensym —

```

(defun |EqualBarGensym| (x y)
  (labels (
    (fn (x y)
      (let (z)
        (declare (special |$GensymAssoc|))
        (cond
         ((equal x y) t)

```

```

((and (gensymp x) (gensymp y))
 (if (setq z (|assoc| x |$GensymAssoc|))
  (if (equal y (cdr z)) t nil)
  (progn
   (setq |$GensymAssoc| (cons (cons x y) |$GensymAssoc|))
   t)))
((null x) (and (consp y) (eq (qrest y) nil) (gensymp (qfirst y))))
((null y) (and (consp x) (eq (qrest x) nil) (gensymp (qfirst x))))
((or (atom x) (atom y)) nil)
(t
 (and (fn (car x) (car y))
  (fn (cdr x) (cdr y))))))
(let (|$GensymAssoc|)
 (declare (special |$GensymAssoc|))
 (setq |$GensymAssoc| NIL)
 (fn x y)))

```

5.3.38 defplist optMkRecord

— postvars —

```

(eval-when (eval load)
 (setf (get '|mkRecord| 'optimize) '|optMkRecord|))

```

5.3.39 defun optMkRecord

[length p??]

— defun optMkRecord —

```

(defun |optMkRecord| (arg)
 (let (u)
  (setq u (cdr arg))
  (cond
   ((and (consp u) (eq (qrest u) nil)) (list 'list (qfirst u)))
   ((eql (|#| u) 2) (cons 'cons u))
   (t (cons 'vector u)))))

```

5.3.40 defplist optRECORDELT

— postvars —

```
(eval-when (eval load)
  (setf (get 'recordelt 'optimize) '|optRECORDELT|))
```

—————

5.3.41 defun optRECORDELT

[keyedSystemError p??]

— defun optRECORDELT —

```
(defun |optRECORDELT| (arg)
  (let (name ind len)
    (setq name (cadr arg))
    (setq ind (caddr arg))
    (setq len (caddr arg))
    (cond
      ((eq len 1)
       (cond
         ((eq ind 0) (list 'qcar name))
         (t (|keyedSystemError| 'S2000002 (list ind)))))
      ((eq len 2)
       (cond
         ((eq ind 0) (list 'qcar name))
         ((eq ind 1) (list 'qcdr name))
         (t (|keyedSystemError| 'S2000002 (list ind)))))
      (t (list 'qvelt name ind)))))
```

—————

5.3.42 defplist optSETRECORDELT

— postvars —

```
(eval-when (eval load)
  (setf (get 'setrecordelt 'optimize) '|optSETRECORDELT|))
```

—————

5.3.43 defun optSETRECORDELT

[keyedSystemError p??]

— defun optSETRECORDELT —

```

(defun |optSETRECORDELT| (arg)
  (let (name ind len expr)
    (setq name (cadr arg))
    (setq ind (caddr arg))
    (setq len (caddr arg))
    (setq expr (car (caddr arg)))
    (cond
      ((eql len 1)
       (if (eql ind 0)
           (list 'progn (list 'rplaca name expr) (list 'qcar name))
           (|keyedSystemError| 'S2000002 (list ind))))
      ((eql len 2)
       (cond
         ((eql ind 0)
          (list 'progn (list 'rplaca name expr) (list 'qcar name)))
         ((eql ind 1)
          (list 'progn (list 'rplacd name expr) (list 'qcdr name)))
         (t (|keyedSystemError| 'S2000002 (list ind)))))
      (t
       (list 'qsetvelt name ind expr)))))

```

—

5.3.44 deflist optRECORDCOPY

— postvars —

```

(eval-when (eval load)
  (setf (get 'recordcopy 'optimize) '|optRECORDCOPY|))

```

—

5.3.45 defun optRECORDCOPY

— defun optRECORDCOPY —

```

(defun |optRECORDCOPY| (arg)

```

```

(let (name len)
  (setq name (cadr arg))
  (setq len (caddr arg))
  (cond
    ((eq len 1) (list 'list (list 'car name)))
    ((eq len 2) (list 'cons (list 'car name) (list 'cdr name)))
    (t
     (list 'replace (list 'make-array len) name))))

```

5.4 Functions to manipulate modemap

5.4.1 defun addDomain

```

[identp p??]
[qslessp p??]
[getDomainsInScope p234]
[domainMember p244]
[isLiteral p??]
[addNewDomain p236]
[getmode p??]
[isCategoryForm p??]
[isFunctor p233]
[constructor? p??]
[member p??]
[unknownTypeError p233]

```

— defun addDomain —

```

(defun |addDomain| (domain env)
  (let (s name tmp1)
    (cond
      ((atom domain)
       (cond
         ((eq domain '|$EmptyMode|) env)
         ((eq domain '|$NoValueMode|) env)
         ((or (null (identp domain))
              (and (qslessp 2 (|#| (setq s (princ-to-string domain))))
                   (eq (|char| '|#|) (elt s 0))
                   (eq (|char| '|#|) (elt s 1))))
          env)
         ((member domain (|getDomainsInScope| env)) env)
         ((|isLiteral| domain env) env)
         (t (|addNewDomain| domain env))))
      ((eq (setq name (car domain)) '|Category|) env)
      ((|domainMember| domain (|getDomainsInScope| env)) env)

```

```

((and (progn
      (setq tmp1 (|getmode| name env))
      (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)
           (consp (qrest tmp1))))
      (|isCategoryForm| (second tmp1) env))
      (|addNewDomain| domain env))
      ((or (|isFunction| name) (|constructor?| name))
       (|addNewDomain| domain env))
      (t
       (when (and (null (|isCategoryForm| domain env))
                  (null (|member| name '|Mapping| category))))
         (|unknownTypeError| name))
       env))))

```

5.4.2 defun unknownTypeError

```

[qcar p??]
[stackSemanticError p??]

```

— defun unknownTypeError —

```

(defun |unknownTypeError| (name)
  (let (op)
    (setq name
      (if (and (consp name) (setq op (qfirst name)))
          op
          name))
    (|stackSemanticError| (list '|%b| name '|%d| '|is not a known type|) nil)))

```

5.4.3 defun isFunctor

```

[opOf p??]
[identp p??]
[getdatabase p??]
[get p??]
[constructor? p??]
[updateCategoryFrameForCategory p113]
[updateCategoryFrameForConstructor p112]
[$CategoryFrame p??]
[$InteractiveMode p??]

```

— defun isFunctor —

```
(defun |isFunctor| (x)
  (let (op u prop)
    (declare (special |$CategoryFrame| |$InteractiveMode|))
    (setq op (|opOf| x))
    (cond
      ((null (identp op)) nil)
      (|$InteractiveMode|
       (if (member op '(|Union| |SubDomain| |Mapping| |Record|))
           t
           (member (getdatabase op 'constructorkind) '(|domain| |package|))))
      ((setq u
        (or (|get| op 'isFunctor| |$CategoryFrame|)
            (member op '(|SubDomain| |Union| |Record|))))
         u)
      ((|constructor?| op)
       (cond
         ((setq prop (|get| op 'isFunctor| |$CategoryFrame|)) prop)
         (t
          (if (eq (getdatabase op 'constructorkind) '|category|)
              (|updateCategoryFrameForCategory| op)
              (|updateCategoryFrameForConstructor| op))
          (|get| op 'isFunctor| |$CategoryFrame|))))
      (t nil))))
```

—

5.4.4 defun getDomainsInScope

The way XLAMs work:

```
((XLAM ($1 $2 $3) (SETELT $1 0 $3)) X "c" V) ==> (SETELT X 0 V)
```

```
[get p??]
[$CapsuleDomainsInScope p??]
[$insideCapsuleFunctionIfTrue p??]
```

— defun getDomainsInScope —

```
(defun |getDomainsInScope| (env)
  (declare (special |$CapsuleDomainsInScope| |$insideCapsuleFunctionIfTrue|))
  (if |$insideCapsuleFunctionIfTrue|
      |$CapsuleDomainsInScope|
      (|get| '|$DomainsInScope| 'special env)))
```

5.4.5 defun putDomainsInScope

```
[getDomainsInScope p234]
[put p??]
[delete p??]
[say p??]
[member p??]
[$CapsuleDomainsInScope p??]
[$insideCapsuleFunctionIfTrue p??]
```

— defun putDomainsInScope —

```
(defun |putDomainsInScope| (x env)
  (let (z newValue)
    (declare (special |$CapsuleDomainsInScope| |$insideCapsuleFunctionIfTrue|))
    (setq z (|getDomainsInScope| env))
    (when (|member| x z) (say "***** Domain: " x " already in scope"))
    (setq newValue (cons x (|delete| x z)))
    (if |$insideCapsuleFunctionIfTrue|
        (progn
          (setq |$CapsuleDomainsInScope| newValue)
          env)
        (|put| '|$DomainsInScope| 'special newValue env))))
```

5.4.6 defun isSuperDomain

```
[isSubset p??]
[lassoc p??]
[opOf p??]
[get p??]
```

— defun isSuperDomain —

```
(defun |isSuperDomain| (domainForm domainFormp env)
  (cond
    ((|isSubset| domainFormp domainForm env) t)
    ((and (eq domainForm '|Rep|) (eq domainFormp '$)) t)
    (t (lassoc (|opOf| domainFormp) (|get| domainForm '|SubDomain| env)))))
```

5.4.7 defun addNewDomain

[augModemapsFromDomain p236]

— defun addNewDomain —

```
(defun |addNewDomain| (domain env)
  (|augModemapsFromDomain| domain domain env))
```

—————

5.4.8 defun augModemapsFromDomain

[member p??]
[kar p??]
[getDomainsInScope p234]
[getdatabase p??]
[opOf p??]
[addNewDomain p236]
[listOrVectorElementNode p??]
[stripUnionTags p??]
[augModemapsFromDomain1 p237]
[\$Category p??]
[\$DummyFunctorNames p??]

— defun augModemapsFromDomain —

```
(defun |augModemapsFromDomain| (name functorForm env)
  (let (curDomainsInScope u innerDom)
    (declare (special |$Category| |$DummyFunctorNames|))
    (cond
      ((|member| (or (kar name) name) |$DummyFunctorNames|)
       env)
      ((or (equal name |$Category|) (|isCategoryForm| name env))
       env)
      ((|member| name (setq curDomainsInScope (|getDomainsInScope| env)))
       env)
      (t
       (when (setq u (getdatabase (|opOf| functorForm) 'superdomain))
         (setq env (|addNewDomain| (car u) env)))
       (when (setq innerDom (|listOrVectorElementNode| name))
         (setq env (|addDomain| innerDom env)))
       (when (and (consp name) (eq (qfirst name) '|Union|))
         (dolist (d (|stripUnionTags| (qrest name)))
           (setq env (|addDomain| d env))))
       (|augModemapsFromDomain1| name functorForm env))))))
```

5.4.9 defun augModemapsFromDomain1

```
[getl p??]
[kar p??]
[addConstructorModemaps p238]
[getmode p??]
[augModemapsFromCategory p245]
[getmodeOrMapping p??]
[substituteCategoryArguments p237]
[stackMessage p??]
```

— defun augModemapsFromDomain1 —

```
(defun |augModemapsFromDomain1| (name functorForm env)
  (let (mappingForm categoryForm functArgTypes catform)
    (cond
      ((getl (kar functorForm) '|makeFunctionList|)
        (|addConstructorModemaps| name functorForm env))
      ((and (atom functorForm) (setq catform (|getmode| functorForm env)))
        (|augModemapsFromCategory| name functorForm catform env))
      ((setq mappingForm (|getmodeOrMapping| (kar functorForm) env))
        (when (eq (car mappingForm) '|Mapping|) (car mappingForm))
        (setq categoryForm (cadr mappingForm))
        (setq functArgTypes (caddr mappingForm))
        (setq catform
          (|substituteCategoryArguments| (cdr functorForm) categoryForm))
        (|augModemapsFromCategory| name functorForm catform env))
      (t
        (|stackMessage| (list functorForm '| is an unknown mode|))
        env))))
```

5.4.10 defun substituteCategoryArguments

```
[msubst p??]
[internl p??]
[stringimage p??]
[sublis p??]
```

— defun substituteCategoryArguments —

```
(defun |substituteCategoryArguments| (argl catform)
```

```
(let (arglAssoc (i 0))
  (setq argl (msubst '$$ '$ argl))
  (setq arglAssoc
    (loop for a in argl
      collect (cons (internl '|#| (stringimage (incf i))) a)))
  (sublis arglAssoc catform)))
```

5.4.11 defun addConstructorModemaps

```
[putDomainsInScope p235]
[getl p??]
[msubst p??]
[qcar p??]
[qcdr p??]
[addModemap p253]
[$InteractiveMode p??]
```

— defun addConstructorModemaps —

```
(defun |addConstructorModemaps| (name form env)
  (let (|$InteractiveMode| functorName fn tmp1 funList op sig nsig opcode)
    (declare (special |$InteractiveMode|))
    (setq functorName (car form))
    (setq |$InteractiveMode| nil)
    (setq env (|putDomainsInScope| name env))
    (setq fn (getl functorName '|makeFunctionList|))
    (setq tmp1 (funcall fn name form env))
    (setq funList (car tmp1))
    (setq env (cadr tmp1))
    (dolist (item funList)
      (setq op (first item))
      (setq sig (second item))
      (setq opcode (third item))
      (when (and (consp opcode) (consp (qrest opcode))
                (consp (qcddr opcode))
                (eq (qcdddr opcode) nil)
                (eq (qfirst opcode) 'elt))
        (setq nsig (msubst '$$$ name sig))
        (setq nsig (msubst '$ '$$$ (msubst '$$ '$ nsig)))
        (setq opcode (list (first opcode) (second opcode) nsig)))
      (setq env (|addModemap| op name sig t opcode env)))
    env))
```

5.4.12 defun getModemap

```
[get p??]
[compApplyModemap p239]
[sublis p??]
```

— defun getModemap —

```
(defun |getModemap| (x env)
  (let (u)
    (dolist (modemap (|get| (first x) '|modemap| env))
      (when (setq u (|compApplyModemap| x modemap env nil))
        (return (sublis (third u) modemap))))))
```

—————

5.4.13 defun compApplyModemap

```
[length p??]
[pmatchWithSl p??]
[sublis p??]
[comp p557]
[coerce p346]
[compMapCond p240]
[member p??]
[genDeltaEntry p??]
[$e p??]
[$bindings p??]
[$e p??]
[$bindings p??]
```

— defun compApplyModemap —

```
(defun |compApplyModemap| (form modemap |$e| sl)
  (declare (special |$e|))
  (let (op argl mc mr margl fnsl g mp lt ltp temp1 f)
    (declare (special |$bindings| |$e|))
    ; -- $e      is the current environment
    ; -- sl      substitution list, nil means bottom-up, otherwise top-down
    ; -- 0.      fail immediately if #argl=#margl
    (setq op (car form))
    (setq argl (cdr form))
    (setq mc (caar modemap))
    (setq mr (cadar modemap))
    (setq margl (cddar modemap))
    (setq fnsl (cdr modemap))
```

```

(when (= (|#| argl) (|#| margl))
; 1. use modemap to evaluate arguments, returning failed if not possible
(setq lt
  (prog (t0)
    (return
      (do ((t1 argl (cdr t1)) (y NIL) (t2 margl (cdr t2)) (m nil))
        ((or (atom t1) (atom t2)) (nreverse0 t0))
        (setq y (car t1))
        (setq m (car t2))
        (setq t0
          (cons
            (progn
              (setq sl (|pmatchWithSl| mp m sl))
              (setq g (sublis sl m))
              (setq temp1 (or (|comp| y g |$e|) (return '|failed|)))
              (setq mp (cadr temp1))
              (setq |$e| (caddr temp1))
              temp1)
            t0))))))
; 2. coerce each argument to final domain, returning failed
; if not possible
(unless (eq lt '|failed|)
  (setq ltp
    (loop for y in lt for d in (sublis sl margl)
      collect (or (|coerce| y d) (return '|failed|))))
  (unless (eq ltp '|failed|)
    ; 3. obtain domain-specific function, if possible, and return
    ; $bindings is bound by compMapCond
    (setq temp1 (|compMapCond| op mc sl fnsl))
    (when temp1
      ; can no longer trust what the modemap says for a reference into
      ; an exterior domain (it is calculating the displacement based on view
      ; information which is no longer valid; thus ignore this index and
      ; store the signature instead.
      (setq f (car temp1))
      (setq |$bindings| (cadr temp1))
      (if (and (consp f) (consp (qcdr f)) (consp (qcddr f)) ; f is [op1..]
        (eq (qcdddr f) nil)
        (|member| (qcar f) '(elt const |Subsumed|)))
        (list (|genDeltaEntry| (cons op modemap)) ltp |$bindings|)
        (list f ltp |$bindings|))))))

```

5.4.14 defun compMapCond

[compMapCond' p241]
 [\$bindings p??]

— **defun compMapCond** —

```
(defun |compMapCond| (op mc |$bindings| fnsel)
  (declare (special |$bindings|))
  (let (t0)
    (do ((t1 nil t0) (t2 fnsel (cdr t2)) (u nil))
      ((or t1 (atom t2) (progn (setq u (car t2)) nil)) t0)
      (setq t0 (or t0 (|compMapCond'| u op mc |$bindings|))))))
```

—

5.4.15 **defun compMapCond'**

```
[compMapCond" p241]
[compMapCondFun p??]
[stackMessage p??]
```

— **defun compMapCond'** —

```
(defun |compMapCond'| (t0 op dc bindings)
  (let ((cexpr (car t0)) (fnexpr (cadr t0)))
    (if (|compMapCond''| cexpr dc)
        (|compMapCondFun| fnexpr op dc bindings)
        (|stackMessage| '("not known that" %b ,dc %d "has" %b ,cexpr %d)))))
```

—

5.4.16 **defun compMapCond"**

```
[compMapCond" p241]
[knownInfo p??]
[get p??]
[stackMessage p??]
[$Information p??]
[$e p??]
```

— **defun compMapCond"** —

```
(defun |compMapCond''| (cexpr dc)
  (let (l u tmp1 tmp2)
    (declare (special |$Information| |$e|))
    (cond
      ((eq cexpr t) t)
```

```

((and (consp cexpr)
      (eq (qcar cexpr) 'and)
      (progn (setq l (qcdr cexpr)) t))
 (prog (t0)
       (setq t0 t)
       (return
        (do ((t1 nil (null t0)) (t2 l (cdr t2)) (u nil))
            ((or t1 (atom t2) (progn (setq u (car t2)) nil)) t0)
            (setq t0 (and t0 (|compMapCond''| u dc)))))))
((and (consp cexpr)
      (eq (qcar cexpr) 'or)
      (progn (setq l (qcdr cexpr)) t))
 (prog (t3)
       (setq t3 nil)
       (return
        (do ((t4 nil t3) (t5 l (cdr t5)) (u nil))
            ((or t4 (atom t5) (progn (setq u (car t5)) nil)) t3)
            (setq t3 (or t3 (|compMapCond''| u dc)))))))
((and (consp cexpr)
      (eq (qcar cexpr) '|not|)
      (progn
        (setq tmp1 (qcdr cexpr))
        (and (consp tmp1)
              (eq (qcdr tmp1) nil)
              (progn (setq u (qcar tmp1)) t))))
 (null (|compMapCond''| u dc)))
((and (consp cexpr)
      (eq (qcar cexpr) '|has|)
      (progn
        (setq tmp1 (qcdr cexpr))
        (and (consp tmp1)
              (progn
                (setq tmp2 (qcdr tmp1))
                (and (consp tmp2)
                      (eq (qcdr tmp2) nil)))))))
 (cond
  ((|knownInfo| cexpr) t)
  (t nil)))
(|member|
 (cons 'attribute (cons dc (cons cexpr nil)))
 (|get| '|$Information| 'special |$e|))
 t)
(t
 (|stackMessage| '("not known that" %b ,dc %d "has" %b ,cexpr %d))
 nil)))

```

5.4.17 defun compMapCondFun

— defun compMapCondFun —

```
(defun |compMapCondFun| (fnexpr op dc bindings)
  (declare (ignore op) (ignore dc))
  (cons fnexpr (cons bindings nil)))
```

5.4.18 defun getUniqueSignature

[getUniqueModemap p243]

— defun getUniqueSignature —

```
(defun |getUniqueSignature| (form env)
  (cdar (|getUniqueModemap| (first form) (|#| (rest form)) env)))
```

5.4.19 defun getUniqueModemap

[getModemapList p244]

[qslessp p??]

[stackWarning p??]

— defun getUniqueModemap —

```
(defun |getUniqueModemap| (op numOfArgs env)
  (let (mml)
    (cond
      ((eq 1 (|#| (setq mml (|getModemapList| op numOfArgs env))))
       (car mml))
      ((qslessp 1 (|#| mml))
       (|stackWarning|
        (list numOfArgs " argument form of: " op " has more than one modemap"))
       (car mml))
      (t nil))))
```

5.4.20 defun getModemapList

```
[qcar p??]
[qcdr p??]
[getModemapListFromDomain p244]
[nreverse0 p??]
[get p??]
```

— defun getModemapList —

```
(defun |getModemapList| (op numOfArgs env)
  (let (result)
    (cond
      ((and (consp op) (eq (qfirst op) '|elt|) (consp (qrest op))
        (consp (qcddr op)) (eq (qcdddr op) nil))
       (|getModemapListFromDomain| (third op) numOfArgs (second op) env))
      (t
       (dolist (term (|get| op '|modemap| env) (nreverse0 result))
         (when (eql numOfArgs (|#| (cddar term))) (push term result)))))))
```

—————

5.4.21 defun getModemapListFromDomain

```
[get p??]
```

— defun getModemapListFromDomain —

```
(defun |getModemapListFromDomain| (op numOfArgs d env)
  (loop for term in (|get| op '|modemap| env)
        when (and (equal (caar term) d) (eql (|#| (cddar term)) numOfArgs))
        collect term))
```

—————

5.4.22 defun domainMember

```
[modeEqual p357]
```

— defun domainMember —

```
(defun |domainMember| (dom domList)
  (let (result)
    (dolist (d domList result)
```

```
(setq result (or result (|modeEqual| dom d))))))
```

5.4.23 defun augModemapsFromCategory

```
[evalAndSub p249]
[compilerMessage p??]
[putDomainsInScope p235]
[addModemapKnown p253]
[$base p??]
```

— defun augModemapsFromCategory —

```
(defun |augModemapsFromCategory| (domainName functorform categoryForm env)
  (let (tmp1 op sig cond fnsl)
    (declare (special |$base|))
    (setq tmp1 (|evalAndSub| domainName domainName functorform categoryForm env))
    (|compilerMessage| (list '|Adding| domainName '| modemaps|))
    (setq env (|putDomainsInScope| domainName (second tmp1)))
    (setq |$base| 4)
    (dolist (u (first tmp1))
      (setq op (caar u))
      (setq sig (cadar u))
      (setq cond (cadr u))
      (setq fnsl (caddr u))
      (setq env (|addModemapKnown| op domainName sig cond fnsl env)))
    env))
```

5.4.24 defun addEltModemap

This is a hack to change selectors from strings to identifiers; and to add flag identifiers as literals in the environment [qcar p??]

```
[qcdr p??]
[makeLiteral p??]
[addModemap1 p254]
[systemErrorHere p??]
[$insideCapsuleFunctionIfTrue p??]
[$e p??]
```

— defun addEltModemap —

```

(defun |addEltModemap| (op mc sig pred fn env)
  (let (tmp1 v sel lt id)
    (declare (special |$e| |$insideCapsuleFunctionIfTrue|))
    (cond
      ((and (eq op '|elt|) (consp sig))
        (setq tmp1 (reverse sig))
        (setq sel (qfirst tmp1))
        (setq lt (nreverse (qrest tmp1)))
        (cond
          ((stringp sel)
            (setq id (intern sel))
            (if |$insideCapsuleFunctionIfTrue|
              (setq |$e| (|makeLiteral| id |$e|))
              (setq env (|makeLiteral| id env)))
            (|addModemap1| op mc (append lt (list id)) pred fn env))
          (t (|addModemap1| op mc sig pred fn env))))
      ((and (eq op '|setelt|) (consp sig))
        (setq tmp1 (reverse sig))
        (setq v (qfirst tmp1))
        (setq sel (qsecond tmp1))
        (setq lt (nreverse (qcddr tmp1)))
        (cond
          ((stringp sel) (setq id (intern sel))
            (if |$insideCapsuleFunctionIfTrue|
              (setq |$e| (|makeLiteral| id |$e|))
              (setq env (|makeLiteral| id env)))
            (|addModemap1| op mc (append lt (list id v)) pred fn env))
          (t (|addModemap1| op mc sig pred fn env))))
      (t (|systemErrorHere| "addEltModemap")))))

```

5.4.25 defun mkNewModemapList

```

[member p??]
[assoc p??]
[qcar p??]
[qcdr p??]
[mergeModemap p248]
[nequal p??]
[nreverse0 p??]
[insertModemap p247]
[$InteractiveMode p??]
[$forceAdd p??]

```

— defun mkNewModemapList —

```

(defun |mkNewModemapList| (mc sig pred fn curModemapList env filenameOrNil)
  (let (map entry oldMap opred result)
    (declare (special |$InteractiveMode| |$forceAdd|))
    (setq entry
      (cons (setq map (cons mc sig)) (cons (list pred fn) filenameOrNil)))
    (cond
      ((|member| entry curModemapList) curModemapList)
      ((and (setq oldMap (|assoc| map curModemapList))
            (consp oldMap) (consp (qrest oldMap))
            (consp (qsecond oldMap))
            (consp (qcadadr oldMap))
            (eq (qcddadr oldMap) nil)
            (equal (qcadadr oldMap) fn))
        (setq opred (qcaadr oldMap))
        (cond
          (|$forceAdd| (|mergeModemap| entry curModemapList env))
          ((eq opred t) curModemapList)
          (t
           (when (and (nequal pred t) (nequal pred opred))
             (setq pred (list 'or pred opred)))
           (dolist (x curModemapList (nreverse0 result))
             (push
              (if (equal x oldMap)
                  (cons map (cons (list pred fn) filenameOrNil))
                  x)
              result))))))
      (|$InteractiveMode|
       (|insertModemap| entry curModemapList))
      (t
       (|mergeModemap| entry curModemapList env))))

```

5.4.26 defun insertModemap

— defun insertModemap —

```

(defun |insertModemap| (new mmList)
  (if (null mmList) (list new) (cons new mmList)))

```

5.4.27 defun mergeModemap

```
[isSuperDomain p235]
[TruthP p249]
[$forceAdd p??]
```

— defun mergeModemap —

```
(defun |mergeModemap| (entry modemapList env)
  (let (mc sig pred mcp sigp predp newmm mm)
    (declare (special |$forceAdd|))
    ; break out the condition, signature, and predicate fields of the new entry
    (setq mc (caar entry))
    (setq sig (cdar entry))
    (setq pred (caadr entry))
    (seq
     ; walk across the successive tails of the modemap list
     (do ((mmtail modemapList (cdr mmtail)))
         ((atom mmtail) nil)
         (setq mcp (caaar mmtail))
         (setq sigp (cdaar mmtail))
         (setq predp (caadar mmtail)))
     (cond
      ((or (equal mc mcp) (|isSuperDomain| mcp mc env))
       ; if this is a duplicate condition
       (exit
        (progn
         (setq newmm nil)
         (setq mm modemapList)
         ; copy the unique modemap terms
         (loop while (not (eq mm mmtail)) do
          (setq newmm (cons (car mm) newmm))
          (setq mm (cdr mm)))
         ; if the conditions and signatures are equal
         (when (and (equal mc mcp) (equal sig sigp))
          ; we only need one of these unless the conditions are hairy
          (cond
           ((and (null |$forceAdd|) (|TruthP| predp))
            ; the new predicate buys us nothing
            (setq entry nil)
            (return modemapList))
           ((|TruthP| pred)
            ; the thing we matched against is useless, by comparison
            (setq mmtail (cdr mmtail))))))
         (setq modemapList (nconc (nreverse newmm) (cons entry mmtail)))
         (setq entry nil)
         (return modemapList))))))
    ; if the entry is still defined, add it to the modemap
    (if entry
```

```
(append modemapList (list entry))
modemapList)))
```

5.4.28 defun TruthP

```
[qcar p??]
```

— defun TruthP —

```
(defun |TruthP| (x)
  (cond
    ((null x) nil)
    ((eq x t) t)
    ((and (consp x) (eq (qfirst x) 'quote)) t)
    (t nil)))
```

5.4.29 defun evalAndSub

```
[isCategory p??]
[substNames p251]
[contained p??]
[put p??]
[get p??]
[getOperationAlist p250]
[$lhsOfColon p??]
```

— defun evalAndSub —

```
(defun |evalAndSub| (domainName viewName functorForm form |$e|)
  (declare (special |$e|))
  (let (|$lhsOfColon| opAlist substAlist)
    (declare (special |$lhsOfColon|))
    (setq |$lhsOfColon| domainName)
    (cond
      ((|isCategory| form)
       (list (|substNames| domainName viewName functorForm (elt form 1)) |$e|))
      (t
       (when (contained '$$ form)
         (setq |$e| (|put| '$$ '|mode| (|get| '$ '|mode| |$e|) |$e|)))
       (setq opAlist (|getOperationAlist| domainName functorForm form))
       (setq substAlist (|substNames| domainName viewName functorForm opAlist))
```

```
(list substAlist |$e|))))))
```

5.4.30 defun getOperationAlist

```
[getdatabase p??]
[isFunctor p233]
[systemError p??]
[compMakeCategoryObject p182]
[stackMessage p??]
[$e p??]
[$domainShell p??]
[$insideFunctorIfTrue p??]
[$functorForm p??]
```

— defun getOperationAlist —

```
(defun |getOperationAlist| (name functorForm form)
  (let (u tt)
    (declare (special |$e| |$domainShell| |$insideFunctorIfTrue| |$functorForm|))
    (when (and (atom name) (getdatabase name 'niladic))
      (setq functorform (list functorForm)))
    (cond
      ((and (setq u (|isFunctor| functorForm))
            (null (and |$insideFunctorIfTrue|
                      (equal (first functorForm) (first |$functorForm|)))))
       u)
      ((and |$insideFunctorIfTrue| (eq name '$))
       (if |$domainShell|
          (elt |$domainShell| 1)
          (|systemError| "$ has no shell now")))
      ((setq tt (|compMakeCategoryObject| form |$e|))
       (setq |$e| (third tt))
       (elt (first tt) 1))
      (t
       (|stackMessage| (list '|not a category form: | form|))))))
```

5.4.31 defvar \$FormalMapVariableList

— initvars —

```
(defvar |$FormalMapVariableList|
  '(\#1 \#2 \#3 \#4 \#5 \#6 \#7 \#8 \#9 \#10 \#11 \#12 \#13 \#14 \#15))
```

5.4.32 defun substNames

```
[substq p??]
[isCategoryPackageName p190]
[eqsubstlist p??]
[nreverse0 p??]
[$FormalMapVariableList p250]
```

— defun substNames —

```
(defun |substNames| (domainName viewName functorForm opalist)
  (let (nameForDollar sel pos modemapform tmp0 tmp1)
    (declare (special |$FormalMapVariableList|))
    (setq functorForm (substq '$$ '$ functorForm))
    (setq nameForDollar
      (if (|isCategoryPackageName| functorForm)
          (second functorForm)
          domainName))
    ; following calls to SUBSTQ must copy to save RPLAC's in
    ; putInLocalDomainReferences
    (dolist (term
              (eqsubstlist (kdr functorForm) |$FormalMapVariableList| opalist)
              (nreverse0 tmp0)))
      (setq tmp1 (reverse term))
      (setq sel (caar tmp1))
      (setq pos (caddar tmp1))
      (setq modemapform (nreverse (cdr tmp1)))
      (push
        (append
          (substq '$ '$$ (substq nameForDollar '$ modemapform))
          (list
            (list sel viewName (if (eq domainName '$) pos (cadar modemapform))))))
        tmp0))))
```

5.4.33 defun augModemapsFromCategoryRep

```
[evalAndSub p249]
[isCategory p??]
```

```
[compilerMessage p??]
[putDomainsInScope p235]
[assoc p??]
[msubst p??]
[addModemap p253]
[$base p??]
```

— **defun augModemapsFromCategoryRep** —

```
(defun |augModemapsFromCategoryRep|
  (domainName repDefn functorBody categoryForm env)
  (labels (
    (redefinedList (op z)
      (let (result)
        (dolist (u z result)
          (setq result (or result (redefined op u))))))
    (redefined (opname u)
      (let (op z result)
        (when (consp u)
          (setq op (qfirst u))
          (setq z (qrest u))
          (cond
            ((eq op 'def) (equal opname (caar z)))
            ((member op '(progn seq)) (redefinedList opname z))
            ((eq op 'cond)
              (dolist (v z result)
                (setq result (or result (redefinedList opname (cdr v))))))))))
    (let (fnAlist tmp1 repFnAlist catform lhs op sig cond fnsel u)
      (declare (special |$base|))
      (setq tmp1 (|evalAndSub| domainName domainName domainName categoryForm env))
      (setq fnAlist (car tmp1))
      (setq env (cadr tmp1))
      (setq tmp1 (|evalAndSub| '|Rep| '|Rep| repDefn (|getmode| repDefn env) env))
      (setq repFnAlist (car tmp1))
      (setq env (cadr tmp1))
      (setq catform
        (if (|isCategory| categoryForm) (elt categoryForm 0) categoryForm))
      (|compilerMessage| (list '|Adding| domainName '| modemaps|))
      (setq env (|putDomainsInScope| domainName env))
      (setq |$base| 4)
      (dolist (term fnAlist)
        (setq lhs (car term))
        (setq op (caar term))
        (setq sig (cadar term))
        (setq cond (cadr term))
        (setq fnsel (caddr term))
        (setq u (|assoc| (msubst '|Rep| domainName lhs) repFnAlist))
        (if (and u (null (redefinedList op functorBody)))
          (setq env (|addModemap| op domainName sig cond (caddr u) env))
```

```
(setq env (|addModemap| op domainName sig cond fn sel env))))
env)))
```

5.5 Maintaining Modemaps

5.5.1 defun addModemapKnown

```
[addModemap0 p254]
[$e p??]
[$insideCapsuleFunctionIfTrue p??]
[$CapsuleModemapFrame p??]
```

— defun addModemapKnown —

```
(defun |addModemapKnown| (op mc sig pred fn |$e|)
  (declare (special |$e| |$CapsuleModemapFrame| |$insideCapsuleFunctionIfTrue|))
  (if (eq |$insideCapsuleFunctionIfTrue| t)
      (progn
        (setq |$CapsuleModemapFrame|
              (|addModemap0| op mc sig pred fn |$CapsuleModemapFrame|)
              |$e|)
        (|addModemap0| op mc sig pred fn |$e|)))
```

5.5.2 defun addModemap

```
[addModemap0 p254]
[knownInfo p??]
[$e p??]
[$InteractiveMode p??]
[$insideCapsuleFunctionIfTrue p??]
[$CapsuleModemapFrame p??]
[$CapsuleModemapFrame p??]
```

— defun addModemap —

```
(defun |addModemap| (op mc sig pred fn |$e|)
  (declare (special |$e| |$CapsuleModemapFrame| |$InteractiveMode|
                    |$insideCapsuleFunctionIfTrue|))
  (cond
```

```

(|$InteractiveModel| |$e|)
(t
  (when (|knownInfo| pred) (setq pred t))
  (cond
    ((eq |$insideCapsuleFunctionIfTrue| t)
     (setq |$CapsuleModemapFrame|
      (|addModemap0| op mc sig pred fn |$CapsuleModemapFrame|))
     |$e|)
    (t
     (|addModemap0| op mc sig pred fn |$e|))))))

```

5.5.3 defun addModemap0

```

[qcar p??]
[addEltModemap p245]
[addModemap1 p254]
[$functorForm p??]

```

— defun addModemap0 —

```

(defun |addModemap0| (op mc sig pred fn env)
  (declare (special |$functorForm|))
  (cond
    ((and (consp |$functorForm|)
          (eq (qfirst |$functorForm|) '|CategoryDefaults|)
          (eq mc '$))
     env)
    ((or (eq op '|elt|) (eq op '|setelt|))
     (|addEltModemap| op mc sig pred fn env))
    (t (|addModemap1| op mc sig pred fn env))))

```

5.5.4 defun addModemap1

```

[msubst p??]
[getProplist p??]
[mkNewModemapList p246]
[lassoc p??]
[augProplist p??]
[unErrorRef p??]
[addBinding p??]

```

— defun addModemap1 —

```
(defun |addModemap1| (op mc sig pred fn env)
  (let (currentProplist newModemapList newProplist newProplistp)
    (when (eq mc '|Rep|) (setq sig (msubst '$ '|Rep| sig)))
    (setq currentProplist (or (|getProplist| op env) nil))
    (setq newModemapList
      (|mkNewModemapList| mc sig pred fn
        (lassoc '|modemap| currentProplist) env nil))
    (setq newProplist (|augProplist| currentProplist '|modemap| newModemapList))
    (setq newProplistp (|augProplist| newProplist 'fluid t))
    (|unErrorRef| op)
    (|addBinding| op newProplistp env)))
```

—————

5.6 Indirect called comp routines

In the **compExpression** function there is the code:

```
(if (and (atom (car x)) (setq fn (get1 (car x) 'special)))
  (funcall fn x m e)
  (|compForm| x m e)))
```

5.6.1 defplist compAdd plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|add| 'special) 'compAdd))
```

—————

5.6.2 defun compAdd

The **compAdd** function expects three arguments:

1. the **form** which is an —add— specifying the domain to extend and a set of functions to be added
2. the **mode** a —Join—, which is a set of categories and domains

3. the **env** which is a list of functions and their modemaps

The bulk of the work is performed by a call to `compOrCroak` which compiles the functions in the add form capsule.

The `compAdd` function returns a triple, the result of a call to `compCapsule`.

1. the **compiled capsule** which is a progn form which returns the domain
2. the **mode** from the input argument
3. the **env** prepended with the signatures of the functions in the body of the add.

```
[comp p557]
[qcdr p??]
[qcar p??]
[compSubDomain1 p341]
[nreverse0 p??]
[NRTgetLocalIndex p192]
[compTuple2Record p258]
[compOrCroak p556]
[compCapsule p258]
[/editfile p??]
[$addForm p??]
[$addFormLhs p??]
[$EmptyMode p131]
[$NRTaddForm p??]
[$packagesUsed p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— **defun compAdd** —

```
(defun compAdd (form mode env)
  (let (|$addForm| |$addFormLhs| code domainForm predicate tmp3 tmp4)
    (declare (special |$addForm| |$addFormLhs| |$EmptyMode| |$NRTaddForm|
                      |$packagesUsed| |$functorForm| |$bootStrapMode| /editfile))
    (setq |$addForm| (second form))
    (cond
      ((eq |$bootStrapMode| t)
       (cond
         ((and (consp |$addForm|) (eq (qfirst |$addForm|) '@Tuple)))
         (setq code nil))
        (t
         (setq tmp3 (|comp| |$addForm| mode env))
         (setq code (first tmp3))
         (setq mode (second tmp3))
         (setq env (third tmp3)) tmp3))
      (list
```

```

(list 'cond
  (list '$bootStrapMode| code)
  (list 't
    (list '|systemError|
      (list 'list '|%b| (mkq (car |$functorForm|)) '|%d| "from"
        '|%b| (mkq (|namestring| /editfile)) '|%d|
          "needs to be compiled"))))
    mode env))
(t
  (setq |$addFormLhs| |$addForm|)
  (cond
    ((and (consp |$addForm|) (eq (qfirst |$addForm|) '|SubDomain|)
      (consp (qrest |$addForm|)) (consp (qcddr |$addForm|))
      (eq (qcdddr |$addForm|) nil))
      (setq domainForm (second |$addForm|))
      (setq predicate (third |$addForm|))
      (setq |$packagesUsed| (cons domainForm |$packagesUsed|))
      (setq |$NRTaddForm| domainForm)
      (|NRTgetLocalIndex| domainForm)
      ; need to generate slot for add form since all $ go-get
      ; slots will need to access it
      (setq tmp3 (|compSubDomain1| domainForm predicate mode env))
      (setq |$addForm| (first tmp3))
      (setq env (third tmp3)) tmp3)
    (t
      (setq |$packagesUsed|
        (if (and (consp |$addForm|) (eq (qfirst |$addForm|) '|@Tuple|))
          (append (qrest |$addForm|) |$packagesUsed|)
          (cons |$addForm| |$packagesUsed|)))
        (setq |$NRTaddForm| |$addForm|)
        (setq tmp3
          (cond
            ((and (consp |$addForm|) (eq (qfirst |$addForm|) '|@Tuple|))
              (setq |$NRTaddForm|
                (cons '|@Tuple|
                  (dolist (x (cdr |$addForm|)) (nreverse0 tmp4))
                  (push (|NRTgetLocalIndex| x) tmp4))))
              (|compOrCroak| (|compTuple2Record| |$addForm|) |$EmptyMode| env))
            (t
              (|compOrCroak| |$addForm| |$EmptyMode| env))))
          (setq |$addForm| (first tmp3))
          (setq env (third tmp3))
          tmp3))
    (|compCapsule| (third form) mode env))))

```

5.6.3 defun compTuple2Record

— defun compTuple2Record —

```
(defun |compTuple2Record| (u)
  (let ((i 0))
    (cons '|Record|
      (loop for x in (rest u)
        collect (list '|:| (incf i) x))))))
```

5.6.4 defplist compCapsule plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'capsule 'special) '|compCapsule|))
```

5.6.5 defun compCapsule

```
[bootStrapError p196]
[compCapsuleInner p259]
[addDomain p232]
[editfile p??]
[$insideExpressionIfTrue p??]
[$functorForm p??]
[$bootStrapMode p??]
```

— defun compCapsule —

```
(defun |compCapsule| (form mode env)
  (let (|$insideExpressionIfTrue| itemList)
    (declare (special |$insideExpressionIfTrue| |$functorForm| /editfile
      |$bootStrapMode|))
    (setq itemList (cdr form))
    (cond
      ((eq |$bootStrapMode| t)
        (list (|bootStrapError| |$functorForm| /editfile) mode env))
      (t
```

```
(setq |$insideExpressionIfTrue| nil)
(|compCapsuleInner| itemList mode (|addDomain| '$ env))))))
```

5.6.6 defun compCapsuleInner

```
[addInformation p??]
[compCapsuleItems p260]
[processFunctor p259]
[mkpf p??]
[$getDomainCode p??]
[$signature p??]
[$form p??]
[$addForm p??]
[$insideCategoryPackageIfTrue p??]
[$insideCategoryIfTrue p??]
[$functorLocalParameters p??]
```

— defun compCapsuleInner —

```
(defun |compCapsuleInner| (form mode env)
  (let (localParList data code)
    (declare (special |$getDomainCode| |$signature| |$form| |$addForm|
                      |$insideCategoryPackageIfTrue| |$insideCategoryIfTrue|
                      |$functorLocalParameters|))
    (setq env (|addInformation| mode env))
    (setq data (cons 'progn form))
    (setq env (|compCapsuleItems| form nil env))
    (setq localParList |$functorLocalParameters|)
    (when |$addForm| (setq data (list '|add| |$addForm| data)))
    (setq code
      (if (and |$insideCategoryIfTrue| (null |$insideCategoryPackageIfTrue|))
          data
          (|processFunctor| |$form| |$signature| data localParList env)))
    (cons (mkpf (append |$getDomainCode| (list code))) 'progn) (list mode env))))
```

5.6.7 defun processFunctor

```
[error p??]
[buildFunctor p??]
```

— defun processFunctor —

```
(defun |processFunctor| (form signature data localParList e)
  (cond
    ((and (consp form) (eq (qrest form) nil)
      (eq (qfirst form) '|CategoryDefaults|))
      (|error| '|CategoryDefaults is a reserved name|))
    (t (|buildFunctor| form signature data localParList e))))
```

5.6.8 defun compCapsuleItems

The variable `data` appears to be unbound at runtime. Optimized code won't check for this but interpreted code fails. We should PROVE that `data` is unbound at runtime but have not done so yet. Rather than remove the code entirely (since there MIGHT be a path where it is used) we check for the runtime bound case and assign `$myFunctorBody` if `data` has a value.

The `compCapsuleInner` function in this file LOOKS like it sets `data` and expects code to manipulate the assigned `data` structure. Since we can't be sure we take the least disruptive course of action.

```
[compSingleCapsuleItem p261]
[$top-level p??]
[$myFunctorBody p??]
[$signatureOfForm p??]
[$suffix p??]
[$e p??]
[$pred p??]
[$e p??]
```

— defun compCapsuleItems —

```
(defun |compCapsuleItems| (itemlist |$predl| |$e|)
  (declare (special |$predl| |$e|))
  (let ($top_level |$myFunctorBody| |$signatureOfForm| |$suffix|)
    (declare (special $top_level |$myFunctorBody| |$signatureOfForm| |$suffix|))
    (setq $top_level nil)
    (setq |$myFunctorBody| nil)
    (when (boundp '|data|) (setq |$myFunctorBody| |data|))
    (setq |$signatureOfForm| nil)
    (setq |$suffix| 0)
    (loop for item in itemlist do
      (setq |$e| (|compSingleCapsuleItem| item |$predl| |$e|)))
    |$e|))
```

5.6.9 defun compSingleCapsuleItem

```
[doit p??]
[$pred p??]
[$e p??]
[macroExpandInPlace p136]
```

— **defun compSingleCapsuleItem** —

```
(defun |compSingleCapsuleItem| (item |$predl| |$e|)
  (declare (special |$predl| |$e|))
  (|doIt| (|macroExpandInPlace| item |$e|) |$predl|)
  |$e|)
```

5.6.10 defun doIt

```
[qcar p??]
[qcdr p??]
[lastnode p??]
[compSingleCapsuleItem p261]
[isDomainForm p338]
[stackWarning p??]
[doIt p261]
[compOrCroak p556]
[stackSemanticError p??]
[bright p??]
[member p??]
[kar p??]
[—isFunction p??]
[insert p??]
[opOf p??]
[get p??]
[NRTgetLocalIndex p192]
[sublis p??]
[compOrCroak p556]
[sayBrightly p??]
[formatUnabbreviated p??]
[doItIf p265]
[isMacro p267]
[put p??]
[cannotDo p??]
[$predl p??]
[$e p??]
```

```

[$EmptyMode p131]
[$NonMentionableDomainNames p??]
[$functorLocalParameters p??]
[$functorsUsed p??]
[$packagesUsed p??]
[$NRTopt p??]
[$Representation p??]
[$LocalDomainAlist p??]
[$QuickCode p??]
[$signatureOfForm p??]
[$genno p??]
[$e p??]
[$functorLocalParameters p??]
[$functorsUsed p??]
[$packagesUsed p??]
[$Representation p??]
[$LocalDomainAlist p??]

```

— defun doIt —

```

(defun |doIt| (item |$pred1|)
  (declare (special |$pred1|))
  (prog ($genno x rhs lhsp lhs rhsp rhsCode z tmp1 tmp2 tmp6 op body tt
        functionPart u code)
    (declare (special $genno |$e| |$EmptyMode| |$signatureOfForm|
                    |$QuickCode| |$LocalDomainAlist| |$Representation|
                    |$NRTopt| |$packagesUsed| |$functorsUsed|
                    |$functorLocalParameters| |$NonMentionableDomainNames|))
    (setq $genno 0)
    (cond
      ((and (consp item) (eq (qfirst item) 'seq) (consp (qrest item))
            (progn (setq tmp6 (reverse (qrest item))) t)
            (consp tmp6) (consp (qfirst tmp6))
            (eq (qcaar tmp6) '|exit|)
            (consp (qcдар tmp6))
            (equal (qcadar tmp6) 1)
            (consp (qcddar tmp6))
            (eq (qcdddar tmp6) nil))
        (setq x (qcaddar tmp6))
        (setq z (qrest tmp6))
        (setq z (nreverse z))
        (rplaca item 'progn)
        (rplaca (lastnode item) x)
        (loop for it1 in (rest item)
              do (setq |$e| (|compSingleCapsuleItem| it1 |$pred1| |$e|))))
      ((|isDomainForm| item |$e|)
        (setq u (list '|import| (cons (car item) (cdr item))))
        (|stackWarning| (list '|Use: import | (cons (car item) (cdr item)))))

```

```
(rplaca item (car u))
(rplacd item (cdr u))
(|doIt| item |$predl|))
(and (consp item) (eq (qfirst item) 'let) (consp (qrest item))
      (consp (qcddr item)))
(setq lhs (qsecond item))
(setq rhs (qthird item))
(cond
  ((null (progn
            (setq tmp2 (|compOrCroak| item |$EmptyMode| |$e|))
            (and (consp tmp2)
                  (progn
                    (setq code (qfirst tmp2))
                    (and (consp (qrest tmp2))
                          (progn
                            (and (consp (qcddr tmp2))
                                  (eq (qcdddr tmp2) nil)
                                  (PROGN
                                    (setq |$e| (qthird tmp2))
                                    t))))))))))
    (|stackSemanticError|
     (cons '|cannot compile assigned value to| (|bright| lhs))
     nil))
  ((null (and (consp code) (eq (qfirst code) 'let)
              (progn
                (and (consp (qrest code))
                      (progn
                        (setq lhsp (qsecond code))
                        (and (consp (qcddr code)))))
                    (atom (qsecond code))))))
    (cond
      ((and (consp code) (eq (qfirst code) 'progn))
       (|stackSemanticError|
        (list '|multiple assignment | item '| not allowed|)
        nil))
      (t
       (rplaca item (car code))
       (rplacd item (cdr code)))))
  (t
   (setq lhs lhsp)
   (cond
     ((and (null (|member| (kar rhs) |$NonMentionableDomainNames|))
            (null (member lhs |$functorLocalParameters|)))
      (setq |$functorLocalParameters|
            (append |$functorLocalParameters| (list lhs)))))
     (cond
       ((and (consp code) (eq (qfirst code) 'let)
              (progn
                (setq tmp2 (qrest code))
                (and (consp tmp2)
```

```

(progn
  (setq tmp6 (qrest tmp2))
  (and (consp tmp6)
    (progn
      (setq rhsp (qfirst tmp6))
      t))))
(|isDomainForm| rhsp |$e|))
(cond
  ((|isFunctor| rhsp)
    (setq |$functorsUsed| (|insert| (|opOf| rhsp) |$functorsUsed|))
    (setq |$packagesUsed| (|insert| (list (|opOf| rhsp))
      |$packagesUsed|))))
  (cond
    ((eq lhs '|Rep|)
      (setq |$Representation| (elt (|get| '|Rep| '|value| |$e|) 0))
      (cond
        ((eq |$NRTopt| t)
          (|NRTgetLocalIndex| |$Representation|)
          (t nil))))
      (setq |$LocalDomainAlist|
        (cons (cons lhs
          (sublis |$LocalDomainAlist| (elt (|get| lhs '|value| |$e|) 0)))
          |$LocalDomainAlist|))))
    (cond
      ((and (consp code) (eq (qfirst code) 'let))
        (rplaca item (if |$QuickCode| 'qsetrefv 'setelt))
        (setq rhsCode rhsp)
        (rplacd item (list '$ (|NRTgetLocalIndex| lhs) rhsCode)))
      (t
        (rplaca item (car code))
        (rplacd item (cdr code))))))
  ((and (consp item) (eq (qfirst item) '|:|') (consp (qrest item))
    (consp (qcddr item)) (eq (qcddr item) nil))
    (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
    (setq |$e| (caddr tmp1))
    tmp1)
  ((and (consp item) (eq (qfirst item) '|import|))
    (loop for dom in (qrest item)
      do (|sayBrightly| (cons " importing " (|formatUnabbreviated| dom))))
    (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
    (setq |$e| (caddr tmp1))
    (rplaca item 'progn)
    (rplacd item nil))
  ((and (consp item) (eq (qfirst item) 'if))
    (|doItIf| item |$predl| |$e|))
  ((and (consp item) (eq (qfirst item) '|where|') (consp (qrest item))
    (|compOrCroak| item |$EmptyMode| |$e|))
    (|compOrCroak| item |$EmptyMode| |$e|))
  ((and (consp item) (eq (qfirst item) 'mdef))
    (setq tmp1 (|compOrCroak| item |$EmptyMode| |$e|))
    (setq |$e| (caddr tmp1)) tmp1)

```

```

((and (consp item) (eq (qfirst item) 'def) (consp (qrest item))
      (consp (qsecond item)))
 (setq op (qcaadr item))
 (cond
  ((setq body (|isMacro| item |$e|))
   (setq |$e| (|put| op '|macro| body |$e|)))
  (t
   (setq tt (|compOrCroak| item |$EmptyModel| |$e|))
   (setq |$e| (caddr tt))
   (rplaca item '|CodeDefine|)
   (rplacd (cadr item) (list |$signatureOfForm|))
   (setq functionPart (list '|dispatchFunction| (car tt)))
   (rplaca (cddr item) functionPart)
   (rplacd (cddr item) nil))))
((setq u (|compOrCroak| item |$EmptyModel| |$e|))
 (setq code (car u))
 (setq |$e| (caddr u))
 (rplaca item (car code))
 (rplacd item (cdr code)))
(t (|cannotDo|))))

```

5.6.11 defun doItIf

```

[comp p557]
[userError p??]
[compSingleCapsuleItem p261]
[getSuccessEnvironment p308]
[localExtras p??]
[rplaca p??]
[rplacd p??]
[$e p??]
[$functorLocalParameters p??]
[$predl p??]
[$e p??]
[$functorLocalParameters p??]
[$getDomainCode p??]
[$Boolean p??]

```

— defun doItIf —

```

(defun |doItIf| (item |$predl| |$e|)
  (declare (special |$predl| |$e|))
  (labels (
    (localExtras (oldFLP)

```

```

(let (oldFLPp flp1 gv ans nils n)
(declare (special |$functorLocalParameters| |$getDomainCode|))
(unless (eq oldFLP |$functorLocalParameters|)
  (setq flp1 |$functorLocalParameters|)
  (setq oldFLPp oldFLP)
  (setq n 0)
  (loop while oldFLPp
    do
      (setq oldFLPp (cdr oldFLPp))
      (setq n (1+ n)))
  (setq nils (setq ans nil))
  (loop for u in flp1
    do
      (if (or (atom u)
              (let (result)
                (loop for v in |$getDomainCode|
                  do
                    (setq result (or result
                                      (and (consp v) (consp (qrest v))
                                           (equal (qsecond v) u))))
                result)))
          ; Now we have to add code to compile all the elements of
          ; functorLocalParameters that were added during the conditional compilation
          (setq nils (cons u nils))
          (progn
            (setq gv (gensym))
            (setq ans (cons (list 'let gv u) ans))
            (setq nils (CONS gv nils))))
          (setq n (1+ n)))
      (setq |$functorLocalParameters| (append oldFLP (nreverse nils))
            (nreverse ans))))))
(let (p x y olde tmp1 pp xp oldFLP yp)
(declare (special |$functorLocalParameters| |$Boolean|))
(setq p (second item))
(setq x (third item))
(setq y (fourth item))
(setq olde |$e|)
(setq tmp1
  (or (|comp| p |$Boolean| |$e|)
      (|userError| (list "not a Boolean:" p))))
(setq pp (first tmp1))
(setq |$e| (third tmp1))
(setq oldFLP |$functorLocalParameters|)
(unless (eq x '|noBranch|)
  (|compSingleCapsuleItem| x |$pred1| (|getSuccessEnvironment| p |$e|))
  (setq xp (localExtras oldFLP)))
(setq oldFLP |$functorLocalParameters|)
(unless (eq y '|noBranch|)
  (|compSingleCapsuleItem| y |$pred1| (|getInverseEnvironment| p olde))
  (setq yp (localExtras oldFLP)))

```

```
(rplaca item 'cond)
(rplacd item (list (cons pp (cons x xp)) (cons ''t (cons y yp))))))
```

5.6.12 defun isMacro

```
[qcar p??]
[qcdr p??]
[get p??]
```

— defun isMacro —

```
(defun |isMacro| (x env)
  (let (op args signature body)
    (when
      (and (consp x) (eq (qfirst x) 'def) (consp (qrest x))
           (consp (qsecond x)) (consp (qcddr x))
           (consp (qcdddr x))
           (consp (qcddddr x))
           (eq (qrest (qcddddr x)) nil))
      (setq op (qcaadr x))
      (setq args (qcdadr x))
      (setq signature (qthird x))
      (setq body (qfirst (qcddddr x)))
      (when
        (and (null (|get| op '|modemap| env))
              (null args)
              (null (|get| op '|mode| env))
              (consp signature)
              (eq (qrest signature) nil)
              (null (qfirst signature)))
          body))))
```

5.6.13 defplist compCase plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|case| 'special) '|compCase|))
```

5.6.14 defun compCase

Will the jerk who commented out these two functions please NOT do so again. These functions ARE needed, and case can NOT be done by modemap alone. The reason is that A case B requires to take A evaluated, but B unevaluated. Therefore a special function is required. You may have thought that you had tested this on “failed” etc., but “failed” evaluates to it’s own mode. Try it on x case \$ next time.

An angry JHD - August 15th., 1984 [addDomain p232]
 [compCase1 p268]
 [coerce p346]

— defun compCase —

```
(defun |compCase| (form mode env)
  (let (mp td)
    (setq mp (third form))
    (setq env (|addDomain| mp env))
    (when (setq td (|compCase1| (second form) mp env)) (|coerce| td mode))))
```

—————

5.6.15 defun compCase1

[comp p557]
 [getModemapList p244]
 [nreverse0 p??]
 [modeEqual p357]
 [\$Boolean p??]
 [\$EmptyMode p131]

— defun compCase1 —

```
(defun |compCase1| (form mode env)
  (let (xp mp ep map tmp3 tmp5 tmp6 u fn)
    (declare (special |$Boolean| |$EmptyMode|))
    (when (setq tmp3 (|comp| form |$EmptyMode| env))
      (setq xp (first tmp3))
      (setq mp (second tmp3))
      (setq ep (third tmp3))
      (when
        (setq u
          (dolist (modemap (|getModemapList| ' |case| 2 ep) (nreverse0 tmp5))
            (setq map (first modemap))
            (when
              (and (consp map) (consp (qrest map)) (consp (qcddr map))
                (consp (qcddr map))
```

```

      (eq (qcdddr map) nil)
      (|modeEqual| (fourth map) mode)
      (|modeEqual| (third map) mp))
      (push (second modemap) tmp5))))
  (when
    (setq fn
      (dolist (onepair u tmp6)
        (when (first onepair) (setq tmp6 (or tmp6 (second onepair))))))
      (list (list '|call| fn xp) |$Boolean| ep))))))

```

5.6.16 defplist compCat plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|Record| 'special) '|compCat|))

```

5.6.17 defplist compCat plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|Mapping| 'special) '|compCat|))

```

5.6.18 defplist compCat plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|Union| 'special) '|compCat|))

```

5.6.19 defun compCat

[getl p??]

— defun compCat —

```
(defun |compCat| (form mode env)
  (declare (ignore mode))
  (let (functorName fn tmp1 tmp2 funList op sig catForm)
    (setq functorName (first form))
    (when (setq fn (getl functorName '|makeFunctionList|))
      (setq tmp1 (funcall fn form form env))
      (setq funList (first tmp1))
      (setq env (second tmp1))
      (setq catForm
        (list '|Join| '(|SetCategory|)
          (cons 'category
            (cons '|domain|
              (dolist (item funList (nreverse0 tmp2))
                (setq op (first item))
                (setq sig (second item))
                (unless (eq op '=) (push (list 'signature op sig) tmp2)))))))
      (list form catForm env))))
```

—————

5.6.20 defplist compCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'category 'special) '|compCategory|))
```

—————

5.6.21 defun compCategory

```
[resolve p356]
[qcar p??]
[qcdr p??]
[compCategoryItem p271]
[mkExplicitCategoryFunction p273]
[systemErrorHere p??]
[$sigList p??]
```

```

[$atList p??]
[$top-level p??]
[$sigList p??]
[$atList p??]

```

— defun compCategory —

```

(defun |compCategory| (form mode env)
  (let ($top_level |$sigList| |$atList| domainOrPackage z rep)
    (declare (special $top_level |$sigList| |$atList|))
    (setq $top_level t)
    (cond
      ((and
        (equal (setq mode (|resolve| mode (list '|Category|)))
              (list '|Category|))
        (consp form)
        (eq (qfirst form) 'category)
        (consp (qrest form)))
        (setq domainOrPackage (second form))
        (setq z (qcddr form))
        (setq |$sigList| nil)
        (setq |$atList| nil)
        (dolist (x z) (|compCategoryItem| x nil))
        (setq rep
          (|mkExplicitCategoryFunction| domainOrPackage |$sigList| |$atList|))
        (list rep mode env))
      (t
        (|systemErrorHere| "compCategory")))))

```

5.6.22 defun compCategoryItem

```

[qcar p??]
[qcdr p??]
[compCategoryItem p271]
[mkpf p??]
[$sigList p??]
[$atList p??]

```

— defun compCategoryItem —

```

(defun |compCategoryItem| (x pred1)
  (let (p e a b c predlp pred y z op sig)
    (declare (special |$sigList| |$atList|))
    (cond
      ((null x) nil)

```

```

; 1. if x is a conditional expression, recurse; otherwise, form the predicate
((and (consp x) (eq (qfirst x) 'cond)
      (consp (qrest x)) (eq (qcddr x) nil)
      (consp (qsecond x))
      (consp (qcaddr x))
      (eq (qcddadr x) nil))
 (setq p (qcaadr x))
 (setq e (qcadadr x))
 (setq predlp (cons p predl))
 (cond
  ((and (consp e) (eq (qfirst e) 'progn))
   (setq z (qrest e))
   (dolist (y z) (|compCategoryItem| y predlp)))
  (t (|compCategoryItem| e predlp))))
((and (consp x) (eq (qfirst x) 'if) (consp (qrest x))
      (consp (qcddr x)) (consp (qcdddr x))
      (eq (qcddddr x) nil))
 (setq a (qsecond x))
 (setq b (qthird x))
 (setq c (qfourth x))
 (setq predlp (cons a predl))
 (unless (eq b '|noBranch|)
  (cond
   ((and (consp b) (eq (qfirst b) 'progn))
    (setq z (qrest b))
    (dolist (y z) (|compCategoryItem| y predlp)))
   (t (|compCategoryItem| b predlp))))
 (cond
  ((eq c '|noBranch|) nil)
  (t
   (setq predlp (cons (list '|not| a) predlp))
   (cond
    ((and (consp c) (eq (qfirst c) 'progn))
     (setq z (qrest c))
     (dolist (y z) (|compCategoryItem| y predlp)))
    (t (|compCategoryItem| c predlp))))))
 (t
  (setq pred (if predl (mkpf predl 'and) t))
  (cond
   ; 2. if attribute, push it and return
   ((and (consp x) (eq (qfirst x) 'attribute)
        (consp (qrest x)) (eq (qcddr x) nil))
    (setq y (qsecond x))
    (push (mkq (list y pred)) '|atList|))
   ; 3. it may be a list, with PROG as the CAR, and some information as the CDR
   ((and (consp x) (eq (qfirst x) 'progn))
    (setq z (qrest x))
    (dolist (u z) (|compCategoryItem| u predl)))
   (t
    ; 4. otherwise, x gives a signature for a single operator name or a list of

```

```

; names; if a list of names, recurse
  (cond ((eq (car x) 'signature) (car x)))
  (setq op (cadr x))
  (setq sig (caddr x))
  (cond
   ((null (atom op))
    (dolist (y op)
     (|compCategoryItem| (cons 'signature (cons y sig)) pred1)))
   (t
    ; 5. branch on a single type or a signature %with source and target
    (push (mkq (list (cdr x) pred)) |$sigList|))))))

```

5.6.23 defun mkExplicitCategoryFunction

```

[mkq p??]
[union p??]
[mustInstantiate p274]
[remdup p??]
[identp p??]
[nequal p??]
[wrapDomainSub p274]

```

— defun mkExplicitCategoryFunction —

```

(defun |mkExplicitCategoryFunction| (domainOrPackage sigList atList)
  (let (body sig parameters)
    (setq body
      (list '|mkCategory| (mkq domainOrPackage)
        (cons 'list (reverse sigList))
        (cons 'list (reverse atList))
        (mkq
          (let (result)
            (loop for item in sigList
              do
                (setq sig (car (cdaadr item)))
                (setq result
                  (|union| result
                    (loop for d in sig
                      when (|mustInstantiate| d)
                        collect d))))
            result)))
          nil)))
    (setq parameters
      (remdup
        (let (result)

```

```

(loop for item in sigList
  do
    (setq sig (car (cdaadr item)))
    (setq result
      (append result
        (loop for x in sig
          when (and (identp x) (nequal x '$))
            collect x))))
  result)))
(|wrapDomainSub| parameters body)))

```

5.6.24 defun mustInstantiate

```

[qcar p??]
[getl p??]
[$DummyFunctorNames p??]

```

— defun mustInstantiate —

```

(defun |mustInstantiate| (d)
  (declare (special |$DummyFunctorNames|))
  (and (consp d)
    (null (or (member (qfirst d) |$DummyFunctorNames|)
      (getl (qfirst d) '|makeFunctionList|)))))

```

5.6.25 defun wrapDomainSub

— defun wrapDomainSub —

```

(defun |wrapDomainSub| (parameters x)
  (list '|DomainSubstitutionMacro| parameters x))

```

5.6.26 defplist compColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|:| 'special) '|compColon|))
```

5.6.27 defun compColon

```
[compColonInside p564]
[assoc p??]
[getDomainsInScope p234]
[isDomainForm p338]
[compColon member (vol5)]
[addDomain p232]
[isDomainForm p338]
[isCategoryForm p??]
[unknownTypeError p233]
[compColon p275]
[eqsubstlist p??]
[take p??]
[length p??]
[nreverse0 p??]
[getmode p??]
[systemErrorHere p??]
[put p??]
[makeCategoryForm p278]
[genSomeVariable p??]
[$lhsOfColon p??]
[$noEnv p??]
[$insideFunctorIfTrue p??]
[$bootStrapMode p??]
[$FormalMapVariableList p250]
[$insideCategoryIfTrue p??]
[$insideExpressionIfTrue p??]
```

— defun compColon —

```
(defun |compColon| (form mode env)
  (let (|$lhsOfColon| argf argt tprime mprime r td op argl newTarget a
        signature tmp2 catform tmp3 g2 g5)
    (declare (special |$lhsOfColon| |$noEnv| |$insideFunctorIfTrue|
                      |$bootStrapMode| |$FormalMapVariableList|
                      |$insideCategoryIfTrue| |$insideExpressionIfTrue|))
    (setq argf (second form))
    (setq argt (third form))
    (if |$insideExpressionIfTrue|
```

```
(|compColonInside| argf mode env argt)
(progn
  (setq |$lhsOfColon| argf)
  (setq argt
    (cond
      ((and (atom argt)
            (setq tprime (|assoc| argt (|getDomainsInScope| env))))
        tprime)
      ((and (|isDomainForm| argt env) (null |$insideCategoryIfTrue|))
        (unless (|member| argt (|getDomainsInScope| env))
          (setq env (|addDomain| argt env)))
        argt)
      ((or (|isDomainForm| argt env) (|isCategoryForm| argt env))
        argt)
      ((and (consp argt) (eq (qfirst argt) '|Mapping|)
            (progn
              (setq tmp2 (qrest argt))
              (and (consp tmp2)
                    (progn
                     (setq mprime (qfirst tmp2))
                     (setq r (qrest tmp2))
                     t))))
        argt)
      (t
        (|unknownTypeError| argt
          argt)))
    )
  (cond
    ((eq (car argf) '|listof|)
      (dolist (x (cdr argf) td)
        (setq td (|compColon| (list '|:| x argt) mode env))
        (setq env (third td))))
    (t
      (setq env
        (cond
          ((and (consp argf)
                (progn
                 (setq op (qfirst argf))
                 (setq argl (qrest argf))
                 t)
                (null (and (consp argt) (eq (qfirst argt) '|Mapping|))))
            (setq newTarget
              (eqsubstlist (take (|#| argl) |$FormalMapVariableList|)
                (dolist (x argl (nreverse0 g2))
                  (setq g2
                    (cons
                     (cond
                       ((and (consp x) (eq (qfirst x) '|:|)
                         (progn
                          (setq tmp2 (qrest x))
                          (and (consp tmp2)

```

```
(progn
  (setq a (qfirst tmp2))
  (setq tmp3 (qrest tmp2))
  (and (consp tmp3)
        (eq (qrest tmp3) nil)
        (progn
          (setq mode (qfirst tmp3))
          t))))))
    a)
  (t x))
g2)))
argt))
(setq signature
  (cons '|Mapping|
    (cons newTarget
      (dolist (x argl (nreverse0 g5))
        (setq g5
          (cons
            (cond
              ((and (consp x) (eq (qfirst x) '|:|))
               (progn
                 (setq tmp2 (qrest x))
                 (and (consp tmp2)
                     (progn
                      (setq a (qfirst tmp2))
                      (setq tmp3 (qrest tmp2))
                      (and (consp tmp3)
                          (eq (qrest tmp3) nil)
                          (progn
                           (setq mode (qfirst tmp3))
                           t))))))
                mode)
              (t
               (or (|getmode| x env)
                   (|systemErrorHere| "compColonOld")))))
             g5))))))
    (|put| op '|mode| signature env))
  (t (|put| argf '|mode| argt env))))
(cond
  ((and (null |$bootStrapMode|) |$insideFunctorIfTrue|)
   (progn
     (setq tmp2 (|makeCategoryForm| argt env))
     (and (consp tmp2)
          (progn
            (setq catform (qfirst tmp2))
            (setq tmp3 (qrest tmp2))
            (and (consp tmp3)
                  (eq (qrest tmp3) nil)
                  (progn
                    (setq env (qfirst tmp3))
```

```

t))))))
(setq env
  (|put| argf '|value| (list (|genSomeVariable|) argt |$noEnv|)
    env)))
(list '|/throwAway| (|getmode| argf env) env )))))))

```

5.6.28 defun makeCategoryForm

[isCategoryForm p??]
 [compOrCroak p556]
 [\$EmptyMode p131]

— defun makeCategoryForm —

```

(defun |makeCategoryForm| (c env)
  (let (tmp1)
    (declare (special |$EmptyMode|))
    (when (|isCategoryForm| c env)
      (setq tmp1 (|compOrCroak| c |$EmptyMode| env))
      (list (first tmp1) (third tmp1)))))

```

5.6.29 defplist compCons plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'cons 'special) '|compCons|))

```

5.6.30 defun compCons

[compCons1 p279]
 [compForm p571]

— defun compCons —

```
(defun |compCons| (form mode env)
  (or (|compCons1| form mode env) (|compForm| form mode env)))
```

5.6.31 defun compCons1

```
[comp p557]
[convert p568]
[qcar p??]
[qcdr p??]
[$EmptyMode p131]
```

— defun compCons1 —

```
(defun |compCons1| (arg mode env)
  (let (mx y my yt mp mr ytp tmp1 x td)
    (declare (special |$EmptyMode|))
    (setq x (second arg))
    (setq y (third arg))
    (when (setq tmp1 (|comp| x |$EmptyMode| env))
      (setq x (first tmp1))
      (setq mx (second tmp1))
      (setq env (third tmp1))
      (cond
        ((null y)
         (|convert| (list (list 'list x) (list '|List| mx) env ) mode))
        (t
         (when (setq yt (|comp| y |$EmptyMode| env))
           (setq y (first yt))
           (setq my (second yt))
           (setq env (third yt))
           (setq td
            (cond
              ((and (consp my) (eq (qfirst my) '|List|) (consp (qrest my)))
               (setq mp (second my))
               (when (setq mr (list '|List| (|resolve| mp mx)))
                 (when (setq ytp (|convert| yt mr))
                   (when (setq tmp1 (|convert| (list x mx (third ytp)) (second mr)))
                     (setq x (first tmp1))
                     (setq env (third tmp1))
                     (cond
                       ((and (consp (car ytp)) (eq (qfirst (car ytp)) 'list))
                        (list (cons 'list (cons x (cdr (car ytp)))) mr env))
                       (t
                        (list (list 'cons x (car ytp)) mr env)))))))
           (t
            (list (list 'cons x (car ytp)) mr env)))))))
```

```
(list (list 'cons x y) (list '|Pair| mx my) env ))))
(|convert| td mode))))))
```

5.6.32 defplist compConstruct plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| 'special) '|compConstruct|))
```

5.6.33 defun compConstruct

```
[modeIsAggregateOf p??]
[compList p570]
[convert p568]
[compForm p571]
[compVector p344]
[getDomainsInScope p234]
```

— defun compConstruct —

```
(defun |compConstruct| (form mode env)
  (let (z y td tp)
    (setq z (cdr form))
    (cond
      ((setq y (|modeIsAggregateOf| '|List| mode env))
       (if (setq td (|compList| z (list '|List| (cadr y)) env))
           (|convert| td mode)
           (|compForm| form mode env)))
      ((setq y (|modeIsAggregateOf| '|Vector| mode env))
       (if (setq td (|compVector| z (list '|Vector| (cadr y)) env))
           (|convert| td mode)
           (|compForm| form mode env)))
      ((setq td (|compForm| form mode env)) td)
      (t
       (dolist (d (|getDomainsInScope| env))
         (cond
           ((and (setq y (|modeIsAggregateOf| '|List| d env))
                  (setq td (|compList| z (list '|List| (cadr y)) env))
                  (setq tp (|convert| td mode))))
```

```

(return tp))
((and (setq y (|modeIsAggregateOf| '|Vector| d env))
      (setq td (|compVector| z (list '|Vector| (cadr y)) env))
      (setq tp (|convert| td mode)))
 (return tp))))))

```

5.6.34 defplist compConstructorCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|ListCategory| 'special) '|compConstructorCategory|))

```

5.6.35 defplist compConstructorCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|RecordCategory| 'special) '|compConstructorCategory|))

```

5.6.36 defplist compConstructorCategory plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|UnionCategory| 'special) '|compConstructorCategory|))

```

5.6.37 defplist compConstructorCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|VectorCategory| 'special) '|compConstructorCategory|))
```

5.6.38 defun compConstructorCategory

```
[resolve p356]
[$Category p??]
```

— defun compConstructorCategory —

```
(defun |compConstructorCategory| (form mode env)
  (declare (special |$Category|))
  (list form (|resolve| |$Category| mode) env))
```

5.6.39 defplist compDefine plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'def 'special) '|compDefine|))
```

5.6.40 defun compDefine

```
[compDefine1 p283]
[$tripleCache p??]
[$tripleHits p??]
[$macroIfTrue p??]
[$packagesUsed p??]
```

— defun compDefine —

```
(defun |compDefine| (form mode env)
  (let (|$tripleCache| |$tripleHits| |$macroIfTrue| |$packagesUsed|)
    (declare (special |$tripleCache| |$tripleHits| |$macroIfTrue|
                      |$packagesUsed|)))
```

```
(setq |$tripleCache| nil)
(setq |$tripleHits| 0)
(setq |$macroIfTrue| nil)
(setq |$packagesUsed| nil)
(|compDefine1| form mode env)))
```

5.6.41 defun compDefine1

```
[macroExpand p136]
[isMacro p267]
[getSignatureFromMode p287]
[compDefine1 p283]
[compInternalFunction p287]
[compDefineAddSignature p133]
[compDefWhereClause p204]
[compDefineCategory p170]
[isDomainForm p338]
[getTargetFromRhs p135]
[giveFormalParametersValues p135]
[addEmptyCapsuleIfNecessary p134]
[compDefineFunctor p183]
[stackAndThrow p??]
[strconc p??]
[getAbbreviation p285]
[length p??]
[compDefineCapsuleFunction p288]
[$insideExpressionIfTrue p??]
[$formalArgList p??]
[$form p??]
[$op p??]
[$prefix p??]
[$insideFunctorIfTrue p??]
[$Category p??]
[$insideCategoryIfTrue p??]
[$insideCapsuleFunctionIfTrue p??]
[$ConstructorNames p??]
[$NoValueMode p131]
[$EmptyMode p131]
[$insideWhereIfTrue p??]
[$insideExpressionIfTrue p??]
```

— defun compDefine1 —

```

(defun |compDefine1| (form mode env)
  (let (|$insideExpressionIfTrue| lhs specialCases sig signature rhs newPrefix
        (tmp1 t))
    (declare (special |$insideExpressionIfTrue| |$formalArgList| |$form|
                      |$op| |$prefix| |$insideFunctorIfTrue| |$Category|
                      |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue|
                      |$ConstructorNames| |$NoValueMode| |$EmptyMode|
                      |$insideWhereIfTrue| |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| nil)
    (setq form (|macroExpand| form env))
    (setq lhs (second form))
    (setq signature (third form))
    (setq specialCases (fourth form))
    (setq rhs (fifth form))
    (cond
      ((and |$insideWhereIfTrue|
            (|isMacro| form env)
            (or (equal mode |$EmptyMode|) (equal mode |$NoValueMode|))))
        (list lhs mode (|put| (car lhs) '|macro| rhs env)))
      ((and (null (car signature)) (consp rhs)
            (null (member (qfirst rhs) |$ConstructorNames|))
            (setq sig (|getSignatureFromMode| lhs env))))
        (|compDefine1|
         (list 'def lhs (cons (car sig) (cdr signature)) specialCases rhs)
         mode env))
      (|$insideCapsuleFunctionIfTrue| (|compInternalFunction| form mode env))
      (t
       (when (equal (car signature) |$Category|) (setq |$insideCategoryIfTrue| t))
       (setq env (|compDefineAddSignature| lhs signature env))
       (cond
         ((null (dolist (x (rest signature) tmp1) (setq tmp1 (and tmp1 (null x)))))
          (|compDefWhereClause| form mode env))
         ((equal (car signature) |$Category|)
          (|compDefineCategory| form mode env nil |$formalArgList|))
         ((and (|isDomainForm| rhs env) (null |$insideFunctorIfTrue|))
          (when (null (car signature))
            (setq signature
              (cons (|getTargetFromRhs| lhs rhs
                    (|giveFormalParametersValues| (cdr lhs) env))
                    (cdr signature))))
          (setq rhs (|addEmptyCapsuleIfNecessary| (car signature) rhs))
          (|compDefineFunctor|
           (list 'def lhs signature specialCases rhs)
           mode env NIL |$formalArgList|))
         ((null |$form|)
          (|stackAndThrow| (list "bad == form " form)))
         (t
          (setq newPrefix
            (if |$prefix|
              (intern (strconc (|encodeItem| |$prefix|) ", " (|encodeItem| |$op|))))

```

```
(|getAbbreviation| |$op| (|#| (cdr |$form|))))
(|compDefineCapsuleFunction|
 form mode env newPrefix |$formalArgList|))))))
```

5.6.42 defun getAbbreviation

```
[constructor? p??]
[assq p??]
[mkAbbrev p286]
[rplac p??]
[$abbreviationTable p??]
[$abbreviationTable p??]
```

— defun getAbbreviation —

```
(defun |getAbbreviation| (name c)
  (let (cname x n upc newAbbreviation)
    (declare (special |$abbreviationTable|))
    (setq cname (|constructor?| name))
    (cond
      ((setq x (assq cname |$abbreviationTable|))
       (cond
         ((setq n (assq name (cdr x)))
          (cond
            ((setq upc (assq c (cdr n)))
             (cdr upc))
            (t
             (setq newAbbreviation (|mkAbbrev| x cname))
             (rplac (cdr n) (cons (cons c newAbbreviation) (cdr n)))
             newAbbreviation)))
         (t
          (setq newAbbreviation (|mkAbbrev| x x))
          (rplac (cdr x)
                  (cons (cons name (list (cons c newAbbreviation))) (cdr x)))
          newAbbreviation)))
      (t
       (setq |$abbreviationTable|
              (cons (list cname (list name (cons c cname))) |$abbreviationTable|)
                    cname))))))
```

5.6.43 defun mkAbbrev

```
[addSuffix p286]
[alistSize p286]
```

— defun mkAbbrev —

```
(defun |mkAbbrev| (x z)
  (|addSuffix| (|alistSize| (cdr x)) z))
```

5.6.44 defun addSuffix

— defun addSuffix —

```
(defun |addSuffix| (n u)
  (let (s)
    (if (alpha-char-p (elt (spadlet s (stringimage u)) (maxindex s)))
        (intern (strconc s (stringimage n)))
        (intern1 (strconc s (stringimage '|;|) (stringimage n))))))
```

5.6.45 defun alistSize

— defun alistSize —

```
(defun |alistSize| (c)
  (labels (
    (count (x level)
      (cond
        ((eql level 2) (|#| x))
        ((null x) 0)
        (+ (count (cdar x) (1+ level))
           (count (cdr x) level)))))
    (count c 1)))
```

5.6.46 defun getSignatureFromMode

```
[getmode p??]
[opOf p??]
[qcar p??]
[qcdr p??]
[nequal p??]
[length p??]
[stackAndThrow p??]
[eqsubstlist p??]
[take p??]
[$FormalMapVariableList p250]
```

— **defun getSignatureFromMode** —

```
(defun |getSignatureFromMode| (form env)
  (let (tmp1 signature)
    (declare (special |$FormalMapVariableList|))
    (setq tmp1 (|getmode| (|opOf| form) env))
    (when (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|))
      (setq signature (qrest tmp1))
      (if (nequal (|#| form) (|#| signature))
          (|stackAndThrow| (list '|Wrong number of arguments: | form))
          (eqsubstlist (cdr form)
                        (take (|#| (cdr form)) |$FormalMapVariableList|
                             signature))))))
```

—————

5.6.47 defun compInternalFunction

```
[identp p??]
[stackAndThrow p??]
```

— **defun compInternalFunction** —

```
(defun |compInternalFunction| (df m env)
  (let (form signature specialCases body op argl nbody nf ress)
    (setq form (second df))
    (setq signature (third df))
    (setq specialCases (fourth df))
    (setq body (fifth df))
    (setq op (first form))
    (setq argl (rest form))
    (cond
      ((null (identp op))
```

```

(|stackAndThrow| (list '|Bad name for internal function:| op)))
((eq1 (|#| arg1) 0)
 (|stackAndThrow|
  (list '|Argumentless internal functions unsupported:| op )))
(t
 (setq nbody (list '+-> arg1 body))
 (setq nf (list 'let (list '|:| op (cons '|Mapping| signature)) nbody))
 (setq ress (|comp| nf m env)) ress)))

```

5.6.48 defun compDefineCapsuleFunction

```

[length p??]
[get p??]
[profileRecord p??]
[compArgumentConditions p294]
[addDomain p232]
[giveFormalParametersValues p135]
[getSignature p296]
[put p??]
[stripOffSubdomainConditions p295]
[getArgumentModeOrMoan p156]
[checkAndDeclare p298]
[hasSigInTargetCategory p298]
[stripOffArgumentConditions p296]
[resolve p356]
[member p??]
[getmode p??]
[formatUnabbreviated p??]
[sayBrightly p??]
[compOrCroak p556]
[NRTassignCapsuleFunctionSlot p??]
[mkq p??]
[replaceExitEtc p327]
[addArgumentConditions p293]
[compileCases p291]
[addStats p??]
[$semanticErrorStack p??]
[$DomainsInScope p??]
[$op p??]
[$formalArgList p??]
[$signatureOfForm p??]
[$functionLocations p??]
[$profileCompiler p??]

```

```

[$compileOnlyCertainItems p??]
[$returnMode p??]
[$functorStats p??]
[$functionStats p??]
[$form p??]
[$functionStats p??]
[$argumentConditionList p??]
[$finalEnv p??]
[$initCapsuleErrorCount p??]
[$insideCapsuleFunctionIfTrue p??]
[$CapsuleModemapFrame p??]
[$CapsuleDomainsInScope p??]
[$insideExpressionIfTrue p??]
[$returnMode p??]
[$op p??]
[$formalArgList p??]
[$signatureOfForm p??]
[$functionLocations p??]

```

— **defun compDefineCapsuleFunction** —

```

(defun |compDefineCapsuleFunction| (df m oldE |$prefix| |$formalArgList|)
  (declare (special |$prefix| |$formalArgList|))
  (let (|$form| |$op| |$functionStats| |$argumentConditionList| |$finalEnv|
        |$initCapsuleErrorCount| |$insideCapsuleFunctionIfTrue|
        |$CapsuleModemapFrame| |$CapsuleDomainsInScope|
        |$insideExpressionIfTrue| form signature body tmp1 lineNumber
        specialCases argl identSig argModeList signaturep e rettype tmp2
        localOrExported formattedSig tt catchTag bodyp finalBody fun val)
    (declare (special |$form| |$op| |$functionStats| |$functorStats|
                      |$argumentConditionList| |$finalEnv| |$returnMode|
                      |$initCapsuleErrorCount| |$newCompCompare| |$NoValueMode|
                      |$insideCapsuleFunctionIfTrue|
                      |$CapsuleModemapFrame| |$CapsuleDomainsInScope| | |
                      |$insideExpressionIfTrue| |$compileOnlyCertainItems|
                      |$profileCompiler| |$functionLocations| |$finalEnv|
                      |$signatureOfForm| |$semanticErrorStack|))
      (setq form (second df))
      (setq signature (third df))
      (setq specialCases (fourth df))
      (setq body (fifth df))
      (setq tmp1 specialCases)
      (setq lineNumber (first tmp1))
      (setq specialCases (rest tmp1))
      (setq e oldE)
      ;-1. bind global variables
      (setq |$form| nil)
      (setq |$op| nil)

```

```

(setq |$functionStats| (list 0 0))
(setq |$argumentConditionList| nil)
(setq |$finalEnv| nil)
; used by ReplaceExitEtc to get a common environment
(setq |$initCapsuleErrorCount| (|#| |$semanticErrorStack|))
(setq |$insideCapsuleFunctionIfTrue| t)
(setq |$CapsuleModemapFrame| e)
(setq |$CapsuleDomainsInScope| (|get| '|$DomainsInScope| 'special e))
(setq |$insideExpressionIfTrue| t)
(setq |$returnMode| m)
(setq |$op| (first form))
(setq argl (rest form))
(setq |$form| (cons |$op| argl))
(setq argl (|stripOffArgumentConditions| argl))
(setq |$formalArgList| (append argl |$formalArgList|))
; let target and local signatures help determine modes of arguments
(setq argModeList
  (cond
    ((setq identSig (|hasSigInTargetCategory| argl form (car signature) e))
      (setq e (|checkAndDeclare| argl form identSig e))
      (cdr identSig))
    (t
      (loop for a in argl
        collect (|getArgumentModeOrMoan| a form e))))))
(setq argModeList (|stripOffSubdomainConditions| argModeList argl))
(setq signaturep (cons (car signature) argModeList))
(unless identSig
  (setq oldE (|put| |$op| '|mode| (cons '|Mapping| signaturep) oldE)))
; obtain target type if not given
(cond
  ((null (car signaturep))
    (setq signaturep
      (cond
        (identSig identSig)
        (t (|getSignature| |$op| (cdr signaturep) e))))))
(when signaturep
  (setq e (|giveFormalParametersValues| argl e))
  (setq |$signatureOfForm| signaturep)
  (setq |$functionLocations|
    (cons (cons (list |$op| |$signatureOfForm|) lineNumber)
      |$functionLocations|))
  (setq e (|addDomain| (car signaturep) e))
  (setq e (|compArgumentConditions| e))
  (when |$profileCompiler|
    (loop for x in argl for y in signaturep
      do (|profileRecord| '|arguments| x y)))
  ; 4. introduce needed domains into extendedEnv
  (loop for domain in signaturep
    do (setq e (|addDomain| domain e)))
  ; 6. compile body in environment with extended environment

```

```

(setq rettype (|resolve| (car signaturep) |$returnMode|))
(setq localOrExported
  (cond
    ((and (null (|member| |$op| |$formalArgList|))
      (progn
        (setq tmp2 (|getModel| |$op| e))
        (and (consp tmp2) (eq (qfirst tmp2) '|Mapping|))))
      '|local|)
    (t '|exported|)))
; 6a skip if compiling only certain items but not this one
; could be moved closer to the top
(setq formattedSig (|formatUnabbreviated| (cons '|Mapping| signaturep)))
(cond
  ((and |$compileOnlyCertainItems|
    (null (|member| |$op| |$compileOnlyCertainItems|)))
    (|sayBrightly|
      (cons " skipping " (cons localOrExported (|bright| |$op|))))
    (list nil (cons '|Mapping| signaturep) oldE))
  (t
    (|sayBrightly|
      (cons " compiling " (cons localOrExported (append (|bright| |$op|)
        (cons ":" formattedSig)))))
    (setq tt (catch '|compCapsuleBody| (|compOrCroak| body rettype e)))
    (|NRTassignCapsuleFunctionSlot| |$op| signaturep)
    ; A THROW to the above CATCH occurs if too many semantic errors occur
    ; see stackSemanticError
    (setq catchTag (mkq (gensym)))
    (setq fun
      (progn
        (setq bodyp
          (|replaceExitEtc| (car tt) catchTag '|TAGGEDreturn| |$returnMode|))
        (setq bodyp (|addArgumentConditions| bodyp |$op|))
        (setq finalBody (list 'catch catchTag bodyp))
        (|compileCases|
          (list |$op| (list 'lam (append argl (list '$) finalBody))
            oldE)))
        (setq |$functorStats| (|addStats| |$functorStats| |$functionStats|))
        ; 7. give operator a 'value property
        (setq val (list fun signaturep e))
        (list fun (list '|Mapping| signaturep) oldE))))))

```

5.6.49 defun compileCases

```

[eval p??]
[qcar p??]
[qcdr p??]

```

```

[msubst p??]
[compile p145]
[getSpecialCaseAssoc p293]
[get p??]
[assocleft p??]
[outerProduct p??]
[assocright p??]
[mkpf p??]
[$getDomainCode p??]
[$insideFunctorIfTrue p??]
[$specialCaseKeyList p??]

```

— defun compileCases —

```

(defun |compileCases| (x |$e|)
  (declare (special |$e|))
  (labels (
    (isEltArgumentIn (Rlist x)
      (cond
        ((atom x) nil)
        ((and (consp x) (eq (qfirst x) 'elt) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (or (member (second x) Rlist)
              (isEltArgumentIn Rlist (cdr x))))))
        ((and (consp x) (eq (qfirst x) 'qrefelt) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (or (member (second x) Rlist)
              (isEltArgumentIn Rlist (cdr x))))))
        (t
         (or (isEltArgumentIn Rlist (car x))
             (isEltArgumentIn Rlist (cdr x))))))
    (FindNamesFor (r rp)
      (let (v u)
        (declare (special |$getDomainCode|))
        (cons r
          (loop for item in |$getDomainCode|
            do
              (setq v (second item))
              (setq u (third item))
              when (and (equal (second u) r) (|eval| (msubst rp r u)))
                collect v))))))
    (let (|$specialCaseKeyList| specialCaseAssoc listOfDomains listOfAllCases cl)
      (declare (special |$specialCaseKeyList| |$true| |$insideFunctorIfTrue|))
      (setq |$specialCaseKeyList| nil)
      (cond
        ((null (eq |$insideFunctorIfTrue| t)) (|compile| x))
        (t
         (setq specialCaseAssoc
           (loop for y in (|getSpecialCaseAssoc|)

```

```

when (and (null (|get| (first y) '|specialCase| |$e|))
          (isEltArgumentIn (FindNamesFor (first y) (second y)) x))
  collect y))
(cond
 ((null specialCaseAssoc) (|compile| x))
 (t
  (setq listOfDomains (assocleft specialCaseAssoc))
  (setq listOfAllCases (|outerProduct| (assocright specialCaseAssoc)))
  (setq cl
    (loop for z in listOfAllCases
      collect
      (progn
        (setq |$specialCaseKeyList|
          (loop for d in listOfDomains for c in z
            collect (cons d c)))
        (cons
          (mkpf
            (loop for d in listOfDomains for c in z
              collect (list 'equal d c))
            'and)
          (list (|compile| (copy x))))))
    (setq |$specialCaseKeyList| nil)
    (cons 'cond (append cl (list (list |$true| (|compile| x))))))))))

```

5.6.50 defun getSpecialCaseAssoc

```

[$functorForm p??]
[$functorSpecialCases p??]

```

— defun getSpecialCaseAssoc —

```

(defun |getSpecialCaseAssoc| ()
  (declare (special |$functorSpecialCases| |$functorForm|))
  (loop for r in (rest |$functorForm|)
    for z in (rest |$functorSpecialCases|)
    when z
    collect (cons r z)))

```

5.6.51 defun addArgumentConditions

```

[qcar p??]
[qcdr p??]

```

```

[mkq p??]
[systemErrorHere p??]
[$true p??]
[$functionName p??]
[$body p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— **defun addArgumentConditions** —

```

(defun |addArgumentConditions| (|$body| |$functionName|)
  (declare (special |$body| |$functionName| |$argumentConditionList| |$true|))
  (labels (
    (fn (clist)
      (let (n untypedCondition typedCondition)
        (cond
          ((and (consp clist) (consp (qfirst clist)) (consp (qcдар clist))
              (consp (qcddar clist))
              (eq (qcdddar clist) nil))
            (setq n (qcaar clist))
            (setq untypedCondition (qcadar clist))
            (setq typedCondition (qcaddar clist))
            (list 'cond
              (list typedCondition (fn (cdr clist)))
              (list |$true|
                (list '|argumentDataError| n
                  (mkq untypedCondition) (mkq |$functionName|))))))
          ((null clist) |$body|)
          (t (|systemErrorHere| "addArgumentConditions")))))
    (if |$argumentConditionList|
      (fn |$argumentConditionList|
        |$body|)))

```

—

5.6.52 defun compArgumentConditions

```

[msubst p??]
[compOrCroak p556]
[$Boolean p??]
[$argumentConditionList p??]
[$argumentConditionList p??]

```

— **defun compArgumentConditions** —

```

(defun |compArgumentConditions| (env)

```

```
(let (n a x y tmp1)
(declare (special |$Boolean| |$argumentConditionList|))
(setq |$argumentConditionList|
(loop for item in |$argumentConditionList|
do
(setq n (first item))
(setq a (second item))
(setq x (third item))
(setq y (msubst a '|#1|' x))
(setq tmp1 (|compOrCroak| y |$Boolean| env))
(setq env (third tmp1)))
collect
(list n x (first tmp1))))
env))
```

```
[qcar p??]
[qcdr p??]
[assoc p??]
[mkpf p??]
[$argumentConditionList p??]
[$argumentConditionList p??]
```

— defun stripOffSubdomainConditions —

```
(defun |stripOffSubdomainConditions| (margl argl)
  (let (pair (i 0))
    (declare (special |$argumentConditionList|))
    (loop for x in margl for arg in argl
      do (incf i)
      collect
        (cond
          ((and (consp x) (eq (qfirst x) '|SubDomain|) (consp (qrest x))
            (consp (qcddr x)) (eq (qcdddr x) nil))
            (cond
              ((setq pair (|assoc| i |$argumentConditionList|))
                (rplac (cadr pair) (mkpf (list (third x) (cadr pair)) 'and))
                (second x))
              (t
               (setq |$argumentConditionList|
                 (cons (list i arg (third x)) |$argumentConditionList|)
                 (second x))))
            (t x))))))
```

5.6.54 defun stripOffArgumentConditions

```
[qcar p??]
[qcdr p??]
[msubst p??]
[$argumentConditionList p??]
[$argumentConditionList p??]
```

— defun stripOffArgumentConditions —

```
(defun |stripOffArgumentConditions| (argl)
  (let (condition (i 0))
    (declare (special |$argumentConditionList|))
    (loop for x in argl
      do (incf i)
        collect
          (cond
            ((and (consp x) (eq (qfirst x) '|\\|') (consp (qrest x))
              (consp (qcddr x)) (eq (qcdddr x) nil))
             (setq condition (msubst '|#1| (second x) (third x)))
             (setq |$argumentConditionList|
               (cons (list i (second x) condition) |$argumentConditionList|))
             (second x))
            (t x)))))
```

5.6.55 defun getSignature

Try to return a signature. If there isn't one, complain and return nil. If there are more than one then remove any that are subsumed. If there is still more than one complain else return the only signature. [get p??]

```
[length p??]
[remdup p??]
[knownInfo p??]
[getmode p??]
[qcar p??]
[qcdr p??]
[say p??]
[printSignature p??]
[SourceLevelSubsume p??]
[stackSemanticError p??]
[$e p??]
```

— defun getSignature —

```
(defun |getSignature| (op argModeList |$e|)
  (declare (special |$e|))
  (let (mmList pred u tmp1 dc sig sigl)
    (setq mmList (|get| op '|modemap| |$e|))
    (cond
      ((eq 1
        (|#| (setq sigl (remdup
          (loop for item in mmList
            do
              (setq dc (caar item))
              (setq sig (cdar item))
              (setq pred (caadr item))
              when (and (eq dc '$) (equal (cdr sig) argModeList) (|knownInfo| pred))
                collect sig))))))
        (car sigl))
      ((null sigl)
        (cond
          ((progn
            (setq tmp1 (setq u (|getModel| op |$e|)))
            (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)))
            (qrest tmp1))
          (t
            (say "***** USER ERROR *****")
            (say "available signatures for " op ": ")
            (cond
              ((null mmList) (say "  NONE"))
              (t
                (loop for item in mmList
                  do (|printSignature| '|      | op (cdar item)))
                (|printSignature| '|NEED | op (cons '? argModeList)))
              nil)))
          (t
            ; Before we complain about duplicate signatures, we should
            ; check that we do not have for example, a partial - as
            ; well as a total one. SourceLevelSubsume should do this
            (loop for u in sigl do
              (loop for v in sigl
                when (null (equal u v))
                  do (when (|SourceLevelSubsume| u v) (setq sigl (|delete| v sigl)))))
            (cond
              ((eq 1 (|#| sigl)) (car sigl))
              (t
                (|stackSemanticError|
                  (list '|duplicate signatures for | op '|: | argModeList) nil))))))))))
```

5.6.56 defun checkAndDeclare

```
[getArgumentMode p299]
[modeEqual p357]
[put p??]
[sayBrightly p??]
[bright p??]
```

— defun checkAndDeclare —

```
(defun |checkAndDeclare| (argl form sig env)
  (let (m1 stack)
    (loop for a in argl for m in (rest sig)
      do
        (if (setq m1 (|getArgumentMode| a env))
          (if (null (|modeEqual| m1 m))
            (setq stack
              (cons '| | (append (|bright| a)
                (cons "must have type "
                  (cons m
                    (cons " not "
                      (cons m1
                        (cons '|%1| stack))))))))
            (setq env (|put| a '|mode| m env)))
          (when stack
            (|sayBrightly|
              (cons " Parameters of "
                (append (|bright| (car form))
                  (cons " are of wrong type:"
                    (cons '|%1| stack))))))
            env)))
```

—————

5.6.57 defun hasSigInTargetCategory

```
[getArgumentMode p299]
[remdup p??]
[length p??]
[getSignatureFromMode p287]
[stackWarning p??]
[compareMode2Arg p??]
[bright p??]
[$domainShell p??]
```

— defun hasSigInTargetCategory —

```

(defun |hasSigInTargetCategory| (arg1 form opsig env)
  (labels (
    (fn (opName sig opsig mList form)
      (declare (special |$op|))
      (and
        (and
          (and (equal opName |$op|) (equal (|#| sig) (|#| form)))
          (or (null opsig) (equal opsig (car sig))))
        (let ((result t))
          (loop for x in mList for y in (rest sig)
            do (setq result (and result (or (null x) (|modeEqual| x y))))
          result))))
    (let (mList potentialSigList c sig)
      (declare (special |$domainShell|))
      (setq mList
        (loop for x in arg1
          collect (|getArgumentMode| x env)))
      (setq potentialSigList
        (remdup
          (loop for item in (elt |$domainShell| 1)
            when (fn (caar item) (cadar item) opsig mList form)
              collect (cadar item))))
      (setq c (|#| potentialSigList))
      (cond
        ((eql 1 c) (car potentialSigList))
        ((eql 0 c)
          (when (equal (|#| (setq sig (|getSignatureFromMode| form env))) (|#| form))
            sig))
        ((> c 1)
          (setq sig (car potentialSigList))
          (|stackWarning|
            (cons '|signature of lhs not unique:|
              (append (|bright| sig) (list '|chosen|))))
          sig)
        (t nil))))))

```

5.6.58 defun getArgumentMode

[get p??]

— defun getArgumentMode —

```

(defun |getArgumentMode| (x e)
  (if (stringp x) x (|get| x '|mode| e)))

```

5.6.59 defplist compElt plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|elt| 'special) '|compElt|))
```

5.6.60 defun compElt

```
[compForm p571]
[isDomainForm p338]
[addDomain p232]
[getModemapListFromDomain p244]
[length p??]
[stackMessage p??]
[stackWarning p??]
[convert p568]
[opOf p??]
[getDeltaEntry p??]
[nequal p??]
[$One p??]
[$Zero p??]
```

— defun compElt —

```
(defun |compElt| (form mode env)
  (let (aDomain anOp mmList n modemap sig pred val)
    (declare (special |$One| |$Zero|))
    (setq anOp (third form))
    (setq aDomain (second form))
    (cond
      ((null (and (consp form) (eq (qfirst form) '|elt|)
                  (consp (qrest form)) (consp (qcddr form))
                  (eq (qcdddr form) nil))))
      (|compForm| form mode env))
    ((eq aDomain '|Lisp|)
     (list (cond
              ((equal anOp |$Zero|) 0)
              ((equal anOp |$One|) 1)
              (t anOp))
```

```

      mode env))
((|isDomainForm| aDomain env)
 (setq env (|addDomain| aDomain env))
 (setq mmList (|getModemapListFromDomain| anOp 0 aDomain env))
 (setq modemap
  (progn
   (setq n (|#| mmList))
   (cond
    ((eq 1 n) (elt mmList 0))
    ((eq 0 n)
     (|stackMessage|
      (list "Operation " '|%b| anOp '|%d| "missing from domain: "
            aDomain nil))
      nil)
    (t
     (|stackWarning|
      (list "more than 1 modemap for: " anOp " with dc="
            aDomain " ==>" mmList ))
      (elt mmList 0))))))
(when modemap
 (setq sig (first modemap))
 (setq pred (caadr modemap))
 (setq val (cadadr modemap))
 (unless (and (nequal (|#| sig) 2)
              (null (and (consp val) (eq (qfirst val) '|elt|))))
  (setq val (|genDeltaEntry| (cons (|opOf| anOp) modemap)))
  (|convert| (list (list '|call| val) (second sig) env) mode))))
(t
 (|compForm| form mode env))))

```

5.6.61 defplist compExit plist

— postvars —

```

(eval-when (eval load)
 (setf (get '|exit| 'special) '|compExit|))

```

5.6.62 defun compExit

```

[comp p557]
[modifyModeStack p593]

```

```
[stackMessageIfNone p??]
[$exitModeStack p??]
```

— **defun compExit** —

```
(defun |compExit| (form mode env)
  (let (exitForm index m1 u)
    (declare (special |$exitModeStack|))
    (setq index (1- (second form)))
    (setq exitForm (third form))
    (cond
      ((null |$exitModeStack|)
       (|comp| exitForm mode env))
      (t
       (setq m1 (elt |$exitModeStack| index))
       (setq u (|comp| exitForm m1 env))
       (cond
         (u
          (|modifyModeStack| (second u) index)
          (list (list ' |TAGGEDexit| index u) mode env))
         (t
          (|stackMessageIfNone|
           (list ' |cannot compile exit expression| exitForm ' |in mode| m1))))))))))
```

5.6.63 defplist compHas plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get ' |has| 'special) ' |compHas|))
```

5.6.64 defun compHas

```
[chaseInferences p??]
[compHasFormat p303]
[coerce p346]
[$e p??]
[$e p??]
[$Boolean p??]
```

— **defun compHas** —

```
(defun |compHas| (pred mode |$e|)
  (declare (special |$e| |$Boolean|))
  (let (a b predCode)
    (setq a (second pred))
    (setq b (third pred))
    (setq |$e| (|chaseInferences| pred |$e|))
    (setq predCode (|compHasFormat| pred))
    (|coerce| (list predCode |$Boolean| |$e|) mode)))
```

—————

5.6.65 **defun compHasFormat**

```
[take p??]
[length p??]
[sublislis p??]
[comp p557]
[qcar p??]
[qcdr p??]
[mkList p304]
[mkDomainConstructor p??]
[isDomainForm p338]
[$FormalMapVariableList p250]
[$EmptyMode p131]
[$e p??]
[$form p??]
[$EmptyEnvironment p??]
```

— **defun compHasFormat** —

```
(defun |compHasFormat| (pred)
  (let (olda b argl formals tmp1 a)
    (declare (special |$EmptyEnvironment| |$e| |$EmptyMode|
                      |$FormalMapVariableList| |$form|))
    (when (eq (car pred) '|has|) (car pred))
    (setq olda (second pred))
    (setq b (third pred))
    (setq argl (rest |$form|))
    (setq formals (take (|#| argl) |$FormalMapVariableList|))
    (setq a (sublislis argl formals olda))
    (setq tmp1 (|comp| a |$EmptyMode| |$e|))
    (when tmp1
      (setq a (car tmp1)))
```

```

(setq a (sublislis formalis argl a))
(cond
  ((and (consp b) (eq (qfirst b) 'attribute) (consp (qrest b))
        (eq (qcddr b) nil))
    (list '|HasAttribute| a (list 'quote (qsecond b))))
  ((and (consp b) (eq (qfirst b) 'signature) (consp (qrest b))
        (consp (qcddr b)) (eq (qcdddr b) NIL))
    (list '|HasSignature| a
          (|mkList|
            (list (MKQ (qsecond b))
                  (|mkList|
                    (loop for type in (qthird b)
                        collect (|mkDomainConstructor| type)))))))
  ((|isDomainForm| b |$EmptyEnvironment|)
    (list 'equal a b))
  (t
    (list '|HasCategory| a (|mkDomainConstructor| b))))))

```

5.6.66 defun mkList

— defun mkList —

```

(defun |mkList| (u)
  (when u (cons 'list u)))

```

5.6.67 defplist compIf plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'if 'special) '|compIf|))

```

5.6.68 defun compIf

```

[canReturn p306]
[intersectionEnvironment p??]

```

[compBoolean p308]
 [compFromIf p305]
 [resolve p356]
 [coerce p346]
 [quotify p??]
 [\$Boolean p??]

— **defun compIf** —

```
(defun |compIf| (form mode env)
  (labels (
    (environ (bEnv cEnv b c env)
      (cond
        ((|canReturn| b 0 0 t)
          (if (|canReturn| c 0 0 t) (|intersectionEnvironment| bEnv cEnv) bEnv))
        ((|canReturn| c 0 0 t) cEnv)
        (t env))))
    (let (a b c tmp1 xa ma Ea Einv Tb xb mb Eb Tc xc mc Ec xbp x returnEnv)
      (declare (special |$Boolean|))
      (setq a (second form))
      (setq b (third form))
      (setq c (fourth form))
      (when (setq tmp1 (|compBoolean| a |$Boolean| env))
        (setq xa (first tmp1))
        (setq ma (second tmp1))
        (setq Ea (third tmp1))
        (setq Einv (fourth tmp1))
        (when (setq Tb (|compFromIf| b mode Ea))
          (setq xb (first Tb))
          (setq mb (second Tb))
          (setq Eb (third Tb))
          (when (setq Tc (|compFromIf| c (|resolve| mb mode) Einv))
            (setq xc (first Tc))
            (setq mc (second Tc))
            (setq Ec (third Tc))
            (when (setq xbp (|coerce| Tb mc))
              (setq x (list 'if xa (first xbp) xc))
              (setq returnEnv (environ (third xbp) Ec (first xbp) xc env))
              (list x mc returnEnv))))))))))
```

—————

5.6.69 defun compFromIf

[comp p557]

— **defun compFromIf** —

```
(defun |compFromIf| (a m env)
  (if (eq a '|noBranch|)
      (list '|noBranch| m env)
      (|comp| a m env)))
```

5.6.70 defun canReturn

```
[say p??]
[qcar p??]
[qcdr p??]
[canReturn p306]
[systemErrorHere p??]
```

— defun canReturn —

```
(defun |canReturn| (expr level exitCount ValueFlag)
  (labels (
    (findThrow (gs expr level exitCount ValueFlag)
      (cond
        ((atom expr) nil)
        ((and (consp expr) (eq (qfirst expr) 'throw) (consp (qrest expr))
          (equal (qsecond expr) gs) (consp (qcddr expr))
          (eq (qcddr expr) nil))
        t)
      ((and (consp expr) (eq (qfirst expr) 'seq))
        (let (result)
          (loop for u in (qrest expr)
            do (setq result
              (or result
                (findThrow gs u (1+ level) exitCount ValueFlag))))
          result))
      (t
        (let (result)
          (loop for u in (rest expr)
            do (setq result
              (or result
                (findThrow gs u level exitCount ValueFlag))))
          result))))))
  (let (op gs)
    (cond
      ((atom expr) (and ValueFlag (equal level exitCount)))
      ((eq (setq op (car expr)) 'quote) (and ValueFlag (equal level exitCount)))
      ((eq op '|TAGGEDexit|)
        (cond
          ((and (consp expr) (consp (qrest expr)) (consp (qcddr expr))
```

```

      (eq (qcdddr expr) nil))
    (|canReturn| (car (third expr)) level (second expr)
      (equal (second expr) level))))
  ((and (equal level exitCount) (null ValueFlag))
    nil)
  ((eq op 'seq)
    (let (result)
      (loop for u in (rest expr)
        do (setq result (or result (|canReturn| u (1+ level) exitCount nil))))
      result))
  ((eq op '|TAGGEDreturn|) nil)
  ((eq op 'catch)
    (cond
      ((findThrow (second expr) (third expr) level
        exitCount ValueFlag)
        t)
      (t
        (|canReturn| (third expr) level exitCount ValueFlag))))
  ((eq op 'cond)
    (cond
      ((equal level exitCount)
        (let (result)
          (loop for u in (rest expr)
            do (setq result (or result
              (|canReturn| (|last| u) level exitCount ValueFlag))))
          result))
      (t
        (let (outer)
          (loop for v in (rest expr)
            do (setq outer (or outer
              (let (inner)
                (loop for u in v
                  do (setq inner
                    (or inner
                      (findThrow gs u level exitCount ValueFlag))))
                inner))))
          outer))))))
  ((eq op 'if)
    (and (consp expr) (consp (qrest expr)) (consp (qcddr expr))
      (consp (qcdddr expr))
      (eq (qcdddr expr) nil))
    (cond
      ((null (|canReturn| (second expr) 0 0 t))
        (say "IF statement can not cause consequents to be executed")
        (|pp| expr)))
      (or (|canReturn| (second expr) level exitCount nil)
        (|canReturn| (third expr) level exitCount ValueFlag)
        (|canReturn| (fourth expr) level exitCount ValueFlag)))
  ((atom op)
    (let ((result t))

```

```

(loop for u in expr
  do (setq result
    (and result (|canReturn| u level exitCount ValueFlag))))
result))
((and (consp op) (eq (qfirst op) 'xlam) (consp (qrest op))
  (consp (qcddr op)) (eq (qcdddr op) nil))
  (let ((result t))
    (loop for u in expr
      do (setq result
        (and result (|canReturn| u level exitCount ValueFlag))))
    result))
(t (|systemErrorHere| "canReturn")))))))

```

5.6.71 defun compBoolean

```

[comp p557]
[getSuccessEnvironment p308]
[getInverseEnvironment p310]

```

— defun compBoolean —

```

(defun |compBoolean| (p mode env)
  (let (tmp1 pp)
    (when (setq tmp1 (OR (|comp| p mode env)))
      (setq pp (car tmp1))
      (setq mode (cadr tmp1))
      (setq env (caddr tmp1))
      (list pp mode (|getSuccessEnvironment| p env)
        (|getInverseEnvironment| p env)))))

```

5.6.72 defun getSuccessEnvironment

```

[qcar p??]
[qcdr p??]
[isDomainForm p338]
[put p??]
[identp p??]
[getProplist p??]
[comp p557]
[consProplistOf p??]

```

```
[removeEnv p??]
[addBinding p??]
[get p??]
[$EmptyEnvironment p??]
[$EmptyMode p131]
```

— **defun getSuccessEnvironment** —

```
(defun |getSuccessEnvironment| (a env)
  (let (id currentProplist tt newProplist x m)
    (declare (special |$EmptyMode| |$EmptyEnvironment|))
    (cond
      ((and (consp a) (eq (qfirst a) '|has|) (CONSP (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil))
        (if
          (and (identp (second a)) (|isDomainForm| (third a) |$EmptyEnvironment|))
          (|put| (second a) '|specialCase| (third a) env)
          env))
      ((and (consp a) (eq (qfirst a) '|is|) (consp (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil))
        (setq id (qsecond a))
        (setq m (qthird a))
        (cond
          ((and (identp id) (|isDomainForm| m |$EmptyEnvironment|))
            (setq env (|put| id '|specialCase| m env))
            (setq currentProplist (|getProplist| id env))
            (setq tt (|comp| m |$EmptyMode| env))
            (when tt
              (setq env (caddr tt))
              (setq newProplist
                (|consProplistOf| id currentProplist '|value|
                  (cons m (cdr (|removeEnv| tt))))))
              (|addBinding| id newProplist env)))
          (t env)))
      ((and (consp a) (eq (qfirst a) '|case|) (consp (qrest a))
        (consp (qcddr a)) (eq (qcdddr a) nil)
        (identp (qsecond a)))
        (setq x (qsecond a))
        (setq m (qthird a))
        (|put| x '|condition| (cons a (|get| x '|condition| env)) env))
      (t env))))
```

—

5.6.73 defun getInverseEnvironment

```
[qcar p??]
[qcdr p??]
[identp p??]
[isDomainForm p338]
[put p??]
[get p??]
[member p??]
[mkpf p??]
[delete p??]
[getUnionMode p311]
[$EmptyEnvironment p??]
```

— defun getInverseEnvironment —

```
(defun |getInverseEnvironment| (a env)
  (let (op arg1 x m oldpred tmp1 zz newpred)
    (declare (special |$EmptyEnvironment|))
    (cond
      ((atom a) env)
      (t
       (setq op (car a))
       (setq arg1 (cdr a))
       (cond
         ((eq op '|has|)
          (setq x (car arg1))
          (setq m (cadr arg1))
          (cond
            ((and (identp x) (|isDomainForm| m |$EmptyEnvironment|))
             (|put| x '|specialCase| m env))
            (t env)))
         ((and (consp a) (eq (qfirst a) '|case|) (consp (qrest a))
              (consp (qcddr a)) (eq (qcdddr a) nil)
              (identp (qsecond a)))
          (setq x (qsecond a))
          (setq m (qthird a))
          (setq tmp1 (|get| x '|condition| env))
          (cond
            ((and tmp1 (consp tmp1) (eq (qrest tmp1) nil) (consp (qfirst tmp1))
              (eq (qcaar tmp1) '|or|) (|member| a (qcdr tmp1)))
             (setq oldpred (qcdr tmp1))
             (|put| x '|condition| (list (mkpf (|delete| a oldpred) '|or|) env))
             (t
              (setq tmp1 (|getUnionMode| x env))
              (setq zz (|delete| m (qrest tmp1)))
              (loop for u in zz
                    when (and (consp u) (eq (qfirst u) '|:|)
                              (consp (qrest u)) (equal (qsecond u) m))
```

```

do (setq zz (|delete| u zz)))
(setq newpred
  (mkpf (loop for mp in zz collect (list '|case| x mp)) 'or))
(|put| x '|condition|
  (cons newpred (|get| x '|condition| env)) env)))
(t env))))))

```

5.6.74 defun getUnionMode

[isUnionMode p311]
[getmode p??]

— defun getUnionMode —

```

(defun |getUnionMode| (x env)
  (let (m)
    (setq m (when (atom x) (|getmode| x env)))
    (when m (|isUnionMode| m env))))

```

5.6.75 defun isUnionMode

[getmode p??]
[get p??]

— defun isUnionMode —

```

(defun |isUnionMode| (m env)
  (let (mp v tmp1)
    (cond
      ((and (consp m) (eq (qfirst m) '|Union|)) m)
      ((progn
        (setq tmp1 (setq mp (|getmode| m env)))
        (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|)
          (consp (qrest tmp1)) (eq (qcddr tmp1) nil)
          (consp (qsecond tmp1))
          (eq (qcaadr tmp1) '|UnionCategory|)))
        (second mp))
      ((setq v (|get| (if (eq m '$) '|Rep| m) '|value| env))
        (when (and (consp (car v)) (eq (qfirst (car v)) '|Union|)) (car v))))))

```

5.6.76 defplist compImport plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|import| 'special) '|compImport|))
```

5.6.77 defun compImport

```
[addDomain p232]
[$NoValueMode p131]
```

— defun compImport —

```
(defun |compImport| (form mode env)
  (declare (ignore mode))
  (declare (special |$NoValueMode|))
  (dolist (dom (cdr form)) (setq env (|addDomain| dom env)))
  (list '|/throwAway| |$NoValueMode| env))
```

5.6.78 defplist compIs plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|is| 'special) '|compIs|))
```

5.6.79 defun compIs

```
[comp p557]
[coerce p346]
[$Boolean p??]
[$EmptyMode p131]
```

— defun compIs —

```
(defun |compIs| (form mode env)
  (let (a b aval am tmp1 bval bm td)
    (declare (special |$Boolean| |$EmptyMode|))
    (setq a (second form))
    (setq b (third form))
    (when (setq tmp1 (|comp| a |$EmptyMode| env))
      (setq aval (first tmp1))
      (setq am (second tmp1))
      (setq env (third tmp1))
      (when (setq tmp1 (|comp| b |$EmptyMode| env))
        (setq bval (first tmp1))
        (setq bm (second tmp1))
        (setq env (third tmp1))
        (setq td (list (list '|domainEqual| aval bval) |$Boolean| env ))
        (|coerce| td mode))))))
```

5.6.80 defplist compJoin plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| 'special) '|compJoin|))
```

5.6.81 defun compJoin

```
[nreverse0 p??]
[compForMode p315]
[stackSemanticError p??]
[nreverse0 p??]
[isCategoryForm p??]
[union p??]
[compJoin,getParms p??]
[qcar p??]
[qcdr p??]
[wrapDomainSub p274]
[convert p568]
[$Category p??]
```

— defun compJoin —

```

(defun |compJoin| (form mode env)
  (labels (
    (getParms (y env)
      (cond
        ((atom y)
          (when (|isDomainForm| y env) (list y)))
        ((and (consp y) (eq (qfirst y) 'length)
          (consp (qrest y)) (eq (qcddr y) nil))
          (list y (second y)))
        (t (list y))))))
    (let (arg1 catList pl tmp3 tmp4 tmp5 body parameters catListp td)
      (declare (special |$Category|))
      (setq arg1 (cdr form))
      (setq catList
        (dolist (x arg1 (nreverse0 tmp3))
          (push (car (or (|compForMode| x |$Category| env) (return '|failed|)))
            tmp3)))
      (cond
        ((eq catList '|failed|)
          (|stackSemanticError| (list '|cannot form Join of: | arg1) nil))
        (t
          (setq catListp
            (dolist (x catList (nreverse0 tmp4))
              (setq tmp4
                (cons
                  (cond
                    ((|isCategoryForm| x env)
                     (setq parameters
                       (|union|
                         (dolist (y (cdr x) tmp5)
                           (setq tmp5 (append tmp5 (getParms y env))))
                         parameters))
                    (x)
                    ((and (consp x) (eq (qfirst x) '|DomainSubstitutionMacro|)
                     (consp (qrest x)) (consp (qcddr x))
                     (eq (qcdddr x) nil))
                     (setq pl (second x))
                     (setq body (third x))
                     (setq parameters (|union| pl parameters)) body)
                    ((and (consp x) (eq (qfirst x) '|mkCategory|))
                     x)
                    ((and (atom x) (equal (|getmode| x env) |$Category|))
                     x)
                    (t
                     (|stackSemanticError| (list '|invalid argument to Join: | x) nil)
                     x))
                  tmp4))))
            (setq td (list (|wrapDomainSub| parameters (cons '|Join| catListp))
              |$Category| env))
            (|convert| td mode))))))

```

5.6.82 defun compForMode

```
[comp p557]
[$compForModeIfTrue p??]
```

— defun compForMode —

```
(defun |compForMode| (x m e)
  (let (|$compForModeIfTrue|)
    (declare (special |$compForModeIfTrue|))
    (setq |$compForModeIfTrue| t)
    (|comp| x m e)))
```

5.6.83 defplist compLambda plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|+>| 'special) '|compLambda|))
```

5.6.84 defun compLambda

```
[qcar p??]
[qcdr p??]
[argsToSig p592]
[compAtSign p352]
[stackAndThrow p??]
```

— defun compLambda —

```
(defun |compLambda| (form mode env)
  (let (v1 body tmp1 tmp2 tmp3 target args arg1 sig1 ress)
    (setq v1 (second form))
    (setq body (third form))))
```

```

(cond
  ((and (consp vl) (eq (qfirst vl) '|:|)
    (progn
      (setq tmp1 (qrest vl))
      (and (consp tmp1)
        (progn
          (setq args (qfirst tmp1))
          (setq tmp2 (qrest tmp1))
          (and (consp tmp2)
            (eq (qrest tmp2) nil)
            (progn
              (setq target (qfirst tmp2))
              t))))))
    (when (and (consp args) (eq (qfirst args) '|@Tuple|))
      (setq args (qrest args)))
    (cond
      ((listp args)
        (setq tmp3 (|argsToSig| args))
        (setq arg1 (first tmp3))
        (setq sig1 (second tmp3))
        (cond
          (sig1
            (setq res
              (compAtSign
                (list '@
                  (list '+-> arg1 body)
                  (cons '|Mapping| (cons target sig1))) mode env))
              res)
            (t (|stackAndThrow| (list '|compLambda| form )))))
          (t (|stackAndThrow| (list '|compLambda| form )))))
      (t (|stackAndThrow| (list '|compLambda| form ))))))

```

5.6.85 defplist compLeave plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|leave| 'special) '|compLeave|))

```

5.6.86 defun compLeave

```
[comp p557]
[modifyModeStack p593]
[$exitModeStack p??]
[$leaveLevelStack p??]
```

— **defun compLeave** —

```
(defun |compLeave| (form mode env)
  (let (level x index u)
    (declare (special |$exitModeStack| |$leaveLevelStack|))
    (setq level (second form))
    (setq x (third form))
    (setq index
      (- (1- (|#| |$exitModeStack|)) (elt |$leaveLevelStack| (1- level))))
    (when (setq u (|comp| x (elt |$exitModeStack| index) env))
      (|modifyModeStack| (second u) index)
      (list (list 'TAGGEDexit| index u) mode env )))))
```

5.6.87 defplist compMacro plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get 'mdef 'special) '|compMacro|))
```

5.6.88 defun compMacro

```
[qcar p??]
[formatUnabbreviated p??]
[sayBrightly p??]
[put p??]
[macroExpand p136]
[$macroIfTrue p??]
[$NoValueMode p131]
[$EmptyMode p131]
```

— **defun compMacro** —

```

(defun |compMacro| (form mode env)
  (let (|$macroIfTrue| lhs signature specialCases rhs prhs)
    (declare (special |$macroIfTrue| |$NoValueMode| |$EmptyMode|))
    (setq |$macroIfTrue| t)
    (setq lhs (second form))
    (setq signature (third form))
    (setq specialCases (fourth form))
    (setq rhs (fifth form))
    (setq prhs
      (cond
        ((and (consp rhs) (eq (qfirst rhs) 'category))
         (list "-- the constructor category"))
        ((and (consp rhs) (eq (qfirst rhs) '|Join|))
         (list "-- the constructor category"))
        ((and (consp rhs) (eq (qfirst rhs) 'capsule))
         (list "-- the constructor capsule"))
        ((and (consp rhs) (eq (qfirst rhs) '|add|))
         (list "-- the constructor capsule"))
        (t (|formatUnabbreviated| rhs))))
    (|sayBrightly|
     (cons "   processing macro definition"
       (cons '|%b|
         (append (|formatUnabbreviated| lhs)
           (cons " ==> "
             (append prhs (list '|%d|))))))))
    (when (or (equal mode |$EmptyMode|) (equal mode |$NoValueMode|))
      (list '|/throwAway| |$NoValueMode|
        (|put| (CAR lhs) '|macro| (|macroExpand| rhs env) env))))))

```

5.6.89 defplist compPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| 'special) '|compPretend|))

```

5.6.90 defun compPretend

[addDomain p232]
 [comp p557]

```
[opOf p??]
[nequal p??]
[stackSemanticError p??]
[stackWarning p??]
[$newCompilerUnionFlag p??]
[$EmptyMode p131]
```

— **defun compPretend** —

```
(defun |compPretend| (form mode env)
  (let (x tt warningMessage td tp)
    (declare (special |$newCompilerUnionFlag| |$EmptyMode|))
    (setq x (second form))
    (setq tt (third form))
    (setq env (|addDomain| tt env))
    (when (setq td (or (|comp| x tt env) (|comp| x |$EmptyMode| env)))
      (when (equal (second td) tt)
        (setq warningMessage (list '|pretend| tt '| -- should replace by @|)))
      (cond
        ((and |$newCompilerUnionFlag|
              (eq (|opOf| (second td)) '|Union|)
              (nequal (|opOf| mode) '|Union|))
          (|stackSemanticError|
           (list '|cannot pretend | x '| of mode | (second td) '| to mode | mode)
           nil))
        (t
         (setq td (list (first td) tt (third td)))
         (when (setq tp (|coerce| td mode))
           (when warningMessage (|stackWarning| warningMessage)
             tp))))))
```

—————

5.6.91 defplist compQuote plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get 'quote 'special) '|compQuote|))
```

—————

5.6.92 defun compQuote

— defun compQuote —

```
(defun |compQuote| (form mode env)
  (list form mode env))
```

—————

5.6.93 defplist compReduce plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'reduce 'special) '|compReduce|))
```

—————

5.6.94 defun compReduce

```
[compReduce1 p320]
[$formalArgList p??]
```

— defun compReduce —

```
(defun |compReduce| (form mode env)
  (declare (special |$formalArgList|))
  (|compReduce1| form mode env |$formalArgList|))
```

—————

5.6.95 defun compReduce1

```
[systemError p??]
[nreverse0 p??]
[compIterator p??]
[comp p557]
[parseTran p93]
[getIdentity p??]
[msubst p??]
```

```

[$sideEffectsList p??]
[$until p??]
[$initList p??]
[$Boolean p??]
[$e p??]
[$endTestList p??]

```

— **defun compReduce1** —

```

(defun |compReduce1| (form mode env |$formalArgList|)
  (declare (special |$formalArgList|))
  (let (|$sideEffectsList| |$until| |$initList| |$endTestList| collectForm
        collectOp body op itl acc afterFirst bodyVal part1 part2 part3 id
        identityCode untilCode finalCode tmp1 tmp2)
    (declare (special |$sideEffectsList| |$until| |$initList| |$Boolean| |$e|
                      |$endTestList|))
    (setq op (second form))
    (setq collectForm (fourth form))
    (setq collectOp (first collectForm))
    (setq tmp1 (reverse (cdr collectForm)))
    (setq body (first tmp1))
    (setq itl (nreverse (cdr tmp1)))
    (when (stringp op) (setq op (intern op)))
    (cond
      ((null (member collectOp '(collect collectv collectvec)))
       (|systemError| (list '|illegal reduction form:| form)))
      (t
       (setq |$sideEffectsList| nil)
       (setq |$until| nil)
       (setq |$initList| nil)
       (setq |$endTestList| nil)
       (setq |$e| env)
       (setq itl
        (dolist (x itl (nreverse0 tmp2))
          (setq tmp1 (or (|compIterator| x |$e|) (return '|failed|)))
          (setq |$e| (second tmp1))
          (push (elt tmp1 0) tmp2)))
        (unless (eq itl '|failed|)
          (setq env |$e|)
          (setq acc (gensym))
          (setq afterFirst (gensym))
          (setq bodyVal (gensym))
          (when (setq tmp1 (|comp| (list 'let bodyVal body ) mode env))
            (setq part1 (first tmp1))
            (setq mode (second tmp1))
            (setq env (third tmp1))
            (when (setq tmp1 (|comp| (list 'let acc bodyVal) mode env))
              (setq part2 (first tmp1))
              (setq env (third tmp1))

```

```

(when (setq tmp1
  (|comp| (list 'let acc (|parseTran| (list op acc bodyVal)))
    mode env))
  (setq part3 (first tmp1))
  (setq env (third tmp1))
  (when (setq identityCode
    (if (setq id (|getIdentity| op env))
      (car (|comp| id mode env))
      (list '|IdentityError| (mkq op))))
    (setq finalCode
      (cons 'progn
        (cons (list 'let afterFirst nil)
          (cons
            (cons 'repeat
              (append itl
                (list
                  (list 'progn part1
                    (list 'if afterFirst part3
                      (list 'progn part2 (list 'let afterFirst (mkq t)))) nil))))
              (list (list 'if afterFirst acc identityCode ))))))
          (when |$until|
            (setq tmp1 (|comp| |$until| |$Boolean| env))
            (setq untilCode (first tmp1))
            (setq env (third tmp1))
            (setq finalCode
              (msubst (list 'until untilCode) '|$until| finalCode)))
            (list finalCode mode env ))))))))

```

5.6.96 defplist compRepeatOrCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'collect 'special) '|compRepeatOrCollect|))

```

5.6.97 defplist compRepeatOrCollect plist

— postvars —

```

(eval-when (eval load)

```

```
(setf (get 'repeat 'special) '|compRepeatOrCollect|))
```

5.6.98 defun compRepeatOrCollect

```
[length p??]
[compIterator p??]
[modeIsAggregateOf p??]
[stackMessage p??]
[compOrCroak p556]
[comp p557]
[msubst p??]
[coerceExit p351]
[ p??]
[ p??]
[$until p??]
[$Boolean p??]
[$NoValueMode p131]
[$exitModeStack p??]
[$leaveLevelStack p??]
[$formalArgList p??]
```

— defun compRepeatOrCollect —

```
(defun |compRepeatOrCollect| (form mode env)
  (labels (
    (fn (form |$exitModeStack| |$leaveLevelStack| |$formalArgList| env)
      (declare (special |$exitModeStack| |$leaveLevelStack| |$formalArgList|))
      (let (|$until| body itl xp targetMode repeatOrCollect bodyMode bodyp mp tmp1
            untilCode ep itlp formp u mpp tmp2)
        (declare (special |$Boolean| |$until| |$NoValueMode| ))
        (setq |$until| nil)
        (setq repeatOrCollect (car form))
        (setq tmp1 (reverse (cdr form)))
        (setq body (car tmp1))
        (setq itl (nreverse (cdr tmp1)))
        (setq itlp
          (dolist (x itl (nreverse0 tmp2))
            (setq tmp1 (or (|compIterator| x env) (return '|failed|)))
            (setq xp (first tmp1))
            (setq env (second tmp1))
            (push xp tmp2))))
        (unless (eq itlp '|failed|)
          (setq targetMode (car |$exitModeStack|))
          (setq bodyMode
```

```

(if (eq repeatOrCollect 'collect)
  (cond
    ((eq targetMode '|$EmptyMode|)
     '|$EmptyMode|)
    ((setq u (|modeIsAggregateOf| '|List| targetMode env))
     (second u))
    ((setq u (|modeIsAggregateOf| '|PrimitiveArray| targetMode env))
     (setq repeatOrCollect 'collectv)
     (second u))
    ((setq u (|modeIsAggregateOf| '|Vector| targetMode env))
     (setq repeatOrCollect 'collectvec)
     (second u))
    (t
     (|stackMessage| "Invalid collect bodytype")
     '|failed|))
    |$NoValueMode|))
(unless (eq bodyMode '|failed|)
  (when (setq tmp1 (|compOrCroak| body bodyMode env))
    (setq bodyp (first tmp1))
    (setq mp (second tmp1))
    (setq ep (third tmp1))
    (when |$until|
      (setq tmp1 (|comp| |$until| |$Boolean| ep))
      (setq untilCode (first tmp1))
      (setq ep (third tmp1))
      (setq itlp (msubst (list 'until untilCode) '|$until| itlp)))
    (setq formp (cons repeatOrCollect (append itlp (list bodyp))))
    (setq mpp
      (cond
        ((eq repeatOrCollect 'collect)
         (if (setq u (|modeIsAggregateOf| '|List| targetMode env))
             (car u)
             (list '|List| mp)))
        ((eq repeatOrCollect 'collectv)
         (if (setq u (|modeIsAggregateOf| '|PrimitiveArray| targetMode env))
             (car u)
             (list '|PrimitiveArray| mp)))
        ((eq repeatOrCollect 'collectvec)
         (if (setq u (|modeIsAggregateOf| '|Vector| targetMode env))
             (car u)
             (list '|Vector| mp)))
        (t mp)))
      (|coerceExit| (list formp mpp ep) targetMode))))))
(declare (special |$exitModeStack| |$leaveLevelStack| |$formalArgList|))
(fn form
  (cons mode |$exitModeStack|)
  (cons (|#| |$exitModeStack|) |$leaveLevelStack|)
  |$formalArgList|
  env)))

```

5.6.99 defplist compReturn plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|return| 'special) '|compReturn|))
```

5.6.100 defun compReturn

```
[stackSemanticError p??]
[nequal p??]
[userError p??]
[resolve p356]
[comp p557]
[modifyModeStack p593]
[$exitModeStack p??]
[$returnMode p??]
```

— defun compReturn —

```
(defun |compReturn| (form mode env)
  (let (level x index u xp mp ep)
    (declare (special |$returnMode| |$exitModeStack|))
    (setq level (second form))
    (setq x (third form))
    (cond
      ((null |$exitModeStack|)
       (|stackSemanticError|
        (list '|the return before| '|%b| x '|%d| '|is unnecessary|) nil)
       nil)
      ((nequal level 1)
       (|userError| "multi-level returns not supported"))
      (t
       (setq index (max 0 (1- (|#| |$exitModeStack|))))
       (when (>= index 0)
        (setq |$returnMode|
              (|resolve| (elt |$exitModeStack| index) |$returnMode|)))
       (when (setq u (|comp| x |$returnMode| env))
        (setq xp (first u))
        (setq mp (second u))
        (setq ep (third u))
```

```
(when (>= index 0)
  (setq |$returnMode| (|resolve| mp |$returnMode|))
  (|modifyModeStack| mp index))
(list (list ' |TAGGEDreturn| 0 u) mode ep))))))
```

5.6.101 defplist compSeq plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'seq 'special) '|compSeq|))
```

5.6.102 defun compSeq

```
[compSeq1 p326]
[$exitModeStack p??]
```

— defun compSeq —

```
(defun |compSeq| (form mode env)
  (declare (special |$exitModeStack|))
  (|compSeq1| (cdr form) (cons mode |$exitModeStack|) env))
```

5.6.103 defun compSeq1

```
[nreverse0 p??]
[compSeqItem p328]
[mkq p??]
[replaceExitEtc p327]
[$exitModeStack p??]
[$insideExpressionIfTrue p??]
[$finalEnv p??]
[$NoValueMode p131]
```

— defun compSeq1 —

```
(defun |compSeq1| (form |$exitModeStack| env)
  (declare (special |$exitModeStack|))
  (let (|$insideExpressionIfTrue| |$finalEnv| tmp1 tmp2 c catchTag newform)
    (declare (special |$insideExpressionIfTrue| |$finalEnv| |$NoValueMode|))
    (setq |$insideExpressionIfTrue| nil)
    (setq |$finalEnv| nil)
    (when
      (setq c (dolist (x form (nreverse0 tmp2))
        (setq |$insideExpressionIfTrue| nil)
        (setq tmp1 (|compSeqItem| x |$NoValueMode| env))
        (unless tmp1 (return nil))
        (setq env (third tmp1))
        (push (first tmp1) tmp2)))
      (setq catchTag (mkq (gensym)))
      (setq newform
        (cons 'seq
          (|replaceExitEtc| c catchTag '|TAGGEDexit| (elt |$exitModeStack| 0))))
      (list (list 'catch catchTag newform)
        (elt |$exitModeStack| 0) |$finalEnv|))))
```

5.6.104 defun replaceExitEtc

```
[qcar p??]
[qcdr p??]
[rplac p??]
[replaceExitEtc p327]
[intersectionEnvironment p??]
[convertOrCroak p328]
[$finalEnv p??]
[$finalEnv p??]
```

— defun replaceExitEtc —

```
(defun |replaceExitEtc| (x tag opFlag opMode)
  (declare (special |$finalEnv|))
  (cond
    ((atom x) nil)
    ((and (consp x) (eq (qfirst x) 'quote)) nil)
    ((and (consp x) (equal (qfirst x) opFlag) (consp (qrest x))
      (consp (qcddr x)) (eq (qcddr x) nil))
      (|rplac| (caaddr x) (|replaceExitEtc| (caaddr x) tag opFlag opMode))
      (cond
        ((eq1 (second x) 0)
          (setq |$finalEnv|
            (if |$finalEnv|
```

```

(|intersectionEnvironment| |$finalEnv| (third (third x)))
  (third (third x)))
(|rplac| (car x) 'throw)
(|rplac| (cadr x) tag)
(|rplac| (caddr x) (car (|convertOrCroak| (caddr x) opMode))))
(t
  (|rplac| (cadr x) (1- (cadr x)))))
((and (consp x) (consp (qrest x)) (consp (qcddr x))
  (eq (qcddr x) nil)
  (member (qfirst x) '(|TAGGEDreturn| |TAGGEDexit|))))
(|rplac| (car (caddr x))
  (|replaceExitEtc| (car (caddr x)) tag opFlag opMode)))
(t
  (|replaceExitEtc| (car x) tag opFlag opMode)
  (|replaceExitEtc| (cdr x) tag opFlag opMode)))
x)

```

5.6.105 defun convertOrCroak

```

[convert p568]
[userError p??]

```

— defun convertOrCroak —

```

(defun |convertOrCroak| (tt m)
  (let (u)
    (if (setq u (|convert| tt m))
      u
      (|userError|
        (list '|CANNOT CONVERT: | (first tt) '|%1| '| OF MODE: | (second tt)
          '|%1| '| TO MODE: | m '|%1|))))))

```

5.6.106 defun compSeqItem

```

[comp p557]
[macroExpand p136]

```

— defun compSeqItem —

```

(defun |compSeqItem| (form mode env)
  (|comp| (|macroExpand| form env) mode env))

```

5.6.107 defplist compSetq plist**— postvars —**

```
(eval-when (eval load)
  (setf (get 'let 'special) '|compSetq|))
```

5.6.108 defplist compSetq plist**— postvars —**

```
(eval-when (eval load)
  (setf (get 'setq 'special) '|compSetq|))
```

5.6.109 defun compSetq

[compSetq1 p329]

— defun compSetq —

```
(defun |compSetq| (form mode env)
  (|compSetq1| (second form) (third form) mode env))
```

5.6.110 defun compSetq1

```
[setqSingle p334]
[compSetq1 identp (vol5)]
[compMakeDeclaration p593]
[compSetq p329]
[qcar p??]
[qcdr p??]
```

[setqMultiple p330]
 [setqSetelt p334]
 [\$EmptyMode p131]

— **defun compSetq1** —

```
(defun |compSetq1| (form val mode env)
  (let (x y ep op z)
    (declare (special |$EmptyMode|))
    (cond
      ((identp form) (|setqSingle| form val mode env))
      ((and (consp form) (eq (qfirst form) '|:|) (consp (qrest form))
        (consp (qcddr form)) (eq (qcdddr form) nil))
        (setq x (second form))
        (setq y (third form))
        (setq ep (third (|compMakeDeclaration| form |$EmptyMode| env)))
        (|compSetq| (list 'let x val) mode ep))
      ((consp form)
        (setq op (qfirst form))
        (setq z (qrest form))
        (cond
          ((eq op 'cons) (|setqMultiple| (|uncons| form) val mode env))
          ((eq op '|@Tuple|) (|setqMultiple| z val mode env))
          (t (|setqSetelt| form val mode env))))))
```

5.6.111 defun uncons

[uncons p330]

— **defun uncons** —

```
(defun |uncons| (x)
  (cond
    ((atom x) x)
    ((and (consp x) (eq (qfirst x) 'cons) (consp (qrest x))
      (consp (qcddr x)) (eq (qcdddr x) nil))
      (cons (second x) (|uncons| (third x)))))
```

5.6.112 defun setqMultiple

[nreverse0 p??]
 [qcar p??]

```

[qcdr p??]
[stackMessage p??]
[setqMultipleExplicit p333]
[genVariable p??]
[addBinding p??]
[compSetq1 p329]
[convert p568]
[put p??]
[genSomeVariable p??]
[length p??]
[mkprogn p??]
[$EmptyMode p131]
[$NoValueMode p131]
[$noEnv p??]

```

— defun setqMultiple —

```

(defun |setqMultiple| (nameList val m env)
  (labels (
    (decompose (tt len env)
      (declare (ignore len))
      (let (tmp1 z)
        (declare (special |$EmptyMode|))
        (cond
          ((and (consp tt) (eq (qfirst tt) '|Record|)
            (progn (setq z (qrest tt)) t))
          (loop for item in z
            collect (cons (second item) (third item))))
        ((progn
          (setq tmp1 (|comp| tt |$EmptyMode| env))
          (and (consp tmp1) (consp (qrest tmp1)) (consp (qsecond tmp1))
            (eq (qcaadr tmp1) '|RecordCategory|)
            (consp (qcddr tmp1)) (eq (qcdddr tmp1) nil)))
          (loop for item in z
            collect (cons (second item) (third item))))
        (t (|stackMessage| (list '|no multiple assigns to mode: | tt))))))
    (let (g m1 tt x mp selectorModePairs tmp2 assignList)
      (declare (special |$noEnv| |$EmptyMode| |$NoValueMode|))
      (cond
        ((and (consp val) (eq (qfirst val) 'cons) (equal m |$NoValueMode|)
          (|setqMultipleExplicit| nameList (|uncons| val) m env))
        ((and (consp val) (eq (qfirst val) '|@Tuple|) (equal m |$NoValueMode|)
          (|setqMultipleExplicit| nameList (qrest val) m env))
        ; 1 create a gensym, %add to local environment, compile and assign rhs
        (t
          (setq g (|genVariable|))
          (setq env (|addBinding| g nil env))
          (setq tmp2 (|compSetq1| g val |$EmptyMode| env))

```

```

      (when tmp2
        (setq tt tmp2)
        (setq m1 (cadr tmp2))
        (setq env (|put| g 'mode m1 env))
        (setq tmp2 (|convert| tt m))
; 1.1 --exit if result is a list
      (when tmp2
        (setq x (first tmp2))
        (setq mp (second tmp2))
        (setq env (third tmp2))
        (cond
          ((and (consp m1) (eq (qfirst m1) '|List|) (consp (qrest m1))
              (eq (qcddr m1) nil))
            (loop for y in nameList do
              (setq env
                (|put| y '|value| (list (|genSomeVariable|) (second m1) |$noEnv|)
                  env)))
            (|convert| (list (list 'progn x (list 'let nameList g) g) mp env) m))
          (t
            (t
; 2 --verify that the #nameList = number of parts of right-hand-side
              (setq selectorModePairs
                (decompose m1 (|#| nameList) env))
              (when selectorModePairs
                (cond
                  ((nequal (|#| nameList) (|#| selectorModePairs))
                    (|stackMessage|
                     (list val '| must decompose into |
                       (|#| nameList) '| components| )))
                  (t
; 3 --generate code
                    (setq assignList
                      (loop for x in nameList
                        for item in selectorModePairs
                        collect (car
                          (progn
                            (setq tmp2
                              (or (|compSetq1| x (list '|elt| g (first item))
                                (rest item) env)
                                (return '|failed|))))
                            (setq env (third tmp2))
                            tmp2))))
                      (unless (eq assignList '|failed|)
                        (list (mkprogn (cons x (append assignList (list g)))) mp env)
                        ))))))))

```

5.6.113 defun setqMultipleExplicit

```
[nequal p??]
[stackMessage p??]
[genVariable p??]
[compSetq1 p329]
[last p??]
[$EmptyMode p131]
[$NoValueMode p131]
```

— **defun setqMultipleExplicit** —

```
(defun |setqMultipleExplicit| (nameList vallList m env)
  (declare (ignore m))
  (let (gensymList assignList tmp1 reAssignList)
    (declare (special |$NoValueMode| |$EmptyMode|))
    (cond
      ((nequal (|#| nameList) (|#| vallList))
        (|stackMessage|
          (list '|Multiple assignment error; # of items in: | nameList
                '|must = # in: | vallList)))
      (t
        (setq gensymList
          (loop for name in nameList
            collect (|genVariable|)))
        (setq assignList
          (loop for g in gensymList
            for val in vallList
            collect (progn
              (setq tmp1
                (or (|compSetq1| g val |$EmptyMode| env)
                  (return '|failed|)))
              (setq env (third tmp1))
              tmp1)))
        (unless (eq assignList '|failed|)
          (setq reAssignList
            (loop for g in gensymList
              for name in nameList
              collect (progn
                (setq tmp1
                  (or (|compSetq1| name g |$EmptyMode| env)
                    (return '|failed|)))
                (setq env (third tmp1))
                tmp1)))
          (unless (eq reAssignList '|failed|)
            (list
              (cons 'progn
                (append
                  (loop for tt in assignList
```

```

collect (car tt))
(loop for tt in reAssignList
  collect (car tt)))
|$NoValueMode| (third (|last| reAssignList)))))))))

```

5.6.114 defun setqSetelt

[comp p557]

— defun setqSetelt —

```

(defun |setqSetelt| (form val mode env)
  (|comp| (cons '|setelt| (cons (car form) (append (cdr form) (list val))))
    mode env))

```

5.6.115 defun setqSingle

```

[setqSingle getProplist (vol5)]
[getmode p??]
[get p??]
[nequal p??]
[maxSuperType p338]
[comp p557]
[getmode p??]
[assignError p336]
[convert p568]
[setqSingle identp (vol5)]
[profileRecord p??]
[consProplistOf p??]
[removeEnv p??]
[setqSingle addBinding (vol5)]
[isDomainForm p338]
[isDomainInScope p??]
[stackWarning p??]
[augModemapsFromDomain1 p237]
[NRTassocIndex p336]
[isDomainForm p338]
[outputComp p337]
[$insideSetqSingleIfTrue p??]

```

```

[$QuickLet p??]
[$form p??]
[$profileCompiler p??]
[$EmptyMode p131]
[$NoValueMode p131]

```

— defun `setqSingle` —

```

(defun |setqSingle| (form val mode env)
  (let (|$insideSetqSingleIfTrue| currentProplist mpp maxmpp td x mp tp key
        newProplist ep k newform)
    (declare (special |$insideSetqSingleIfTrue| |$QuickLet| |$form|
                      |$profileCompiler| |$EmptyMode| |$NoValueMode|))
    (setq |$insideSetqSingleIfTrue| t)
    (setq currentProplist (|getProplist| form env))
    (setq mpp
      (or (|get| form '|mode| env) (|getmode| form env)
          (if (equal mode |$NoValueMode|) |$EmptyMode| mode)))
    (when (setq td
      (cond
        ((setq td (|comp| val mpp env))
         td)
        ((and (null (|get| form '|mode| env))
              (nequal mpp (setq maxmpp (|maxSuperType| mpp env)))
              (setq td (|comp| val maxmpp env)))
         td)
        ((and (setq td (|comp| val |$EmptyMode| env))
              (|getmode| (second td) env))
         (|assignError| val (second td) form mpp))))
      (when (setq tp (|convert| td mode))
        (setq x (first tp))
        (setq mp (second tp))
        (setq ep (third tp))
        (when (and |$profileCompiler| (identp form))
          (setq key (if (member form (cdr |$form|)) '|arguments| '|locals|)
                    (|profileRecord| key form (second td)))
          (setq newProplist
            (|consProplistOf| form currentProplist '|value|
                              (|removeEnv| (cons val (cdr td)))))
          (setq ep (if (consp form) ep (|addBinding| form newProplist ep)))
          (when (|isDomainForm| val ep)
            (when (|isDomainInScope| form ep)
              (|stackWarning|
                (list '|domain valued variable| '|%b| form '|%d|
                      '|has been reassigned within its scope| )))
              (setq ep (|augModemapsFromDomain1| form val ep)))
          (if (setq k (|NRTassocIndex| form))
              (setq newform (list 'setelt '$ k x))
              (setq newform

```

```

(if |$QuickLet|
  (list 'let form x)
  (list 'let form x
    (if (|isDomainForm| x ep)
      (list 'elt form 0)
      (car (|outputComp| form ep))))))
(list newform mp ep))))

```

5.6.116 defun NRTassocIndex

This function returns the index of domain entry *x* in the association list [*\$NRTaddForm p??*]

[*\$NRTdeltaList p??*]

[*\$found p??*]

[*\$NRTbase p??*]

[*\$NRTdeltaLength p??*]

— defun NRTassocIndex —

```

(defun |NRTassocIndex| (x)
  (let (k (i 0))
    (declare (special |$NRTdeltaLength| |$NRTbase| |$found| |$NRTdeltaList|
      |$NRTaddForm|))
    (cond
      ((null x) x)
      ((equal x |$NRTaddForm|) 5)
      ((setq k
        (let (result)
          (loop for y in |$NRTdeltaList|
            when (and (incf i)
              (eq (elt y 0) '|domain|)
              (equal (elt y 1) x)
              (setq |$found| y))
            do (setq result (or result i)))
          result))
        (- (+ |$NRTbase| |$NRTdeltaLength|) k))
      (t nil))))

```

5.6.117 defun assignError

[*stackMessage p??*]

— defun assignError —

```
(defun |assignError| (val mp form m)
  (let (message)
    (setq message
      (if val
        (list '|CANNOT ASSIGN: | val '|%1|
              '| OF MODE: | mp '|%1|
              '| TO: | form '|%1| '| OF MODE: | m)
        (list '|CANNOT ASSIGN: | val '|%1|
              '| TO: | form '|%1| '| OF MODE: | m)))
      (|stackMessage| message)))
```

5.6.118 defun outputComp

```
[comp p557]
[qcar p??]
[qcdr p??]
[nreverse0 p??]
[outputComp p337]
[get p??]
[$Expression p??]
```

— defun outputComp —

```
(defun |outputComp| (x env)
  (let (arg1 v)
    (declare (special |$Expression|))
    (cond
      ((|comp| (list '|::| x |$Expression|) |$Expression| env))
      ((and (consp x) (eq (qfirst x) '|construct|))
        (setq arg1 (qrest x))
        (list (cons 'list
          (let (result tmp1)
            (loop for x in arg1
              do (setq result
                (cons (car
                  (progn
                    (setq tmp1 (|outputComp| x env))
                    (setq env (third tmp1))
                    tmp1))
                  result))))
            (nreverse0 result)))
          |$Expression| env))
      ((and (setq v (|get| x '|value| env))
        (consp (cadr v)) (eq (qfirst (cadr v)) '|Union|))
        (list (list '|coerceUn2E| x (cadr v)) |$Expression| env)))
```

```
(t (list x |$Expression| env))))))
```

5.6.119 defun maxSuperType

```
[get p??]  
[maxSuperType p338]
```

— defun maxSuperType —

```
(defun |maxSuperType| (m env)  
  (let (typ)  
    (if (setq typ (|get| m '|SuperDomain| env))  
        (|maxSuperType| typ env)  
        m)))
```

5.6.120 defun isDomainForm

```
[kar p??]  
[qcar p??]  
[qcdr p??]  
[isFunctor p233]  
[isCategoryForm p??]  
[isDomainConstructorForm p339]  
[$SpecialDomainNames p??]
```

— defun isDomainForm —

```
(defun |isDomainForm| (d env)  
  (let (tmp1)  
    (declare (special |$SpecialDomainNames|))  
    (or (member (kar d) |$SpecialDomainNames|) (|isFunctor| d)  
        (and (progn  
              (setq tmp1 (|getmode| d env))  
              (and (consp tmp1) (eq (qfirst tmp1) '|Mapping|) (consp (qrest tmp1))))  
              (|isCategoryForm| (qsecond tmp1) env))  
          (|isCategoryForm| (|getmode| d env) env)  
          (|isDomainConstructorForm| d env))))))
```

5.6.121 defun isDomainConstructorForm

```
[qcar p??]
[qcdr p??]
[isCategoryForm p??]
[eqsubstlist p??]
[$FormalMapVariableList p250]
```

— **defun isDomainConstructorForm** —

```
(defun |isDomainConstructorForm| (d env)
  (let (u)
    (declare (special |$FormalMapVariableList|))
    (when
      (and (consp d)
           (setq u (|get| (qfirst d) '|value| env))
           (consp u)
           (consp (qrest u))
           (consp (qsecond u))
           (eq (qcaadr u) '|Mapping|)
           (consp (qcdadr u)))
      (|isCategoryForm|
       (eqsubstlist (rest d) |$FormalMapVariableList| (cadadr u)) env))))
```

—————

5.6.122 defplist compString plist

— **postvars** —

```
(eval-when (eval load)
  (setf (get '|String| 'special) '|compString|))
```

—————

5.6.123 defun compString

```
[resolve p356]
[$StringCategory p??]
```

— **defun compString** —

```
(defun |compString| (form mode env)
```

```
(declare (special |$StringCategory|))
(list form (|resolve| |$StringCategory| mode) env))
```

5.6.124 defplist compSubDomain plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|SubDomain| 'special) '|compSubDomain|))
```

5.6.125 defun compSubDomain

```
[compSubDomain1 p341]
[compCapsule p258]
[$addFormLhs p??]
[$NRTaddForm p??]
[$addForm p??]
[$addFormLhs p??]
```

— defun compSubDomain —

```
(defun |compSubDomain| (form mode env)
  (let (|$addFormLhs| |$addForm| domainForm predicate tmp1)
    (declare (special |$addFormLhs| |$addForm| |$NRTaddForm| |$addFormLhs|))
    (setq domainForm (second form))
    (setq predicate (third form))
    (setq |$addFormLhs| domainForm)
    (setq |$addForm| nil)
    (setq |$NRTaddForm| domainForm)
    (setq tmp1 (|compSubDomain1| domainForm predicate mode env))
    (setq |$addForm| (first tmp1))
    (setq env (third tmp1))
    (|compCapsule| (list 'capsule) mode env)))
```

5.6.126 defun compSubDomain1

```
[compMakeDeclaration p593]
[addDomain p232]
[compOrCroak p556]
[stackSemanticError p??]
[lispize p342]
[evalAndRwriteLispForm p169]
[$CategoryFrame p??]
[$op p??]
[$lisplibSuperDomain p??]
[$Boolean p??]
[$EmptyMode p131]
```

— **defun compSubDomain1** —

```
(defun |compSubDomain1| (domainForm predicate mode env)
  (let (u prefixPredicate opp dFp)
    (declare (special |$CategoryFrame| |$op| |$lisplibSuperDomain| |$Boolean|
                      |$EmptyMode|))
    (setq env (third
      (|compMakeDeclaration| (list '|:| '|#1| domainForm)
        |$EmptyMode| (|addDomain| domainForm env))))
    (setq u (|compOrCroak| predicate |$Boolean| env))
    (unless u
      (|stackSemanticError|
        (list '|predicate: | predicate
              '| cannot be interpreted with #1: | domainForm) nil))
    (setq prefixPredicate (|lispize| (first u)))
    (setq |$lisplibSuperDomain| (list domainForm predicate))
    (|evalAndRwriteLispForm| '|evalOnLoad2|
      (list 'setq '|$CategoryFrame|
        (list '|put|
          (setq opp (list 'quote |$op|))
          '|SuperDomain|
          (setq dFp (list 'quote domainForm))
          (list '|put| dFp '|SubDomain|
            (list 'cons (list 'quote (cons |$op| prefixPredicate))
              (list 'delasc opp (list '|get| dFp '|SubDomain| '|$CategoryFrame|))))
          '|$CategoryFrame|))))
      (list domainForm mode env)))
```

5.6.127 defun lispize

[optimize p209]

— defun lispize —

```
(defun |lispize| (x)
  (car (|optimize| (list x))))
```

—————

5.6.128 defplist compSubsetCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|SubsetCategory| 'special) '|compSubsetCategory|))
```

—————

5.6.129 defun compSubsetCategory**TPDHERE:** See LocalAlgebra for an example call [put p??]

[comp p557]

[msubst p??]

[\$lhsOfColon p??]

— defun compSubsetCategory —

```
(defun |compSubsetCategory| (form mode env)
  (let (cat r)
    (declare (special |$lhsOfColon|))
    (setq cat (second form))
    (setq r (third form))
    ; --1. put "Subsets" property on R to allow directly coercion to subset;
    ; -- allow automatic coercion from subset to R but not vice versa
    (setq env (|put| r '|Subsets| (list (list |$lhsOfColon| '|isFalse|)) env))
    ; --2. give the subset domain modemaps of cat plus 3 new functions
    (|comp|
      (list '|Join| cat
        (msubst |$lhsOfColon| '$
          (list 'category '|domain|
            (list 'signature '|coerce| (list r '$))
            (list 'signature '|lift| (list r '$))
```

```
(list 'signature '|reduce| (list '$ r))))
mode env)))
```

5.6.130 defplist compSuchthat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '\| 'special) '|compSuchthat|))
```

5.6.131 defun compSuchthat

```
[comp p557]
[put p??]
[$Boolean p??]
```

— defun compSuchthat —

```
(defun |compSuchthat| (form mode env)
  (let (x p xp mp tmp1 pp)
    (declare (special |$Boolean|))
    (setq x (second form))
    (setq p (third form))
    (when (setq tmp1 (|comp| x mode env))
      (setq xp (first tmp1))
      (setq mp (second tmp1))
      (setq env (third tmp1))
      (when (setq tmp1 (|comp| p |$Boolean| env))
        (setq pp (first tmp1))
        (setq env (third tmp1))
        (setq env (|put| xp '|condition| pp env))
        (list xp mp env))))))
```

5.6.132 defplist compVector plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'vector 'special) '|compVector|))
```

5.6.133 defun compVector

```
; null l => [$EmptyVector,m,e]
; T1:= [[.,mUnder,e]:= comp(x,mUnder,e) or return "failed" for x in l]
; T1="failed" => nil
; [["VECTOR",:[T.expr for T in T1]],m,e]
```

```
[comp p557]
[$EmptyVector p??]
```

— defun compVector —

```
(defun |compVector| (form mode env)
  (let (tmp1 tmp2 t0 failed (newmode (second mode)))
    (declare (special |$EmptyVector|))
    (if (null form)
      (list |$EmptyVector| mode env)
      (progn
        (setq t0
          (do ((t3 form (cdr t3)) (x nil))
              ((or (atom t3) failed) (unless failed (nreverse0 tmp2)))
            (setq x (car t3))
            (if (setq tmp1 (|comp| x newmode env))
              (progn
                (setq newmode (second tmp1))
                (setq env (third tmp1))
                (push tmp1 tmp2))
              (setq failed t))))))
        (unless failed
          (list (cons 'vector
                     (loop for texpr in t0 collect (car texpr))) mode env))))))
```

5.6.134 defplist compWhere plist

— postvars —

```
(eval-when (eval load)
```

```
(setf (get '|where| 'special) '|compWhere|))
```

5.6.135 defun compWhere

```
[comp p557]
[macroExpand p136]
[deltaContour p??]
[addContour p??]
[$insideExpressionIfTrue p??]
[$insideWhereIfTrue p??]
[$EmptyMode p131]
```

— defun compWhere —

```
(defun |compWhere| (form mode eInit)
  (let (|$insideExpressionIfTrue| |$insideWhereIfTrue| newform exprList e
        eBefore tmp1 x eAfter del eFinal)
    (declare (special |$insideExpressionIfTrue| |$insideWhereIfTrue|
                      |$EmptyMode|))
    (setq newform (second form))
    (setq exprlist (cddr form))
    (setq |$insideExpressionIfTrue| nil)
    (setq |$insideWhereIfTrue| t)
    (setq e eInit)
    (when (dolist (item exprList t)
      (setq tmp1 (|comp| item |$EmptyMode| e))
      (unless tmp1 (return nil))
      (setq e (third tmp1))))
    (setq |$insideWhereIfTrue| nil)
    (setq tmp1 (|comp| (|macroExpand| newform (setq eBefore e)) mode e))
    (when tmp1
      (setq x (first tmp1))
      (setq mode (second tmp1))
      (setq eAfter (third tmp1))
      (setq del (|deltaContour| eAfter eBefore))
      (if del
        (setq eFinal (|addContour| del eInit))
        (setq eFinal eInit))
      (list x mode eFinal))))))
```

5.7 Functions for coercion

5.7.1 defun coerce

The function `coerce` is used by the old compiler for coercions. The function `coerceInteractive` is used by the interpreter. One should always call the correct function, since the representation of basic objects may not be the same. [`keyedSystemError p??`]

```
[rplac p??]
[msubst p??]
[coerceEasy p346]
[coerceSubset p347]
[coerceHard p348]
[isSomeDomainVariable p??]
[stackMessage p??]
[$InteractiveMode p??]
[$Rep p??]
[$fromCoerceable p??]
```

— **defun coerce** —

```
(defun |coerce| (tt mode)
  (labels (
    (fn (x m1 m2)
      (list '|Cannot coerce| '|%b| x '|%d| '|%l| '|      of mode| '|%b| m1
            '|%d| '|%l| '|      to mode| '|%b| m2 '|%d|)))
    (let (tp)
      (declare (special |$fromCoerceable$| |$Rep| |$InteractiveMode|))
      (if |$InteractiveMode|
        (|keyedSystemError| 'S2GE0016
          (list "coerce" "function coerce called from the interpreter."))
        (progn
          (|rplac| (cadr tt) (msubst '$ |$Rep| (cadr tt)))
          (cond
            ((setq tp (|coerceEasy| tt mode)) tp)
            ((setq tp (|coerceSubset| tt mode)) tp)
            ((setq tp (|coerceHard| tt mode)) tp)
            ((or (eq (car tt) '|$fromCoerceable$|) (|isSomeDomainVariable| mode)) nil)
            (t (|stackMessage| (fn (first tt) (second tt) mode))))))))))
```

5.7.2 defun coerceEasy

```
[modeEqualSubst p357]
[$EmptyMode p131]
[$Exit p??]
```

[\$NoValueMode p131]
 [\$Void p??]

— **defun coerceEasy** —

```
(defun |coerceEasy| (tt m)
  (declare (special |$EmptyMode| |$Exit| |$NoValueMode| |$Void|))
  (cond
    ((equal m |$EmptyMode|) tt)
    ((or (equal m |$NoValueMode|) (equal m |$Void|))
     (list (car tt) m (third tt)))
    ((equal (second tt) m) tt)
    ((equal (second tt) |$NoValueMode|) tt)
    ((equal (second tt) |$Exit|)
     (list
      (list 'progn (car tt) (list 'userError| "Did not really exit."))
      m (third tt)))
    ((or (equal (second tt) |$EmptyMode|)
         (|modeEqualSubst| (second tt) m (third tt)))
     (list (car tt) m (third tt)))))
```

—————

5.7.3 defun coerceSubset

[isSubset p??]
 [lassoc p??]
 [get p??]
 [opOf p??]
 [eval p??]
 [msubst p??]
 [isSubset p??]
 [maxSuperType p338]

— **defun coerceSubset** —

```
(defun |coerceSubset| (arg1 mp)
  (let (x m env pred)
    (setq x (first arg1))
    (setq m (second arg1))
    (setq env (third arg1))
    (cond
      ((or (|isSubset| m mp env) (and (eq m '|Rep|) (eq mp '$)))
       (list x mp env))
      ((and (consp m) (eq (qfirst m) '|SubDomain|)
            (consp (qrest m)) (equal (qsecond m) mp))
       (list x mp env))
```

```

((and (setq pred (lassoc (lop0f| mp) (|get| (lop0f| m) '|SubDomain| env)))
      (integerp x) (|eval| (msubst x '|#1| pred)))
      (list x mp env))
((and (setq pred (|isSubset| mp (|maxSuperType| m env) env))
      (integerp x) (|eval| (msubst x '* pred)))
      (list x mp env))
(t nil))))

```

5.7.4 defun coerceHard

```

[modeEqual p357]
[get p??]
[getmode p??]
[isCategoryForm p??]
[extendsCategoryForm p??]
[coerceExtraHard p349]
[$e p??]
[$e p??]
[$String p339]
[$bootstrapMode p??]

```

— defun coerceHard —

```

(defun |coerceHard| (tt m)
  (let (|$e| mp tmp1 mpp)
    (declare (special |$e| |$String| |$bootstrapMode|))
    (setq |$e| (third tt))
    (setq mp (second tt))
    (cond
      ((and (stringp mp) (|modeEqual| m |$String|))
        (list (car tt) m |$e|))
      ((or (|modeEqual| mp m)
           (and (or (progn
                     (setq tmp1 (|get| mp '|value| |$e|))
                     (and (consp tmp1)
                          (progn (setq mpp (qfirst tmp1)) t)))
                (progn
                  (setq tmp1 (|getmode| mp |$e|))
                  (and (consp tmp1)
                       (eq (qfirst tmp1) '|Mapping|)
                       (and (consp (qrest tmp1))
                            (eq (qcddr tmp1) nil)
                            (progn (setq mpp (qsecond tmp1)) t))))))
           (|modeEqual| mpp m))
        (and (or (progn

```

```

      (setq tmp1 (|get| m '|value| |$e|))
      (and (consp tmp1)
        (progn (setq mpp (qfirst tmp1)) t)))
    (progn
      (setq tmp1 (|getmode| m |$e|))
      (and (consp tmp1)
        (eq (qfirst tmp1) '|Mapping|)
        (and (consp (qrest tmp1))
          (eq (qcddr tmp1) nil)
          (progn (setq mpp (qsecond tmp1)) t))))))
    (|modeEqual| mpp mp)))
  (list (car tt) m (third tt)))
((and (stringp (car tt)) (equal (car tt) m))
  (list (car tt) m |$e|))
((|isCategoryForm| m |$e|)
  (cond
    ((eq |$bootStrapMode| t)
      (list (car tt) m |$e|))
    ((|extendsCategoryForm| (car tt) (cadr tt) m)
      (list (car tt) m |$e|))
    (t (|coerceExtraHard| tt m))))
(t (|coerceExtraHard| tt m))))

```

5.7.5 defun coerceExtraHard

```

[autoCoerceByModemap p354]
[isUnionMode p311]
[qcar p??]
[qcdr p??]
[hasType p350]
[member p??]
[autoCoerceByModemap p354]
[coerce p346]
[$Expression p??]

```

— defun coerceExtraHard —

```

(defun |coerceExtraHard| (tt m)
  (let (x mp e tmp1 z ta tp tpp)
    (declare (special |$Expression|))
    (setq x (first tt))
    (setq mp (second tt))
    (setq e (third tt))
    (cond
      ((setq tp (|autoCoerceByModemap| tt m)) tp)

```

```

((and (progn
      (setq tmp1 (|isUnionMode| mp e))
      (and (consp tmp1) (eq (qfirst tmp1) '|Union|)
        (progn
          (setq z (qrest tmp1)) t)))
      (setq ta (|hasType| x e))
      (|member| ta z)
      (setq tp (|autoCoerceByModemap| tt ta))
      (setq tpp (|coerce| tp m)))
      tpp)
  ((and (consp mp) (eq (qfirst mp) '|Record|) (equal m |$Expression|))
    (list (list '|coerceRe2E| x (list 'elt (copy mp) 0)) m e))
  (t nil)))

```

5.7.6 defun hasType

[get p??]

— defun hasType —

```

(defun |hasType| (x e)
  (labels (
    (fn (x)
      (cond
        ((null x) nil)
        ((and (consp x) (consp (qfirst x)) (eq (qcaar x) '|case|)
          (consp (qcddar x)) (consp (qcddar x))
          (eq (qcdddar x) nil))
          (qcaddar x))
        (t (fn (cdr x))))))
    (fn (|get| x '|condition| e))))

```

5.7.7 defun coerceable

[pmatch p??]
 [sublis p??]
 [coerce p346]
 [\$fromCoerceable p??]

— defun coerceable —

```
(defun |coerceable| (m mp env)
  (let (sl)
    (declare (special |$fromCoerceable$|))
    (cond
      ((equal m mp) m)
      ((setq sl (|pmatch| mp m)) (sublis sl mp))
      ((|coerce| (list '|$fromCoerceable$| m env) mp) mp)
      (t nil))))
```

5.7.8 defun coerceExit

[resolve p356]
 [replaceExitEsc p??]
 [coerce p346]
 [\$exitMode p??]

— defun coerceExit —

```
(defun |coerceExit| (arg1 mp)
  (let (x m e catchTag xp)
    (declare (special |$exitMode|))
    (setq x (first arg1))
    (setq m (second arg1))
    (setq e (third arg1))
    (setq mp (|resolve| m mp))
    (setq xp
      (|replaceExitEtc| x
        (setq catchTag (mkq (gensym))) ' |TAGGEDexit| |$exitMode|))
    (|coerce| (list (list 'catch catchTag xp) m e) mp)))
```

5.7.9 defplist compAtSign plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@| 'special) 'compAtSign))
```

5.7.10 defun compAtSign

```
[addDomain p232]
[comp p557]
[coerce p346]
```

— defun compAtSign —

```
(defun compAtSign (form mode env)
  (let ((newform (second form)) (mprime (third form)) tmp)
    (setq env (|addDomain| mprime env))
    (when (setq tmp (|comp| newform mprime env)) (|coerce| tmp mode))))
```

—————

5.7.11 defplist compCoerce plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||::| 'special) '|compCoerce|))
```

—————

5.7.12 defun compCoerce

```
[addDomain p232]
[getmode p??]
[compCoerce1 p353]
[coerce p346]
```

— defun compCoerce —

```
(defun |compCoerce| (form mode env)
  (let (newform newmode tmp1 tmp4 z td)
    (setq newform (second form))
    (setq newmode (third form))
    (setq env (|addDomain| newmode env))
    (setq tmp1 (|getmode| newmode env))
    (cond
      ((setq td (|compCoerce1| newform newmode env))
       (|coerce| td mode))
      ((and (consp tmp1) (eq (qfirst tmp1) '|Mapping|))
```

```

      (cons (qrest tmp1) (eq (qcddr tmp1) nil))
      (cons (qsecond tmp1)
            (eq (qcaadr tmp1) '|UnionCategory|))
      (setq z (qcdadr tmp1))
      (when
        (setq td
          (dolist (mode1 z tmp4)
            (setq tmp4 (or tmp4 (|compCoerce1| newform mode1 env))))))
      (|coerce| (list (car td) newmode (third td) mode))))))

```

5.7.13 defun compCoerce1

```

[comp p557]
[resolve p356]
[coerce p346]
[coerceByModemap p354]
[msubst p??]
[mkq p??]

```

— defun compCoerce1 —

```

(defun |compCoerce1| (form mode env)
  (let (m1 td tp gg pred code)
    (declare (special |$String| |$EmptyMode|))
    (when (setq td (or (|comp| form mode env) (|comp| form |$EmptyMode| env)))
      (setq m1 (if (stringp (second td)) |$String| (second td)))
      (setq mode (|resolve| m1 mode))
      (setq td (list (car td) m1 (third td)))
      (cond
        ((setq tp (|coerce| td mode)) tp)
        ((setq tp (|coerceByModemap| td mode)) tp)
        ((setq pred (|isSubset| mode (second td) env))
         (setq gg (gensym))
         (setq pred (msubst gg '* pred))
         (setq code
           (list 'prog1
             (list 'let gg (first td))
             (cons '|check-subtype| (cons pred (list (mkq mode) gg))))))
        (list code mode (third td))))))

```

5.7.14 defun coerceByModemap

```
[qcar p??]
[qcdr p??]
[modeEqual p357]
[isSubset p??]
[genDeltaEntry p??]
```

— defun coerceByModemap —

```
(defun |coerceByModemap| (arg1 mp)
  (let (x m env map cexpr u mm fn)
    (setq x (first arg1))
    (setq m (second arg1))
    (setq env (third arg1))
    (setq u
      (loop for modemap in (|getModemapList| ' |coerce| 1 env)
        do
          (setq map (first modemap))
          (setq cexpr (second modemap))
          when
            (and (consp map) (consp (qrest map))
                 (consp (qcddr map))
                 (eq (qcdddr map) nil)
                 (or (|modeEqual| (second map) mp) (|isSubset| (second map) mp env))
                 (or (|modeEqual| (third map) m) (|isSubset| m (third map) env)))
            collect modemap))
    (when u
      (setq mm (first u))
      (setq fn (|genDeltaEntry| (cons ' |coerce| mm)))
      (list (list ' |call| fn x) mp env))))
```

—————

5.7.15 defun autoCoerceByModemap

```
[qcar p??]
[qcdr p??]
[getModemapList p244]
[modeEqual p357]
[member p??]
[get p??]
[stackMessage p??]
[$fromCoerceable p??]
```

— defun autoCoerceByModemap —

```

(defun |autoCoerceByModemap| (arg1 target)
  (let (x source e map cexpr u fn y)
    (declare (special |$fromCoerceable$|))
    (setq x (first arg1))
    (setq source (second arg1))
    (setq e (third arg1))
    (setq u
      (loop for modemap in (|getModemapList| '|autoCoerce| 1 e)
        do
          (setq map (first modemap))
          (setq cexpr (second modemap))
          when
            (and (consp map) (consp (qrest map)) (consp (qcddr map))
              (eq (qcdddr map) nil)
              (|modeEqual| (second map) target)
              (|modeEqual| (third map) source))
            collect cexpr))
    (when u
      (setq fn
        (let (result)
          (loop for item in u
            do
              (when (first item) (setq result (or result (second item))))))
        result))
    (when fn
      (cond
        ((and (consp source) (eq (qfirst source) '|Union|)
          (|member| target (qrest source)))
          (cond
            ((and (setq y (|get| x '|condition| e))
              (let (result)
                (loop for u in y do
                  (setq result
                    (or result
                      (and (consp u) (eq (qfirst u) '|case|) (consp (qrest u))
                        (consp (qcddr u))
                        (eq (qcdddr u) nil)
                        (equal (qthird u) target))))))
              result))
            (list (list '|call| fn x) target e))
          ((eq x '|$fromCoerceable$|) nil)
          (t
            (|stackMessage|
              (list '|cannot coerce: | x '|%1| '| of mode: | source
                '|%1| '| to: | target '| without a case statement|))))))
    (t
      (list (list '|call| fn x) target e))))))

```

5.7.16 defun resolve

```
[nequal p??]
[modeEqual p357]
[mkUnion p356]
[$String p339]
[$EmptyMode p131]
[$NoValueMode p131]
```

— defun resolve —

```
(defun |resolve| (din dout)
  (declare (special |$String| |$EmptyMode| |$NoValueMode|))
  (cond
    ((or (equal din |$NoValueMode|) (equal dout |$NoValueMode|)) |$NoValueMode|)
    ((equal dout |$EmptyMode|) din)
    ((and (nequal din dout) (or (stringp din) (stringp dout)))
     (cond
       ((|modeEqual| dout |$String|) dout)
       ((|modeEqual| din |$String|) nil)
       (t (|mkUnion| din dout))))
    (t dout)))
```

—————

5.7.17 defun mkUnion

```
[qcar p??]
[qcdr p??]
[union p??]
[$Rep p??]
```

— defun mkUnion —

```
(defun |mkUnion| (a b)
  (declare (special |$Rep|))
  (cond
    ((and (eq b '$) (consp |$Rep|) (eq (qfirst |$Rep|) '|Union|))
     (qrest |$Rep|))
    ((and (consp a) (eq (qfirst a) '|Union|))
     (cond
       ((and (consp b) (eq (qfirst b) '|Union|))
        (cons '|Union| (|union| (qrest a) (qrest b))))
       (t (cons '|Union| (|union| (list b) (qrest a))))))
    ((and (consp b) (eq (qfirst b) '|Union|))
     (cons '|Union| (|union| (list a) (qrest b))))
```

```
(t (list '|Union| a b))))
```

5.7.18 defun This orders Unions

This orders Unions

— **defun modeEqual** —

```
(defun |modeEqual| (x y)
  (let (x1 y1)
    (cond
      ((or (atom x) (atom y)) (equal x y))
      ((nequal (|#| x) (|#| y)) nil)
      ((and (consp x) (eq (qfirst x) '|Union|) (consp y) (eq (qfirst y) '|Union|))
        (setq x1 (qrest x))
        (setq y1 (qrest y))
        (loop for a in x1 do
          (loop for b in y1 do
            (when (|modeEqual| a b)
              (setq x1 (|delete| a x1))
              (setq y1 (|delete| b y1))
              (return nil))))))
      (unless (or x1 y1) t))
    (t
     (let ((result t))
       (loop for u in x for v in y
         do (setq result (and result (|modeEqual| u v))))
       result))))))
```

5.7.19 defun modeEqualSubst

```
[modeEqual p357]
[modeEqualSubst p357]
[length p??]
```

— **defun modeEqualSubst** —

```
(defun |modeEqualSubst| (m1 m env)
  (let (mp op z1 z2)
    (cond
      ((|modeEqual| m1 m) t)
      ((atom m1)
```

```

(when (setq mp (car (|get| m1 '|value| env)))
  (|modeEqual| mp m)))
((and (consp m1) (consp m) (equal (qfirst m) (qfirst m1))
  (equal (|#| (qrest m1)) (|#| (qrest m)))))
  (setq op (qfirst m1))
  (setq z1 (qrest m1))
  (setq z2 (qrest m))
  (let ((result t))
    (loop for xm1 in z1 for xm2 in z2
      do (setq result (and result (|modeEqualSubst| xm1 xm2 env)))))
    result))
(t nil))))

```

5.7.20 compilerDoitWithScreenedLisplib

```

compilerDoitWithScreenedLisplib [embed p??]
[rwrite p??]
[compilerDoit p538]
[unembed p??]
[$saveableItems p??]
[$libFile p??]

```

— defun compilerDoitWithScreenedLisplib —

```

(defun |compilerDoitWithScreenedLisplib| (constructor fun)
  (declare (special |$saveableItems| |$libFile|))
  (embed 'rwrite
    '(lambda (key value stream)
      (cond
        ((and (eq stream |$libFile|)
          (not (member key |$saveableItems|)))
          value)
        ((not nil) (rwrite key value stream)))))
  (unwind-protect
    (|compilerDoit| constructor fun)
    (unembed 'rwrite)))

```

Chapter 6

Post Transformers

6.1 Direct called postparse routines

6.1.1 defun postTransform

[postTran p360]
[postTransform identp (vol5)]
[postTransformCheck p363]
[aplTran p395]

— defun postTransform —

```
(defun |postTransform| (y)
  (let (x tmp1 tmp2 tmp3 tmp4 tmp5 tt l u)
    (setq x y)
    (setq u (|postTran| x))
    (when
      (and (consp u) (eq (qfirst u) '|@Tuple|))
      (progn
        (setq tmp1 (qrest u))
        (and (consp tmp1)
              (progn (setq tmp2 (reverse tmp1)) t)
              (consp tmp2)
              (progn
                (setq tmp3 (qfirst tmp2))
                (and (consp tmp3)
                     (eq (qfirst tmp3) '|:|)
                     (progn
                      (setq tmp4 (qrest tmp3))
                      (and (consp tmp4)
                          (progn
                           (setq y (qfirst tmp4))
```

```

                (setq tmp5 (qrest tmp4))
                (and (consp tmp5)
                     (eq (qrest tmp5) nil)
                     (progn (setq tt (qfirst tmp5)) t))))))
            (progn (setq l (qrest tmp2)) t)
            (progn (setq l (nreverse l)) t)))
        (dolist (x l t) (unless (identp x) (return nil))))
    (setq u (list '|:| (cons 'listof (append l (list y)) tt)))
    (|postTransformCheck| u)
    (|aplTran| u)))

```

6.1.2 defun postTran

```

[postAtom p361]
[postTran p360]
[qcar p??]
[qcdr p??]
[unTuple p403]
[postTranList p362]
[postForm p364]
[postOp p361]
[postScriptsForm p362]

```

— defun postTran —

```

(defun |postTran| (x)
  (let (op f tmp1 a tmp2 tmp3 b y)
    (if (atom x)
        (|postAtom| x)
        (progn
         (setq op (car x))
         (cond
          ((and (atom op) (setq f (get1 op '|postTran|)))
           (funcall f x))
          ((and (consp op) (eq (qfirst op) '|elt|))
           (progn
            (setq tmp1 (qrest op))
            (and (consp tmp1)
                 (progn
                  (setq a (qfirst tmp1))
                  (setq tmp2 (qrest tmp1))
                  (and (consp tmp2)
                       (eq (qrest tmp2) nil)
                       (progn (setq b (qfirst tmp2)) t))))))
            (cons (|postTran| op) (cdr (|postTran| (cons b (cdr x)))))))

```

```

((and (consp op) (eq (qfirst op) '|Scripts|))
  (|postScriptsForm| op
    (dolist (y (rest x) tmp3)
      (setq tmp3 (append tmp3 (|unTuple| (|postTran| y))))))
  ((nequal op (setq y (|postOp| op)))
    (cons y (|postTranList| (cdr x)))
    (t (|postForm| x)))))

```

6.1.3 defun postOp

— defun postOp —

```

(defun |postOp| (x)
  (declare (special $boot))
  (cond
    ((eq x '|:=|) (if $boot 'spadlet 'let))
    ((eq x '|:-|) 'letd)
    ((eq x '|Attribute|) 'attribute)
    (t x)))

```

6.1.4 defun postAtom

[\$boot p??]

— defun postAtom —

```

(defun |postAtom| (x)
  (declare (special $boot))
  (cond
    ($boot x)
    ((eq1 x 0) '|Zero|)
    ((eq1 x 1) '|One|)
    ((eq x t) 't$)
    ((and (identp x) (getdatabase x 'niladic)) (list x))
    (t x)))

```

6.1.5 defun postTranList

[postTran p360]

— defun postTranList —

```
(defun |postTranList| (x)
  (loop for y in x collect (|postTran| y)))
```

—————

6.1.6 defun postScriptsForm

[getScriptName p399]

[length p??]

[postTranScripts p362]

— defun postScriptsForm —

```
(defun |postScriptsForm| (form arg1)
  (let ((op (second form)) (a (third form)))
    (cons (|getScriptName| op a (|#| arg1))
          (append (|postTranScripts| a) arg1))))
```

—————

6.1.7 defun postTranScripts

[postTranScripts p362]

[postTran p360]

— defun postTranScripts —

```
(defun |postTranScripts| (a)
  (labels (
    (fn (x)
      (if (and (consp x) (eq (qfirst x) '|@Tuple|))
          (qrest x)
          (list x))))
    (let (tmp1 tmp2 tmp3)
      (cond
        ((and (consp a) (eq (qfirst a) '|PrefixSC|))
         (progn
          (setq tmp1 (qrest a))
```

```

      (and (consp tmp1) (eq (qrest tmp1) nil))))
    (|postTranScripts| (qfirst tmp1)))
  ((and (consp a) (eq (qfirst a) '|;|))
   (dolist (y (qrest a) tmp2)
    (setq tmp2 (append tmp2 (|postTranScripts| y)))))
  ((and (consp a) (eq (qfirst a) '|,|))
   (dolist (y (qrest a) tmp3)
    (setq tmp3 (append tmp3 (fn (|postTran| y))))))
  (t (list (|postTran| a)))))

```

6.1.8 defun postTransformCheck

[postcheck p363]
 [\$defOp p??]

— defun postTransformCheck —

```

(defun |postTransformCheck| (x)
  (let (|$defOp|)
    (declare (special |$defOp|))
    (setq |$defOp| nil)
    (|postcheck| x)))

```

6.1.9 defun postcheck

[setDefOp p394]
 [postcheck p363]

— defun postcheck —

```

(defun |postcheck| (x)
  (cond
    ((atom x) nil)
    ((and (consp x) (eq (qfirst x) 'def) (consp (qrest x)))
     (|setDefOp| (qsecond x))
     (|postcheck| (qcddr x)))
    ((and (consp x) (eq (qfirst x) 'quote)) nil)
    (t (|postcheck| (car x)) (|postcheck| (cdr x)))))

```

6.1.10 defun postError

```
[nequal p??]
[bumperrorcount p517]
[$defOp p??]
[$InteractiveMode p??]
[$postStack p??]
```

— defun postError —

```
(defun |postError| (msg)
  (let (xmsg)
    (declare (special |$defOp| |$postStack| |$InteractiveMode|))
    (bumperrorcount ' |precompilation|)
    (setq xmsg
      (if (and (nequal |$defOp| ' |$defOp|) (null |$InteractiveMode|))
          (cons |$defOp| (cons ": " msg))
          msg))
    (push xmsg |$postStack|)
    nil))
```

6.1.11 defun postForm

```
[postTranList p362]
[internal p??]
[postTran p360]
[postError p364]
[bright p??]
[$boot p??]
```

— defun postForm —

```
(defun |postForm| (u)
  (let (op argl arglp numOfArgs opp x)
    (declare (special $boot))
    (seq
      (setq op (car u))
      (setq argl (cdr u))
      (setq x
        (cond
          ((atom op)
           (setq arglp (|postTranList| argl))
           (setq opp
             (seq
```

```

(exit op)
(when $boot (exit op))
(when (or (get1 op '|Led|) (get1 op '|Nud|) (eq op 'in)) (exit op))
(setq numOfArgs
  (cond
    ((and (consp arglp) (eq (qrest arglp) nil) (consp (qfirst arglp))
      (eq (qcaar arglp) '@Tuple|))
      (|#| (qcdar arglp)))
    (t 1)))
(internal '* (princ-to-string numOfArgs) (pname op)))
(cons opp arglp))
((and (consp op) (eq (qfirst op) '|Scripts|))
  (append (|postTran| op) (|postTranList| argl)))
(t
  (setq u (|postTranList| u))
  (cond
    ((and (consp u) (consp (qfirst u)) (eq (qcaar u) '@Tuple|))
      (|postError|
        (cons " "
          (append (|bright| u)
            (list "is illegal because tuples cannot be applied!" '|%1|
              " Did you misuse infix dot?")))))
    (t u)))
(cond
  ((and (consp x) (consp (qrest x)) (eq (qcddr x) nil)
    (consp (qsecond x)) (eq (qcaadr x) '@Tuple|))
    (cons (car x) (qcdadr x)))
  (t x))))

```

6.2 Indirect called postparse routines

In the **postTran** function there is the code:

```

((and (atom op) (setq f (get1 op '|postTran|)))
  (funcall f x))

```

The functions in this section are called through the symbol-plist of the symbol being parsed. The original list read:

add	postAdd
@	postAtSign
:BF:	postBigFloat
Block	postBlock
CATEGORY	postCategory

COLLECT	postCollect	
:	postColon	
::	postColonColon	
,	postComma	
construct	postConstruct	
==	postDef	
=>	postExit	
if	postIf	
in	postIn	;" the infix operator version of in"
IN	postIn	;" the iterator form of in"
Join	postJoin	
->	postMapping	
==>	postMDef	
pretend	postPretend	
QUOTE	postQUOTE	
Reduce	postReduce	
REPEAT	postRepeat	
Scripts	postScripts	
;	postSemiColon	
Signature	postSignature	
/	postSlash	
@Tuple	postTuple	
TupleCollect	postTupleCollect	
where	postWhere	
with	postWith	

6.2.1 defplist postAdd plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|add| '|postTran|) '|postAdd|))
```

—————

6.2.2 defun postAdd

[postTran p360]
[postCapsule p367]

— defun postAdd —

```
(defun |postAdd| (arg)
  (if (null (cddr arg))
      (|postCapsule| (second arg))
```

```
(list '|add| (|postTran| (second arg)) (|postCapsule| (third arg))))
```

6.2.3 defun postCapsule

```
[checkWarning p521]
[postBlockItem p368]
[postBlockItemList p367]
[postFlatten p376]
```

— defun postCapsule —

```
(defun |postCapsule| (x)
  (let (op)
    (cond
      ((null (and (consp x) (progn (setq op (qfirst x)) t)))
        (|checkWarning| (list "Apparent indentation error following add")))
      ((or (integerp op) (eq op '==))
        (list 'capsule (|postBlockItem| x)))
      ((eq op '|;|)
        (cons 'capsule (|postBlockItemList| (|postFlatten| x '|;|))))
      ((eq op '|if|)
        (list 'capsule (|postBlockItem| x)))
      (t (|checkWarning| (list "Apparent indentation error following add")))))
```

6.2.4 defun postBlockItemList

```
[postBlockItem p368]
```

— defun postBlockItemList —

```
(defun |postBlockItemList| (args)
  (let (result)
    (dolist (item args (nreverse result))
      (push (|postBlockItem| item) result))))
```

6.2.5 defun postBlockItem

[postTran p360]

— defun postBlockItem —

```
(defun |postBlockItem| (x)
  (let ((tmp1 t) tmp2 y tt z)
    (setq x (|postTran| x))
    (if
      (and (consp x) (eq (qfirst x) '|@Tuple|))
      (progn
        (and (consp (qrest x))
          (progn (setq tmp2 (reverse (qrest x))) t)
          (consp tmp2)
          (progn
            (and (consp (qfirst tmp2)) (eq (qcaar tmp2) '|:|))
            (progn
              (and (consp (qcddar tmp2))
                (progn
                  (setq y (qcadar tmp2))
                  (and (consp (qcddar tmp2))
                    (eq (qcdddar tmp2) nil)
                    (progn (setq tt (qcaddar tmp2)) t))))))
              (progn (setq z (qrest tmp2)) t)
              (progn (setq z (nreverse z)) T)))
          (do ((tmp6 nil (null tmp1)) (tmp7 z (cdr tmp7)) (x nil))
            ((or tmp6 (atom tmp7)) tmp1)
            (setq x (car tmp7))
            (setq tmp1 (and tmp1 (identp x)))))
          (cons '|:| (cons (cons 'listof (append z (list y))) (list tt)))
          x)))
      (progn
        (progn (setq z (qrest tmp2)) t)
        (progn (setq z (nreverse z)) T)))
      (do ((tmp6 nil (null tmp1)) (tmp7 z (cdr tmp7)) (x nil))
        ((or tmp6 (atom tmp7)) tmp1)
        (setq x (car tmp7))
        (setq tmp1 (and tmp1 (identp x)))))
      (cons '|:| (cons (cons 'listof (append z (list y))) (list tt)))
      x)))
```

—

6.2.6 defplist postAtSign plist

— postvars —

```
(eval-when (eval load)
  (setf (get '@ '|postTran|) '|postAtSign|))
```

—

6.2.7 defun postAtSign

[postTran p360]
[postType p369]

— defun postAtSign —

```
(defun |postAtSign| (arg)
  (cons '@ (cons (|postTran| (second arg)) (|postType| (third arg)))))
```

—————

6.2.8 defun postType

[postTran p360]
[unTuple p403]

— defun postType —

```
(defun |postType| (typ)
  (let (source target)
    (cond
      ((and (consp typ) (eq (qfirst typ) '->) (consp (qrest typ))
        (consp (qcddr typ)) (eq (qcddr typ) nil))
       (setq source (qsecond typ))
       (setq target (qthird typ))
       (cond
         ((eq source '|constant|)
          (list (list (|postTran| target)) '|constant|))
         (t
          (list (cons '|Mapping|
                     (cons (|postTran| target)
                           (|unTuple| (|postTran| source)))))))
      ((and (consp typ) (eq (qfirst typ) '->)
        (consp (qrest typ)) (eq (qcddr typ) nil))
       (list (list '|Mapping| (|postTran| (qsecond typ)))))
      (t (list (|postTran| typ)))))
```

—————

6.2.9 defplist postBigFloat plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|:BF:| '|postTran|) '|postBigFloat|))
```

6.2.10 defun postBigFloat

```
[postTran p360]
[$boot p??]
[$InteractiveMode p??]
```

— defun postBigFloat —

```
(defun |postBigFloat| (arg)
  (let (mant expon eltword)
    (declare (special $boot |$InteractiveMode|))
    (setq mant (second arg))
    (setq expon (cddr arg))
    (if $boot
        (times (float mant) (expt (float 10) expon))
        (progn
          (setq eltword (if |$InteractiveMode| '|$elt| '|elt|))
          (|postTran|
            (list (list eltword '(|Float|) '|float|)
                  (list '|,| (list '|,| mant expon) 10)))))))
```

6.2.11 defplist postBlock plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Block| '|postTran|) '|postBlock|))
```

6.2.12 defun postBlock

```
[postBlockItemList p367]
[postTran p360]
```

— defun postBlock —

```
(defun |postBlock| (arg)
  (let (tmp1 x y)
    (setq tmp1 (reverse (cdr arg)))
    (setq x (car tmp1))
    (setq y (nreverse (cdr tmp1)))
    (cons 'seq
      (append (|postBlockItemList| y) (list (list 'exit| (|postTran| x)))))))
```

6.2.13 defplist postCategory plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'category '|postTran|) '|postCategory|))
```

6.2.14 defun postCategory

```
[postTran p360]
[nreverse0 p??]
[$insidePostCategoryIfTrue p??]
```

— defun postCategory —

```
(defun |postCategory| (u)
  (declare (special |$insidePostCategoryIfTrue|))
  (labels (
    (fn (arg)
      (let (|$insidePostCategoryIfTrue|)
        (declare (special |$insidePostCategoryIfTrue|))
        (setq |$insidePostCategoryIfTrue| t)
        (|postTran| arg))) )
    (let ((z (cdr u)) op tmp1)
      (if (null z)
        u
        (progn
          (setq op (if |$insidePostCategoryIfTrue| 'progn 'category))
          (cons op (dolist (x z (nreverse0 tmp1)) (push (fn x) tmp1))))))))
```

6.2.15 defun postCollect,finish

```
[qcar p??]
[qcdr p??]
[postMakeCons p372]
[tuple2List p521]
[postTranList p362]
```

— defun postCollect,finish —

```
(defun |postCollect,finish| (op itl y)
  (let (tmp2 tmp5 newBody)
    (cond
      ((and (consp y) (eq (qfirst y) '|:|)
           (consp (qrest y)) (eq (qcddr y) nil))
       (list 'reduce '|append| 0 (cons op (append itl (list (qsecond y))))))
      ((and (consp y) (eq (qfirst y) '|Tuple|))
       (setq newBody
              (cond
                ((dolist (x (qrest y) tmp2)
                 (setq tmp2
                        (or tmp2 (and (consp x) (eq (qfirst x) '|:|)
                                         (consp (qrest x)) (eq (qcddr x) nil))))))
                (|postMakeCons| (qrest y)))
              ((dolist (x (qrest y) tmp5)
                 (setq tmp5 (or tmp5 (and (consp x) (eq (qfirst x) 'segment))))))
              (|tuple2List| (qrest y)))
              (t (cons '|construct| (|postTranList| (qrest y))))))
       (list 'reduce '|append| 0 (cons op (append itl (list newBody))))
       (t (cons op (append itl (list y))))))
```

—

6.2.16 defun postMakeCons

```
[postMakeCons p372]
[postTran p360]
```

— defun postMakeCons —

```
(defun |postMakeCons| (args)
  (let (a b)
    (cond
      ((null args) '|nil|)
      ((and (consp args) (consp (qfirst args)) (eq (qcaar args) '|:|)
           (consp (qcдар args)) (eq (qcddar args) nil))
       (setq a (qcadar args))
```

```

(setq b (qrest args))
(if b
  (list 'append| (|postTran| a) (|postMakeCons| b))
  (|postTran| a)))
(t (list 'cons| (|postTran| (car args)) (|postMakeCons| (cdr args))))))

```

6.2.17 defplist postCollect plist

— postvars —

```

(eval-when (eval load)
  (setf (get 'collect '|postTran|) '|postCollect|))

```

6.2.18 defun postCollect

```

[postCollect,finish p372]
[postCollect p373]
[postIteratorList p374]
[postTran p360]

```

— defun postCollect —

```

(defun |postCollect| (arg)
  (let (constructOp tmp3 m itl x)
    (setq constructOp (car arg))
    (setq tmp3 (reverse (cdr arg)))
    (setq x (car tmp3))
    (setq m (nreverse (cdr tmp3)))
    (cond
      ((and (consp x) (consp (qfirst x)) (eq (qcaar x) '|elt|)
            (consp (qcдар x)) (consp (qcddar x))
            (eq (qcdddar x) nil)
            (eq (qcaddar x) '|construct|))
        (|postCollect|
         (cons (list '|elt| (qcadar x) 'collect)
               (append m (list (cons '|construct| (qrest x)))))))
      (t
       (setq itl (|postIteratorList| m))
       (setq x
        (if (and (consp x) (eq (qfirst x) '|construct|)

```

```

      (cons (qrest x) (eq (qcddr x) nil))
    (qsecond x)
    x))
  (|postCollect,finish| constructOp itl (|postTran| x))))))

```

6.2.19 defun postIteratorList

[postTran p360]
 [postInSeq p382]
 [postIteratorList p374]

— defun postIteratorList —

```

(defun |postIteratorList| (args)
  (let (z p y u a b)
    (cond
      ((cons (consp args)
        (setq p (|postTran| (qfirst args)))
        (setq z (qrest args))
        (cond
          ((and (consp p) (eq (qfirst p) 'in) (consp (qrest p))
            (consp (qcddr p)) (eq (qcdddr p) nil))
            (setq y (qsecond p))
            (setq u (qthird p))
            (cond
              ((and (consp u) (eq (qfirst u) '|||) (consp (qrest u))
                (consp (qcddr u)) (eq (qcdddr u) nil))
                (setq a (qsecond u))
                (setq b (qthird u))
                (cons (list 'in y (|postInSeq| a))
                  (cons (list '||| b)
                    (|postIteratorList| z))))
              (t (cons (list 'in y (|postInSeq| u)) (|postIteratorList| z))))
            (t (cons p (|postIteratorList| z))))
          (t args))))

```

6.2.20 defplist postColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||| '|postTran|) '|postColon|))
```

6.2.21 defun postColon

```
[postTran p360]
[postType p369]
```

— defun postColon —

```
(defun |postColon| (u)
  (cond
    ((and (consp u) (eq (qfirst u) '|||)
      (consp (qrest u)) (eq (qcddr u) nil))
      (list '||| (|postTran| (qsecond u))))
    ((and (consp u) (eq (qfirst u) '|||) (consp (qrest u))
      (consp (qcddr u)) (eq (qcdddr u) nil))
      (cons '||| (cons (|postTran| (second u)) (|postType| (third u)))))))
```

6.2.22 defplist postColonColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '||::| '|postTran|) '|postColonColon|))
```

6.2.23 defun postColonColon

```
[postForm p364]
[$boot p??]
```

— defun postColonColon —

```
(defun |postColonColon| (u)
  (if (and $boot (consp u) (eq (qfirst u) '||::|) (consp (qrest u))
    (consp (qcddr u)) (eq (qcdddr u) nil))
```

```
(intern (princ-to-string (third u)) (second u))
(|postForm| u)))
```

6.2.24 defplist postComma plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|,| '|postTran|) '|postComma|))
```

6.2.25 defun postComma

```
[postTuple p392]
[comma2Tuple p376]
```

— defun postComma —

```
(defun |postComma| (u)
  (|postTuple| (|comma2Tuple| u)))
```

6.2.26 defun comma2Tuple

```
[postFlatten p376]
```

— defun comma2Tuple —

```
(defun |comma2Tuple| (u)
  (cons '|@Tuple| (|postFlatten| u '|,|)))
```

6.2.27 defun postFlatten

```
[postFlatten p376]
```

— defun postFlatten —

```
(defun |postFlatten| (x op)
  (let (a b)
    (cond
      ((and (consp x) (equal (qfirst x) op) (consp (qrest x))
            (consp (qcddr x)) (eq (qcddr x) nil))
       (setq a (qsecond x))
       (setq b (qthird x))
       (append (|postFlatten| a op) (|postFlatten| b op)))
      (t (list x)))))
```

6.2.28 defplist postConstruct plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|construct| '|postTran|) '|postConstruct|))
```

6.2.29 defun postConstruct

```
[comma2Tuple p376]
[postTranSegment p378]
[postMakeCons p372]
[tuple2List p521]
[postTranList p362]
[postTran p360]
```

— defun postConstruct —

```
(defun |postConstruct| (u)
  (let (b a tmp4 tmp7)
    (cond
      ((and (consp u) (eq (qfirst u) '|construct|)
            (consp (qrest u)) (eq (qcddr u) nil))
       (setq b (qsecond u))
       (setq a
        (if (and (consp b) (eq (qfirst b) '|,|))
            (|comma2Tuple| b)
            b))
       (cond
         ((and (consp a) (eq (qfirst a) 'segment) (consp (qrest a))
```

```

      (consp (qcddr a)) (eq (qcdddr a) nil))
    (list '|construct| (|postTranSegment| (second a) (third a))))
  ((and (consp a) (eq (qfirst a) '|@Tuple|))
   (cond
    ((dolist (x (qrest a) tmp4)
     (setq tmp4
      (or tmp4
       (and (consp x) (eq (qfirst x) '|:|)
        (consp (qrest x)) (eq (qcddr x) nil))))))
     (|postMakeCons| (qrest a)))
    ((dolist (x (qrest a) tmp7)
     (setq tmp7 (or tmp7 (and (consp x) (eq (qfirst x) 'segment))))))
     (|tuple2List| (qrest a)))
    (t (cons '|construct| (|postTranList| (qrest a))))))
  (t (list '|construct| (|postTran| a))))
  (t u)))

```

6.2.30 defun postTranSegment

[postTran p360]

— defun postTranSegment —

```

(defun |postTranSegment| (p q)
  (list 'segment (|postTran| p) (when q (|postTran| q))))

```

6.2.31 defplist postDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|==| '|postTran|) '|postDef|))

```

6.2.32 defun postDef

[postMDef p385]

[recordHeaderDocumentation p464]

```
[nequal p??]
[postTran p360]
[postDefArgs p380]
[nreverse0 p??]
[$boot p??]
[$maxSignatureLineNumber p??]
[$headerDocumentation p??]
[$docList p??]
[$InteractiveMode p??]
```

— defun postDef —

```
(defun |postDef| (arg)
  (let (defOp rhs lhs targetType tmp1 op arg1 newLhs
        argTypeList typeList form specialCaseForm tmp4 tmp6 tmp8)
    (declare (special $boot |$maxSignatureLineNumber| |$headerDocumentation|
                      |$docList| |$InteractiveMode|))
    (setq defOp (first arg))
    (setq lhs (second arg))
    (setq rhs (third arg))
    (if (and (consp lhs) (eq (qfirst lhs) '|macro|)
            (consp (qrest lhs)) (eq (qcddr lhs) nil))
        (|postMDef| (list '==> (second lhs) rhs))
        (progn
          (unless $boot (|recordHeaderDocumentation| nil))
          (when (nequal |$maxSignatureLineNumber| 0)
            (setq |$docList|
                  (cons (cons '|constructor| |$headerDocumentation|) |$docList|))
            (setq |$maxSignatureLineNumber| 0))
          (setq lhs (|postTran| lhs))
          (setq tmp1
            (if (and (consp lhs) (eq (qfirst lhs) '|:|)) (cdr lhs) (list lhs nil)))
          (setq form (first tmp1))
          (setq targetType (second tmp1))
          (when (and (null |$InteractiveMode|) (atom form)) (setq form (list form)))
          (setq newLhs
            (if (atom form)
                form
                (progn
                  (setq tmp1
                    (dolist (x form (nreverse0 tmp4))
                      (push
                        (if (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
                                (consp (qcddr x)) (eq (qcdddr x) nil))
                            (second x)
                            x)
                        tmp4)))
                  (setq op (car tmp1))
                  (setq arg1 (cdr tmp1))
```

```

      (cons op (|postDefArgs| arg1))))))
(setq argTypeList
(unless (atom form)
(dolist (x (cdr form) (nreverse0 tmp6))
(push
  (when (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
    (consp (qcddr x)) (eq (qcdddr x) nil))
    (third x))
  tmp6))))
(setq typeList (cons targetType argTypeList))
(when (atom form) (setq form (list form)))
(setq specialCaseForm (dolist (x form (nreverse tmp8)) (push nil tmp8)))
(list 'def newLhs typeList specialCaseForm (|postTran| rhs))))))

```

6.2.33 defun postDefArgs

[postError p364]
[postDefArgs p380]

— defun postDefArgs —

```

(defun |postDefArgs| (args)
  (let (a b)
    (cond
      ((null args) args)
      ((and (consp args) (consp (qfirst args)) (eq (qcaar args) '|:|)
        (consp (qcddr args)) (eq (qcdddr args) nil))
        (setq a (qcadar args))
        (setq b (qrest args))
        (cond
          (b (|postError|
            (list " Argument" a "of indefinite length must be last"))
            ((or (atom a) (and (consp a) (eq (qfirst a) 'quote)))
              a)
            (t
              (|postError|
                (list " Argument" a "of indefinite length must be a name")))))
          (t (cons (car args) (|postDefArgs| (cdr args)))))))
  (t (cons (car args) (|postDefArgs| (cdr args))))))

```

6.2.34 defplist postExit plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|=>' '|postTran|) '|postExit|))
```

—————

6.2.35 defun postExit

[postTran p360]

— defun postExit —

```
(defun |postExit| (arg)
  (list 'if (|postTran| (second arg))
    (list '|exit| (|postTran| (third arg))
      '|noBranch|))
```

—————

6.2.36 defplist postIf plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|if|' '|postTran|) '|postIf|))
```

—————

6.2.37 defun postIf

```
[nreverse0 p??]
[postTran p360]
[$boot p??]
```

— defun postIf —

```
(defun |postIf| (arg)
```

```

(let (tmp1)
  (if (null (and (consp arg) (eq (qfirst arg) '|if|)))
      arg
      (cons 'if
        (dolist (x (qrest arg) (nreverse0 tmp1))
          (push
            (if (and (null (setq x (|postTran| x))) (null $boot)) '|noBranch| x)
            tmp1))))))

```

6.2.38 defplist postin plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|in| '|postTran|) '|postin|))

```

6.2.39 defun postin

```

[systemErrorHere p??]
[postTran p360]
[postInSeq p382]

```

— defun postin —

```

(defun |postin| (arg)
  (if (null (and (consp arg) (eq (qfirst arg) '|in|) (consp (qrest arg))
                (consp (qcddr arg)) (eq (qcdddr arg) nil)))
      (|systemErrorHere| "postin")
      (list '|in| (|postTran| (second arg)) (|postInSeq| (third arg)))))

```

6.2.40 defun postInSeq

```

[postTranSegment p378]
[tuple2List p521]
[postTran p360]

```

— defun postInSeq —

```
(defun |postInSeq| (seq)
  (cond
    ((and (consp seq) (eq (qfirst seq) 'segment) (consp (qrest seq))
      (consp (qcddr seq)) (eq (qcddr seq) nil))
      (|postTranSegment| (second seq) (third seq)))
    ((and (consp seq) (eq (qfirst seq) '@Tuple|))
      (|tuple2List| (qrest seq)))
    (t (|postTran| seq))))
```

6.2.41 defplist postIn plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'in '|postTran|) '|postIn|))
```

6.2.42 defun postIn

```
[systemErrorHere p??]
[postTran p360]
[postInSeq p382]
```

— defun postIn —

```
(defun |postIn| (arg)
  (if (null (and (consp arg) (eq (qfirst arg) 'in) (consp (qrest arg))
    (consp (qcddr arg)) (eq (qcddr arg) nil)))
      (|systemErrorHere| "postIn")
      (list 'in (|postTran| (second arg)) (|postInSeq| (third arg)))))
```

6.2.43 defplist postJoin plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Join| '|postTran|) '|postJoin|))
```

6.2.44 defun postJoin

```
[postTran p360]
[postTranList p362]
```

— defun postJoin —

```
(defun |postJoin| (arg)
  (let (a l al)
    (setq a (|postTran| (cadr arg)))
    (setq l (|postTranList| (cddr arg)))
    (when (and (consp l) (eq (qrest l) nil) (consp (qfirst l))
              (member (qcaar l) '(attribute signature))))
      (setq l (list (list 'category (qfirst l)))))
    (setq al (if (and (consp a) (eq (qfirst a) '|@Tuple|)) (qrest a) (list a)))
    (cons '|Join| (append al l))))
```

6.2.45 defplist postMapping plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|->| '|postTran|) '|postMapping|))
```

6.2.46 defun postMapping

```
[postTran p360]
[unTuple p403]
```

— defun postMapping —

```
(defun |postMapping| (u)
  (if (null (and (consp u) (eq (qfirst u) '|->|)) (consp (qrest u))
```

```

      (cons (qcddr u) (eq (qcddr u) nil)))
u
(cons '|Mapping|
  (cons (|postTran| (third u))
    (|unTuple| (|postTran| (second u))))))

```

6.2.47 defplist postMDef plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|=>| '|postTran|) '|postMDef|))

```

6.2.48 defun postMDef

```

[postTran p360]
[throwkeyedmsg p??]
[nreverse0 p??]
[$InteractiveMode p??]
[$boot p??]

```

— defun postMDef —

```

(defun |postMDef| (arg)
  (let (rhs lhs tmp1 targetType form newLhs typeList tmp4 tmp5 tmp8)
    (declare (special |$InteractiveMode| $boot))
    (setq lhs (second arg))
    (setq rhs (third arg))
    (cond
      ((and |$InteractiveMode| (null $boot))
       (setq lhs (|postTran| lhs))
       (if (null (identp lhs))
         (|throwkeyedmsg| 's2ip0001 nil)
         (list 'mdef lhs nil nil (|postTran| rhs)))))
      (t
       (setq lhs (|postTran| lhs))
       (setq tmp1
         (if (and (cons (lhs) (eq (qfirst lhs) '|:|)) (cdr lhs) (list lhs nil)))
         (setq form (first tmp1))
         (setq targetType (second tmp1))

```

```

(setq form (if (atom form) (list form) form))
(setq newLhs
  (dolist (x form (nreverse0 tmp4))
    (push
      (if (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))) (second x) x)
      tmp4)))
(setq typeList
  (cons targetType
    (dolist (x (qrest form) (nreverse0 tmp5))
      (push
        (when (and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
          (consp (qcddr x)) (eq (qcdddr x) nil))
          (third x))
        tmp5))))
(list 'mdef newLhs typeList
  (dolist (x form (nreverse0 tmp8)) (push nil tmp8))
  (|postTran| rhs))))))

```

6.2.49 defplist postPretend plist

— postvars —

```

(eval-when (eval load)
  (setf (get '|pretend| '|postTran|) '|postPretend|))

```

6.2.50 defun postPretend

```

[postTran p360]
[postType p369]

```

— defun postPretend —

```

(defun |postPretend| (arg)
  (cons '|pretend| (cons (|postTran| (second arg)) (|postType| (third arg)))))

```

6.2.51 defplist postQUOTE plist

— postvars —

```
(eval-when (eval load)
  (setf (get 'quote '|postTran|) '|postQUOTE|))
```

6.2.52 defun postQUOTE

— defun postQUOTE —

```
(defun |postQUOTE| (arg) arg)
```

6.2.53 defplist postReduce plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Reduce| '|postTran|) '|postReduce|))
```

6.2.54 defun postReduce

```
[postTran p360]
[postReduce p387]
[$InteractiveMode p??]
```

— defun postReduce —

```
(defun |postReduce| (arg)
  (let (op expr g)
    (setq op (second arg))
    (setq expr (third arg))
    (if (or |$InteractiveMode| (and (consp expr) (eq (qfirst expr) 'collect)))
```

```
(list 'reduce op 0 (|postTran| expr))
(|postReduce|
 (list '|Reduce| op
  (list 'collect
   (list 'in (setq g (gensym)) expr)
   (list '|construct| g))))))
```

6.2.55 defplist postRepeat plist

— postvars —

```
(eval-when (eval load)
 (setf (get 'repeat '|postTran|) '|postRepeat|))
```

6.2.56 defun postRepeat

[postIteratorList p374]
[postTran p360]

— defun postRepeat —

```
(defun |postRepeat| (arg)
 (let (tmp1 x m)
  (setq tmp1 (reverse (cdr arg)))
  (setq x (car tmp1))
  (setq m (nreverse (cdr tmp1)))
  (cons 'repeat (append (|postIteratorList| m) (list (|postTran| x))))))
```

6.2.57 defplist postScripts plist

— postvars —

```
(eval-when (eval load)
 (setf (get '|Scripts| '|postTran|) '|postScripts|))
```

6.2.58 defun postScripts

[getScriptName p399]
[postTranScripts p362]

— defun postScripts —

```
(defun |postScripts| (arg)
  (cons (|getScriptName| (second arg) (third arg) 0)
        (|postTranScripts| (third arg))))
```

—————

6.2.59 defplist postSemiColon plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|;| '|postTran|) '|postSemiColon|))
```

—————

6.2.60 defun postSemiColon

[postBlock p370]
[postFlattenLeft p389]

— defun postSemiColon —

```
(defun |postSemiColon| (u)
  (|postBlock| (cons '|Block| (|postFlattenLeft| u '|;|))))
```

—————

6.2.61 defun postFlattenLeft

[postFlattenLeft p389]

— defun postFlattenLeft —

```
(defun |postFlattenLeft| (x op)
```

```
(let (a b)
  (cond
    ((and (consp x) (equal (qfirst x) op) (consp (qrest x))
          (consp (qcddr x)) (eq (qcddr x) nil))
     (setq a (qsecond x))
     (setq b (qthird x))
     (append (|postFlattenLeft| a op) (list b)))
    (t (list x)))))
```

6.2.62 defplist postSignature plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|Signature| '|postTran|) '|postSignature|))
```

6.2.63 defun postSignature

```
[postType p369]
[removeSuperfluousMapping p391]
[killColons p391]
```

— defun postSignature —

```
(defun |postSignature| (arg)
  (let (sig sig1 op)
    (setq op (second arg))
    (setq sig (third arg))
    (when (and (consp sig) (eq (qfirst sig) '->))
      (setq sig1 (|postType| sig))
      (setq op (|postAtom| (if (stringp op) (setq op (intern op)) op)))
      (cons 'signature
            (cons op (|removeSuperfluousMapping| (|killColons| sig1)))))))
```

6.2.64 defun removeSuperfluousMapping

— defun removeSuperfluousMapping —

```
(defun |removeSuperfluousMapping| (sig1)
  (if (and (consp sig1) (consp (qfirst sig1)) (eq (qcaar sig1) '|Mapping|))
      (cons (cdr (qfirst sig1)) (qrest sig1))
      sig1))
```

6.2.65 defun killColons

[killColons p391]

— defun killColons —

```
(defun |killColons| (x)
  (cond
    ((atom x) x)
    ((and (consp x) (eq (qfirst x) '|Record|)) x)
    ((and (consp x) (eq (qfirst x) '|Union|)) x)
    ((and (consp x) (eq (qfirst x) '|:|) (consp (qrest x))
          (consp (qcddr x)) (eq (qcddr x) nil))
     (|killColons| (third x)))
    (t (cons (|killColons| (car x)) (|killColons| (cdr x))))))
```

6.2.66 defplist postSlash plist

— postvars —

```
(eval-when (eval load)
  (setf (get '/' '|postTran|) '|postSlash|))
```

6.2.67 defun postSlash

[postTran p360]

— defun postSlash —

```
(defun |postSlash| (arg)
  (if (stringp (second arg))
      (|postTran| (list '|Reduce| (intern (second arg)) (third arg) ))
      (list '|/| (|postTran| (second arg)) (|postTran| (third arg)))))
```

6.2.68 defplist postTuple plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|@Tuple| '|postTran|) '|postTuple|))
```

6.2.69 defun postTuple

[postTranList p362]

— defun postTuple —

```
(defun |postTuple| (arg)
  (cond
    ((and (consp arg) (eq (qrest arg) nil) (eq (qfirst arg) '|@Tuple|))
     arg)
    ((and (consp arg) (eq (qfirst arg) '|@Tuple|) (consp (qrest arg)))
     (cons '|@Tuple| (|postTranList| (cdr arg)))))
```

6.2.70 defplist postTupleCollect plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|TupleCollect| '|postTran|) '|postTupleCollect|))
```

6.2.71 defun postTupleCollect

[postCollect p373]

— defun postTupleCollect —

```
(defun |postTupleCollect| (arg)
  (let (constructOp tmp1 x m)
    (setq constructOp (car arg))
    (setq tmp1 (reverse (cdr arg)))
    (setq x (car tmp1))
    (setq m (nreverse (cdr tmp1)))
    (|postCollect| (cons constructOp (append m (list (list '|construct| x)))))))
```

6.2.72 defplist postWhere plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|where| '|postTran|) '|postWhere|))
```

6.2.73 defun postWhere

[postTran p360]

[postTranList p362]

— defun postWhere —

```
(defun |postWhere| (arg)
  (let (b x)
    (setq b (third arg))
    (setq x (if (and (consp b) (eq (qfirst b) '|Block|)) (qrest b) (list b)))
    (cons '|where| (cons (|postTran| (second arg)) (|postTranList| x)))))
```

6.2.74 defplist postWith plist

— postvars —

```
(eval-when (eval load)
  (setf (get '|with| '|postTran|) '|postWith|))
```

—————

6.2.75 defun postWith

```
[postTran p360]
[$insidePostCategoryIfTrue p??]
```

— defun postWith —

```
(defun |postWith| (arg)
  (let (|$insidePostCategoryIfTrue| a)
    (declare (special |$insidePostCategoryIfTrue|))
    (setq |$insidePostCategoryIfTrue| t)
    (setq a (|postTran| (second arg)))
    (cond
      ((and (consp a) (member (qfirst a) '(signature attribute if)))
       (list 'category a))
      ((and (consp a) (eq (qfirst a) 'progn))
       (cons 'category (qrest a)))
      (t a))))
```

—————

6.3 Support routines

6.3.1 defun setDefOp

```
[$defOp p??]
[$topOp p??]
```

— defun setDefOp —

```
(defun |setDefOp| (f)
  (let (tmp1)
    (declare (special |$defOp| |$topOp|))
    (when (and (consp f) (eq (qfirst f) '|:|))
```

```

      (consp (setq tmp1 (qrest f))))
    (setq f (qfirst tmp1)))
  (unless (atom f) (setq f (car f)))
  (if |$topOp|
    (setq |$defOp| f)
    (setq |$topOp| f)))

```

6.3.2 defun aplTran

```

[aplTran1 p395]
[containsBang p398]
[$genno p??]
[$boot p??]

```

— defun aplTran —

```

(defun |aplTran| (x)
  (let ($genno u)
    (declare (special $genno $boot))
    (cond
      ($boot x)
      (t
       (setq $genno 0)
       (setq u (|aplTran1| x))
       (cond
         ((|containsBang| u) (|throwKeyedMsg| 's2ip0002 nil))
         (t u))))))

```

6.3.3 defun aplTran1

```

[aplTranList p397]
[aplTran1 p395]
[hasAplExtension p397]
[nreverse0 p??]
[ p??]
[$boot p??]

```

— defun aplTran1 —

```

(defun |aplTran1| (x)

```

```

(let (op argl1 argl f y opprime yprime tmp1 arglAssoc futureArgl g)
(declare (special $boot))
(if (atom x)
  x
  (progn
    (setq op (car x))
    (setq argl1 (cdr x))
    (setq argl (|aplTranList| argl1))
    (cond
      ((eq op '!))
      (cond
        ((and (consp argl)
              (progn
                (setq f (qfirst argl))
                (setq tmp1 (qrest argl))
                (and (consp tmp1)
                     (eq (qrest tmp1) nil)
                     (progn
                      (setq y (qfirst tmp1))
                      t))))))
        (cond
          ((and (consp y)
                (progn
                  (setq opprime (qfirst y))
                  (setq yprime (qrest y))
                  t)
                (eq opprime '!)))
          (|aplTran1| (cons op (cons op (cons f yprime))))))
      ($boot
       (cons 'collect
             (cons
              (list 'in (setq g (genvar)) (|aplTran1| y))
              (list (list f g) ))))
      (t
       (list 'map f (|aplTran1| y) ))))
    (t x)))
((progn
  (setq tmp1 (|hasAplExtension| argl))
  (and (consp tmp1)
       (progn
        (setq arglAssoc (qfirst tmp1))
        (setq futureArgl (qrest tmp1))
        t)))
 (cons '|reshape|
       (cons
        (cons 'collect
              (append
               (do ((tmp3 arglAssoc (cdr tmp3)) (tmp4 nil))
                   ((or (atom tmp3)
                        (progn (setq tmp4 (car tmp3)) nil)
                        nil)

```

```

      (progn
        (setq g (car tmp4))
        (setq a (cdr tmp4))
        nil))
      (nreverse0 tmp2))
    (push (list 'in g (list '|ravel| a))) tmp2))
  (list (|aplTran1| (cons op futureArg1))))
  (list (cdar arglAssoc)))
  (t (cons op argl))))))

```

6.3.4 defun aplTranList

[aplTran1 p395]
[aplTranList p397]

— defun aplTranList —

```

(defun |aplTranList| (x)
  (if (atom x)
      x
      (cons (|aplTran1| (car x)) (|aplTranList| (cdr x)))))

```

6.3.5 defun hasAplExtension

[nreverse0 p??]
[deepestExpression p398]
[genvar p??]
[aplTran1 p395]
[msubst p??]

— defun hasAplExtension —

```

(defun |hasAplExtension| (arg1)
  (let (tmp2 tmp3 y z g arglAssoc u)
    (when
      (dolist (x arg1 tmp2)
        (setq tmp2 (or tmp2 (and (consp x) (eq (qfirst x) '|))))))
      (setq u
        (dolist (x arg1 (nreverse0 tmp3))
          (push
            (if (and (consp x) (eq (qfirst x) '|)

```

```

      (consp (qrest x)) (eq (qcddr x) nil))
    (progn
      (setq y (qsecond x))
      (setq z (|deepestExpression| y))
      (setq arglAssoc
        (cons (cons (setq g (genvar)) (|aplTran1| z)) arglAssoc))
      (msubst g z y))
    x)
  tmp3)))
(cons arglAssoc u))))

```

6.3.6 defun deepestExpression

[deepestExpression p398]

— defun deepestExpression —

```

(defun |deepestExpression| (x)
  (if (and (consp x) (eq (qfirst x) '!))
      (consp (qrest x)) (eq (qcddr x) nil))
      (|deepestExpression| (qsecond x))
      x))

```

6.3.7 defun containsBang

[containsBang p398]

— defun containsBang —

```

(defun |containsBang| (u)
  (let (tmp2)
    (cond
      ((atom u) (eq u '!))
      ((and (consp u) (equal (qfirst u) 'quote)
        (consp (qrest u)) (eq (qcddr u) nil))
        nil)
      (t
        (dolist (x u tmp2)
          (setq tmp2 (or tmp2 (|containsBang| x)))))))

```

6.3.8 defun getScriptName

```
[getScriptName identp (vol5)]
[postError p364]
[internal p??]
[decodeScripts p399]
[getScriptName pname (vol5)]
```

— **defun getScriptName** —

```
(defun |getScriptName| (op a numberOfFunctionalArgs)
  (when (null (identp op))
    (|postError| (list " " op " cannot have scripts" )))
  (internal '* (princ-to-string numberOfFunctionalArgs)
    (|decodeScripts| a) (pname op)))
```

6.3.9 defun decodeScripts

```
[qcar p??]
[qcdr p??]
[strconc p??]
[decodeScripts p399]
```

— **defun decodeScripts** —

```
(defun |decodeScripts| (a)
  (labels (
    (fn (a)
      (let ((tmp1 0))
        (if (and (consp a) (eq (qfirst a) '|,|))
          (dolist (x (qrest a) tmp1) (setq tmp1 (+ tmp1 (fn x))))
          1))))
    (cond
      ((and (consp a) (eq (qfirst a) '|PrefixSC|)
        (consp (qrest a)) (eq (qcddr a) nil))
       (strconc (princ-to-string 0) (|decodeScripts| (qsecond a))))
      ((and (consp a) (eq (qfirst a) '|;|))
       (apply 'strconc (loop for x in (qrest a) collect (|decodeScripts| x))))
      ((and (consp a) (eq (qfirst a) '|,|))
       (princ-to-string (fn a)))
      (t
       (princ-to-string 1)))))
```

Chapter 7

DEF forms

7.0.10 defvar \$defstack

— initvars —

```
(defvar $defstack nil)
```

—————

7.0.11 defvar \$is-spill

— initvars —

```
(defvar $is-spill nil)
```

—————

7.0.12 defvar \$is-spill-list

— initvars —

```
(defvar $is-spill-list nil)
```

—————

7.0.13 defvar \$vl

— initvars —

```
(defvar $vl nil)
```

—————

7.0.14 defvar \$is-gensymlist

— initvars —

```
(defvar $is-gensymlist nil)
```

—————

7.0.15 defvar \$initial-gensym

— initvars —

```
(defvar initial-gensym (list (gensym)))
```

—————

7.0.16 defvar \$is-eqlist

— initvars —

```
(defvar $is-eqlist nil)
```

—————

7.0.17 defun hackforis

[hackforis1 p403]

— defun hackforis —

```
(defun hackforis (l) (mapcar #'hackforis1 L))
```

7.0.18 defun hackforis1

```
[kar p??]  
[eqcar p??]
```

— defun hackforis1 —

```
(defun hackforis1 (x)  
  (if (and (member (kar x) '(in on)) (eqcar (second x) 'is))  
      (cons (first x) (cons (cons 'spadlet (cdadr x)) (cddr x)))  
      x))
```

7.0.19 defun unTuple

— defun unTuple —

```
(defun |unTuple| (x)  
  (if (and (consp x) (eq (qfirst x) '|@Tuple|))  
      (qrest x)  
      (list x)))
```

7.0.20 defun errhuh

```
[systemError p??]
```

— defun errhuh —

```
(defun errhuh ()  
  (|systemError| "problem with BOOT to LISP translation"))
```

Chapter 8

PARSE forms

8.1 The original meta specification

This package provides routines to support the Metalanguage translator writing system. Metalanguage is described in META/LISP, R.D. Jenks, Tech Report, IBM T.J. Watson Research Center, 1969. Familiarity with this document is assumed.

Note that META/LISP and the meta parser/generator were removed from Axiom. This information is only for documentation purposes.

```
%      Scratchpad II Boot Language Grammar, Common Lisp Version
%      IBM Thomas J. Watson Research Center
%      Summer, 1986
%
%      NOTE: Substantially different from VM/LISP version, due to
%            different parser and attempt to render more within META proper.

.META(New NewExpr Process)
.PACKAGE 'BOOT'
.DECLARE(tmpTok TOK ParseMode DEFINITION-NAME LABLASOC)
.PREFIX 'PARSE-'

NewExpr:      '=')' .(processSynonyms) Command
              / .(SETQ DEFINITION-NAME (CURRENT-SYMBOL)) Statement ;

Command:      ')' SpecialKeyWord SpecialCommand +() ;

SpecialKeyWord: =(MATCH-CURRENT-TOKEN "IDENTIFIER)
                .(SETF (TOKEN-SYMBOL (CURRENT-TOKEN)) (unAbbreviateKeyword (CURRENT-SYMBOL))) ;

SpecialCommand: 'show' '<??' / Expression>! +(show #1) CommandTail
               / ?(MEMBER (CURRENT-SYMBOL) \noParseCommands)
               .(FUNCALL (CURRENT-SYMBOL))
```

```

/ ?(MEMBER (CURRENT-SYMBOL) \ $tokenCommands) TokenList
TokenCommandTail
/ PrimaryOrQM* CommandTail ;

TokenList:      (^?(isTokenDelimiter) +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN))* ;

TokenCommandTail:
    <TokenOption*>! ?(atEndOfLine) +( #2 -#1) .(systemCommand #1) ;

TokenOption:    '))' TokenList ;

CommandTail:    <Option*>! ?(atEndOfLine) +( #2 -#1) .(systemCommand #1) ;

PrimaryOrQM:    '?' +\? / Primary ;

Option:         '))' PrimaryOrQM* ;

Statement:      Expr{0} <(',' Expr{0})* +(Series #2 -#1)>;

InfixWith:      With +(Join #2 #1) ;

With:           'with' Category +(with #1) ;

Category:       'if' Expression 'then' Category <'else' Category>! +(if #3 #2 #1)
/ '(' Category <('; ' Category)*>! '))' +(CATEGORY #2 -#1)
/ .(SETQ $1 (LINE-NUMBER CURRENT-LINE)) Application
( ':' Expression +(Signature #2 #1)
    .(recordSignatureDocumentation ##1 $1)
/ +(Attribute #1)
    .(recordAttributeDocumentation ##1 $1));

Expression:     Expr{(PARSE-rightBindingPowerOf (MAKE-SYMBOL-OF PRIOR-TOKEN) ParseMode)}
    +#1 ;

Import:         'import' Expr{1000} <(',' Expr{1000})*>! +(import #2 -#1) ;

Infix:          =TRUE +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail>
Expression +( #2 #2 #1) ;

Prefix:         =TRUE +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail>
Expression +( #2 #1) ;

Suffix:         +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) <TokTail> +( #1 #1) ;

TokTail:        ?(AND (NULL \ $BOOT) (EQ (CURRENT-SYMBOL) "\$)
    (OR (ALPHA-CHAR-P (CURRENT-CHAR))
        (CHAR-EQ (CURRENT-CHAR) '$')
        (CHAR-EQ (CURRENT-CHAR) '%')
        (CHAR-EQ (CURRENT-CHAR) '(')))
    .(SETQ $1 (COPY-TOKEN PRIOR-TOKEN)) Qualification

```

```

.(SETQ PRIOR-TOKEN $1) ;

Qualification: '$' Primary1 +=(dollarTran #1 #1) ;

SemiColon: ';' (Expr{82} / + \throwAway) +(\; #2 #1) ;

Return: 'return' Expression +(return #1) ;

Exit: 'exit' (Expression / +\NoValue) +(exit #1) ;

Leave: 'leave' ( Expression / +\NoValue )
      ('from' Label +(leaveFrom #1 #1) / +(leave #1)) ;

Seg: GlyphTok{"\.\.} <Expression>! +(SEGMENT #2 #1) ;

Conditional: 'if' Expression 'then' Expression <'else' ElseClause>!
              +(if #3 #2 #1) ;

ElseClause: ?(EQ (CURRENT-SYMBOL) "if) Conditional / Expression ;

Loop: Iterator* 'repeat' Expr{110} +(REPEAT -#2 #1)
      / 'repeat' Expr{110} +(REPEAT #1) ;

Iterator: 'for' Primary 'in' Expression
          ( 'by' Expr{200} +(INBY #3 #2 #1) / +(IN #2 #1) )
          < '\|' Expr{111} +(\| #1) >
          / 'while' Expr{190} +(WHILE #1)
          / 'until' Expr{190} +(UNTIL #1) ;

Expr{RBP}: NudPart{RBP} <LedPart{RBP}>* +#1;

LabelExpr: Label Expr{120} +(LABEL #2 #1) ;

Label: '@<<' Name '>>' ;

LedPart{RBP}: Operation{"Led RBP} +#1;

NudPart{RBP}: (Operation{"Nud RBP} / Reduction / Form) +#1 ;

Operation{ParseMode RBP}:
  ^?(MATCH-CURRENT-TOKEN "IDENTIFIER)
  ?(GETL (SETQ tmptok (CURRENT-SYMBOL)) ParseMode)
  ?(LT RBP (PARSE-leftBindingPowerOf tmptok ParseMode))
  .(SETQ RBP (PARSE-rightBindingPowerOf tmptok ParseMode))
  getSemanticForm{tmptok ParseMode (ELEMN (GETL tmptok ParseMode) 5 NIL)} ;

% Binding powers stored under the Led and Red properties of an operator
% are set up by the file BOTTOMUP.LISP. The format for a Led property
% is <Operator Left-Power Right-Power>, and the same for a Nud, except that
% it may also have a fourth component <Special-Handler>. ELEMN attempts to

```

```

% get the Nth indicator, counting from 1.

leftBindingPowerOf{X IND}: =(LET ((Y (GETL X IND))) (IF Y (ELEMN Y 3 0) 0)) ;

rightBindingPowerOf{X IND}: =(LET ((Y (GETL X IND))) (IF Y (ELEMN Y 4 105) 105)) ;

getSemanticForm{X IND Y}:
    ?(AND Y (EVAL Y)) / ?(EQ IND "Nud) Prefix / ?(EQ IND "Led) Infix ;

Reduction:      ReductionOp Expr{1000} +(Reduce #2 #1) ;

ReductionOp:    ?(AND (GETL (CURRENT-SYMBOL) "Led)
                    (MATCH-NEXT-TOKEN "SPECIAL-CHAR (CODE-CHAR 47))) % Forgive me!
    +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) .(ADVANCE-TOKEN) ;

Form:           'iterate' < 'from' Label +(#1) >! +(iterate -#1)
                / 'yield' Application +(yield #1)
                / Application ;

Application:     Primary <Selector>* <Application +(#2 #1)>;

Selector:       ?NONBLANK ?(EQ (CURRENT-SYMBOL) "\.") ?(CHAR-NE (CURRENT-CHAR) "\ ")
                ' .' PrimaryNoFloat (=$BOOT +(ELT #2 #1)/ +( #2 #1))
                / (Float /' .' Primary) (=$BOOT +(ELT #2 #1)/ +( #2 #1));

PrimaryNoFloat: Primary1 <TokTail> ;

Primary:         Float /PrimaryNoFloat ;

Primary1:        VarForm <=(AND NONBLANK (EQ (CURRENT-SYMBOL) "\()") Primary1 +(#2 #1)>
                /Quad
                /String
                /IntegerTok
                /FormalParameter
                /='\' ( ?$BOOT Data / '\'' Expr{999} +(QUOTE #1))
                /Sequence
                /Enclosure ;

Float:           FloatBase (?NONBLANK FloatExponent / +0) +=(MAKE-FLOAT #4 #2 #2 #1) ;

FloatBase:       ?(FIXP (CURRENT-SYMBOL)) ?(CHAR-EQ (CURRENT-CHAR) ' .')
                ?(CHAR-NE (NEXT-CHAR) ' .')
                IntegerTok FloatBasePart
                /?(FIXP (CURRENT-SYMBOL)) ?(CHAR-EQ (CHAR-UPCASE (CURRENT-CHAR)) "E")
                IntegerTok +0 +0
                /?(DIGITP (CURRENT-CHAR)) ?(EQ (CURRENT-SYMBOL) "\.")
                +0 FloatBasePart ;

FloatBasePart:   ' .'

```

```

(? (DIGITP (CURRENT-CHAR)) += (TOKEN-NONBLANK (CURRENT-TOKEN)) IntegerTok
/ +0 +0);

FloatExponent: =(AND (MEMBER (CURRENT-SYMBOL) "(E e))
(FIND (CURRENT-CHAR) '+-'))
.(ADVANCE-TOKEN)
(IntegerTok/'+' IntegerTok/'-' IntegerTok +=(MINUS #1)/+0)
/?(IDENTP (CURRENT-SYMBOL)) =(SETQ $1 (FLOATEXPID (CURRENT-SYMBOL)))
.(ADVANCE-TOKEN) += $1 ;

Enclosure:      '(' ( Expr{6} ')' / ')' + (Tuple) )
/ '{' ( Expr{6} '}' + (brace (construct #1)) / '}' + (brace)) ;

IntegerTok:      NUMBER ;

FloatTok:        NUMBER += (IF \ $BOOT #1 (BFP- #1)) ;

FormalParameter: FormalParameterTok ;

FormalParameterTok: ARGUMENT-DESIGNATOR ;

Quad:           '$' + \$ / ? \ $BOOT GlyphTok{"\."} + \. ;

String:          SPADSTRING ;

VarForm:        Name <Scripts + (Scripts #2 #1) > + #1 ;

Scripts:         ?NONBLANK '[' ScriptItem ']' ;

ScriptItem:      Expr{90} <(';' ScriptItem)* + (\; #2 -#1)>
/ ';' ScriptItem + (PrefixSC #1) ;

Name:           IDENTIFIER + #1 ;

Data:           . (SETQ LABLASOC NIL) Sexpr + (QUOTE =(TRANSLABEL #1 LABLASOC)) ;

Sexpr:          . (ADVANCE-TOKEN) Sexpr1 ;

Sexpr1:         AnyId
< NBGlyphTok{"\="} Sexpr1
.(SETQ LABLASOC (CONS (CONS #2 ##1) LABLASOC))>
/ '\'' Sexpr1 + (QUOTE #1)
/ IntegerTok
/ '-' IntegerTok += (MINUS #1)
/ String
/ '<' <Sexpr1*>!'>' += (LIST2VEC #1)
/ '(' <Sexpr1* <GlyphTok{"\."} Sexpr1 += (NCONC #2 #1)>>!'>' ;

NBGlyphTok{tok}: ? (AND (MATCH-CURRENT-TOKEN "GLIPH tok) NONBLANK)

```

```

.(ADVANCE-TOKEN) ;

GliphTok{tok}:    ?(MATCH-CURRENT-TOKEN "GLIPH tok) .(ADVANCE-TOKEN) ;

AnyId:           IDENTIFIER
                  / (='$$' +=(CURRENT-SYMBOL) .(ADVANCE-TOKEN) / KEYWORD) ;

Sequence:        OpenBracket Sequence1 ']'
                  / OpenBrace Sequence1 '}' +(brace #1) ;

Sequence1:       (Expression +( #2 #1) / +( #1)) <IteratorTail +(COLLECT -#1 #1)> ;

OpenBracket:     =(EQ (getToken (SETQ $1 (CURRENT-SYMBOL))) "\[ )
                  (= (EQCAR $1 "elt) +(elt =(CADR $1) construct)
                  / +construct) .(ADVANCE-TOKEN) ;

OpenBrace:       =(EQ (getToken (SETQ $1 (CURRENT-SYMBOL))) "\{ )
                  (= (EQCAR $1 "elt) +(elt =(CADR $1) brace)
                  / +construct) .(ADVANCE-TOKEN) ;

IteratorTail:    ('repeat' <Iterator*>! / Iterator*) ;

.FIN ;

```

8.2 The PARSE code

8.2.1 defvar \$tmptok

```

— initvars —

(defvar |tmptok| nil)

```

8.2.2 defvar \$tok

```

— initvars —

(defvar tok nil)

```

8.2.3 defvar \$ParseMode— **initvars** —

```
(defvar |ParseMode| nil)
```

—————

8.2.4 defvar \$definition-name— **initvars** —

```
(defvar definition-name nil)
```

—————

8.2.5 defvar \$lablasoc— **initvars** —

```
(defvar lablasoc nil)
```

—————

8.2.6 defun PARSE-NewExpr

```
[match-string p448]
[action p461]
[PARSE-NewExpr processSynonyms (vol5)]
[must p460]
[current-symbol p454]
[PARSE-Statement p416]
[definition-name p411]
```

— **defun PARSE-NewExpr** —

```
(defun |PARSE-NewExpr| ()
  (or (and (match-string "") (action (|processSynonyms|))
          (must (|PARSE-Command|))))
```

```
(and (action (setq definition-name (current-symbol)))
      (|PARSE-Statement|))))
```

8.2.7 defun PARSE-Command

```
[match-advance-string p449]
[must p460]
[PARSE-SpecialKeyWord p412]
[PARSE-SpecialCommand p413]
[push-reduction p462]
```

— defun PARSE-Command —

```
(defun |PARSE-Command| ()
  (and (match-advance-string "") (must (|PARSE-SpecialKeyWord|))
        (must (|PARSE-SpecialCommand|))
        (push-reduction '|PARSE-Command| nil))))
```

8.2.8 defun PARSE-SpecialKeyWord

```
[match-current-token p453]
[action p461]
[token-symbol p??]
[current-token p455]
[PARSE-SpecialKeyWord unAbbreviateKeyword (vol5)]
[current-symbol p454]
```

— defun PARSE-SpecialKeyWord —

```
(defun |PARSE-SpecialKeyWord| ()
  (and (match-current-token 'identifier)
        (action (setf (token-symbol (current-token))
                      (|unAbbreviateKeyword| (current-symbol))))))
```

8.2.9 defun PARSE-SpecialCommand

[match-advance-string p449]
 [bang p??]
 [optional p461]
 [PARSE-Expression p419]
 [push-reduction p462]
 [PARSE-SpecialCommand p413]
 [pop-stack-1 p522]
 [PARSE-CommandTail p415]
 [must p460]
 [current-symbol p454]
 [action p461]
 [PARSE-TokenList p414]
 [PARSE-TokenCommandTail p413]
 [star p461]
 [PARSE-PrimaryOrQM p415]
 [PARSE-CommandTail p415]
 [\$noParseCommands p??]
 [\$tokenCommands p??]

— defun PARSE-SpecialCommand —

```
(defun |PARSE-SpecialCommand| ()
  (declare (special |$noParseCommands| |$tokenCommands|))
  (or (and (match-advance-string "show")
    (bang fil_test
      (optional
        (or (match-advance-string "?")
          (|PARSE-Expression|))))
    (push-reduction '|PARSE-SpecialCommand|
      (list '|show| (pop-stack-1)))
    (must (|PARSE-CommandTail|)))
    (and (member (current-symbol) |$noParseCommands|)
      (action (funcall (current-symbol))))
    (and (member (current-symbol) |$tokenCommands|)
      (|PARSE-TokenList|) (must (|PARSE-TokenCommandTail|)))
    (and (star repeater (|PARSE-PrimaryOrQM|))
      (must (|PARSE-CommandTail|))))))
```

8.2.10 defun PARSE-TokenCommandTail

[bang p??]
 [optional p461]

```

[star p461]
[PARSE-TokenOption p414]
[atEndOfLine p??]
[push-reduction p462]
[PARSE-TokenCommandTail p413]
[pop-stack-2 p523]
[pop-stack-1 p522]
[action p461]
[PARSE-TokenCommandTail systemCommand (vol5)]

```

— **defun PARSE-TokenCommandTail** —

```

(defun |PARSE-TokenCommandTail| ()
  (and (bang fil_test (optional (star repeater (|PARSE-TokenOption|))))
    (|atEndOfLine|)
    (push-reduction ' |PARSE-TokenCommandTail|
      (cons (pop-stack-2) (append (pop-stack-1) nil)))
    (action (|systemCommand| (pop-stack-1)))))

```

—————

8.2.11 defun PARSE-TokenOption

```

[match-advance-string p449]
[must p460]
[PARSE-TokenList p414]

```

— **defun PARSE-TokenOption** —

```

(defun |PARSE-TokenOption| ()
  (and (match-advance-string "") (must (|PARSE-TokenList|))))

```

—————

8.2.12 defun PARSE-TokenList

```

[star p461]
[isTokenDelimiter p451]
[push-reduction p462]
[current-symbol p454]
[action p461]
[advance-token p456]

```

— **defun PARSE-TokenList** —

```
(defun |PARSE-TokenList| ()
  (star repeater
    (and (not (|isTokenDelimiter|))
      (push-reduction '|PARSE-TokenList| (current-symbol))
      (action (advance-token))))))
```

8.2.13 defun PARSE-CommandTail

```
[bang p??]
[optional p461]
[star p461]
[push-reduction p462]
[PARSE-Option p416]
[PARSE-CommandTail p415]
[pop-stack-2 p523]
[pop-stack-1 p522]
[action p461]
[PARSE-CommandTail systemCommand (vol5)]
```

— defun PARSE-CommandTail —

```
(defun |PARSE-CommandTail| ()
  (and (bang fil_test (optional (star repeater (|PARSE-Option|))))
    (|atEndOfLine|)
    (push-reduction '|PARSE-CommandTail|
      (cons (pop-stack-2) (append (pop-stack-1) nil)))
    (action (|systemCommand| (pop-stack-1)))))
```

8.2.14 defun PARSE-PrimaryOrQM

```
[match-advance-string p449]
[push-reduction p462]
[PARSE-PrimaryOrQM p415]
[PARSE-Primary p428]
```

— defun PARSE-PrimaryOrQM —

```
(defun |PARSE-PrimaryOrQM| ()
  (or (and (match-advance-string "?")
    (push-reduction '|PARSE-PrimaryOrQM| '??))
```

```
(|PARSE-Primary|))
```

8.2.15 defun PARSE-Option

```
[match-advance-string p449]
[must p460]
[star p461]
[PARSE-PrimaryOrQM p415]
```

— defun PARSE-Option —

```
(defun |PARSE-Option| ()
  (and (match-advance-string "")
        (must (star repeater (|PARSE-PrimaryOrQM|))))))
```

8.2.16 defun PARSE-Statement

```
[PARSE-Expr p420]
[optional p461]
[star p461]
[match-advance-string p449]
[must p460]
[push-reduction p462]
[pop-stack-2 p523]
[pop-stack-1 p522]
```

— defun PARSE-Statement —

```
(defun |PARSE-Statement| ()
  (and (|PARSE-Expr| 0)
        (optional
          (and (star repeater
                    (and (match-advance-string ",")
                        (must (|PARSE-Expr| 0))))
                (push-reduction '|PARSE-Statement|
                                (cons '|Series|
                                      (cons (pop-stack-2)
                                            (append (pop-stack-1) nil))))))))))
```

8.2.17 defun PARSE-InfixWith

[PARSE-With p417]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— defun PARSE-InfixWith —

```
(defun |PARSE-InfixWith| ()
  (and (|PARSE-With|)
    (push-reduction '|PARSE-InfixWith|
      (list '|Join| (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.18 defun PARSE-With

[match-advance-string p449]
 [must p460]
 [push-reduction p462]
 [pop-stack-1 p522]

— defun PARSE-With —

```
(defun |PARSE-With| ()
  (and (match-advance-string "with") (must (|PARSE-Category|))
    (push-reduction '|PARSE-With|
      (cons '|with| (cons (pop-stack-1) nil))))))
```

—————

8.2.19 defun PARSE-Category

[match-advance-string p449]
 [must p460]
 [bang p??]
 [optional p461]
 [push-reduction p462]
 [PARSE-Expression p419]
 [PARSE-Category p417]
 [pop-stack-3 p523]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

8.2.20 defun PARSE-Expression

[PARSE-Expr p420]
 [PARSE-rightBindingPowerOf p422]
 [make-symbol-of p454]
 [push-reduction p462]
 [pop-stack-1 p522]
 [ParseMode p411]
 [prior-token p90]

— **defun PARSE-Expression** —

```
(defun |PARSE-Expression| ()
  (declare (special prior-token))
  (and (|PARSE-Expr|
        (|PARSE-rightBindingPowerOf| (make-symbol-of prior-token)
                                       |ParseMode|))
        (push-reduction '|PARSE-Expression| (pop-stack-1))))
```

—————

8.2.21 defun PARSE-Import

[match-advance-string p449]
 [must p460]
 [PARSE-Expr p420]
 [bang p??]
 [optional p461]
 [star p461]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-Import** —

```
(defun |PARSE-Import| ()
  (and (match-advance-string "import") (must (|PARSE-Expr| 1000))
        (bang fil_test
          (optional
            (star repeater
              (and (match-advance-string ",")
                    (must (|PARSE-Expr| 1000)))))))
        (push-reduction '|PARSE-Import|
          (cons '|import|
            (cons (pop-stack-2) (append (pop-stack-1) nil))))))
```

8.2.22 defun PARSE-Expr

[PARSE-NudPart p420]
 [PARSE-LedPart p420]
 [optional p461]
 [star p461]
 [push-reduction p462]
 [pop-stack-1 p522]

— defun PARSE-Expr —

```
(defun |PARSE-Expr| (rbp)
  (declare (special rbp))
  (and (|PARSE-NudPart| rbp)
    (optional (star opt_expr (|PARSE-LedPart| rbp)))
    (push-reduction '|PARSE-Expr| (pop-stack-1))))
```

8.2.23 defun PARSE-LedPart

[PARSE-Operation p421]
 [push-reduction p462]
 [pop-stack-1 p522]

— defun PARSE-LedPart —

```
(defun |PARSE-LedPart| (rbp)
  (declare (special rbp))
  (and (|PARSE-Operation| '|Led| rbp)
    (push-reduction '|PARSE-LedPart| (pop-stack-1))))
```

8.2.24 defun PARSE-NudPart

[PARSE-Operation p421]
 [PARSE-Reduction p425]
 [PARSE-Form p425]
 [push-reduction p462]
 [pop-stack-1 p522]

[rbp p??]

— defun PARSE-NudPart —

```
(defun |PARSE-NudPart| (rbp)
  (declare (special rbp))
  (and (or (|PARSE-Operation| ' |Nud| rbp) (|PARSE-Reduction|)
    (|PARSE-Form|)))
  (push-reduction ' |PARSE-NudPart| (pop-stack-1))))
```

—————

8.2.25 defun PARSE-Operation

[match-current-token p453]
 [current-symbol p454]
 [PARSE-leftBindingPowerOf p421]
 [lt p??]
 [getl p??]
 [action p461]
 [PARSE-rightBindingPowerOf p422]
 [PARSE-getSemanticForm p422]
 [elemn p??]
 [ParseMode p411]
 [rbp p??]
 [tmptok p410]

— defun PARSE-Operation —

```
(defun |PARSE-Operation| (|ParseMode| rbp)
  (declare (special |ParseMode| rbp |tmptok|))
  (and (not (match-current-token 'identifier))
    (getl (setq |tmptok| (current-symbol)) |ParseMode|)
    (lt rbp (|PARSE-leftBindingPowerOf| |tmptok| |ParseMode|))
    (action (setq rbp (|PARSE-rightBindingPowerOf| |tmptok| |ParseMode|))
      (|PARSE-getSemanticForm| |tmptok| |ParseMode|
        (elemn (getl |tmptok| |ParseMode|) 5 nil))))
```

—————

8.2.26 defun PARSE-leftBindingPowerOf

[getl p??]
 [elemn p??]

— defun PARSE-leftBindingPowerOf —

```
(defun |PARSE-leftBindingPowerOf| (x ind)
  (declare (special x ind))
  (let ((y (get1 x ind))) (if y (elemn y 3 0) 0)))
```

8.2.27 defun PARSE-rightBindingPowerOf

[get1 p??]
[elemn p??]

— defun PARSE-rightBindingPowerOf —

```
(defun |PARSE-rightBindingPowerOf| (x ind)
  (declare (special x ind))
  (let ((y (get1 x ind))) (if y (elemn y 4 105) 105)))
```

8.2.28 defun PARSE-getSemanticForm

[PARSE-Prefix p422]
[PARSE-Infix p423]

— defun PARSE-getSemanticForm —

```
(defun |PARSE-getSemanticForm| (x ind y)
  (declare (special x ind y))
  (or (and y (eval y)) (and (eq ind '|Nud|) (|PARSE-Prefix|))
      (and (eq ind '|Led|) (|PARSE-Infix|))))
```

8.2.29 defun PARSE-Prefix

[push-reduction p462]
[current-symbol p454]
[action p461]
[advance-token p456]

[optional p461]
 [PARSE-TokTail p424]
 [must p460]
 [PARSE-Expression p419]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-Prefix** —

```
(defun |PARSE-Prefix| ()
  (and (push-reduction '|PARSE-Prefix| (current-symbol))
        (action (advance-token)) (optional (|PARSE-TokTail|))
        (must (|PARSE-Expression|))
        (push-reduction '|PARSE-Prefix|
                          (list (pop-stack-2) (pop-stack-1))))))
```

8.2.30 defun PARSE-Infix

[push-reduction p462]
 [current-symbol p454]
 [action p461]
 [advance-token p456]
 [optional p461]
 [PARSE-TokTail p424]
 [must p460]
 [PARSE-Expression p419]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-Infix** —

```
(defun |PARSE-Infix| ()
  (and (push-reduction '|PARSE-Infix| (current-symbol))
        (action (advance-token)) (optional (|PARSE-TokTail|))
        (must (|PARSE-Expression|))
        (push-reduction '|PARSE-Infix|
                          (list (pop-stack-2) (pop-stack-2) (pop-stack-1) )))))
```

8.2.31 defun PARSE-TokTail

[current-symbol p454]
 [current-char p457]
 [char-eq p458]
 [copy-token p??]
 [action p461]
 [PARSE-Qualification p424]
 [\$boot p??]

— defun PARSE-TokTail —

```
(defun |PARSE-TokTail| ()
  (let (g1)
    (and (null $boot) (eq (current-symbol) '$)
      (or (alpha-char-p (current-char))
          (char-eq (current-char) "$")
          (char-eq (current-char) "%")
          (char-eq (current-char) "("))
      (action (setq g1 (copy-token prior-token)))
      (|PARSE-Qualification|) (action (setq prior-token g1)))))
```

—————

8.2.32 defun PARSE-Qualification

[match-advance-string p449]
 [must p460]
 [PARSE-Primary1 p428]
 [push-reduction p462]
 [dollarTran p459]
 [pop-stack-1 p522]

— defun PARSE-Qualification —

```
(defun |PARSE-Qualification| ()
  (and (match-advance-string "$") (must (|PARSE-Primary1|))
    (push-reduction '|PARSE-Qualification|
      (|dollarTran| (pop-stack-1) (pop-stack-1)))))
```

—————

8.2.33 defun PARSE-Reduction

[PARSE-ReductionOp p425]
 [must p460]
 [PARSE-Expr p420]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-Reduction** —

```
(defun |PARSE-Reduction| ()
  (and (|PARSE-ReductionOp|) (must (|PARSE-Expr| 1000))
    (push-reduction '|PARSE-Reduction|
      (list '|Reduce| (pop-stack-2) (pop-stack-1) )))))
```

—————

8.2.34 defun PARSE-ReductionOp

[get1 p??]
 [current-symbol p454]
 [match-next-token p454]
 [action p461]
 [advance-token p456]

— **defun PARSE-ReductionOp** —

```
(defun |PARSE-ReductionOp| ()
  (and (get1 (current-symbol) '|Led|)
    (match-next-token 'special-char (code-char 47))
    (push-reduction '|PARSE-ReductionOp| (current-symbol))
    (action (advance-token)) (action (advance-token)))))
```

—————

8.2.35 defun PARSE-Form

[match-advance-string p449]
 [bang p??]
 [optional p461]
 [must p460]
 [push-reduction p462]
 [pop-stack-1 p522]

[PARSE-Application p426]

— **defun PARSE-Form** —

```
(defun |PARSE-Form| ()
  (or (and (match-advance-string "iterate")
    (bang fil_test
      (optional
        (and (match-advance-string "from")
          (must (|PARSE-Label|))
          (push-reduction '|PARSE-Form|
            (list (pop-stack-1))))))
    (push-reduction '|PARSE-Form|
      (cons '|iterate| (append (pop-stack-1) nil))))
    (and (match-advance-string "yield") (must (|PARSE-Application|))
      (push-reduction '|PARSE-Form|
        (list '|yield| (pop-stack-1))))
    (|PARSE-Application|)))
```

8.2.36 defun PARSE-Application

[PARSE-Primary p428]

[optional p461]

[star p461]

[PARSE-Selector p427]

[PARSE-Application p426]

[push-reduction p462]

[pop-stack-2 p523]

[pop-stack-1 p522]

— **defun PARSE-Application** —

```
(defun |PARSE-Application| ()
  (and (|PARSE-Primary|) (optional (star opt_expr (|PARSE-Selector|)))
    (optional
      (and (|PARSE-Application|)
        (push-reduction '|PARSE-Application|
          (list (pop-stack-2) (pop-stack-1)))))))
```

[match-advance-string p449]
[must p460]
[PARSE-Name p435]

```
(defun |PARSE-Label| ()
  (and (match-advance-string "<<") (must (|PARSE-Name|))
        (must (match-advance-string ">>"))))
```

```
[current-symbol p454]
[char-ne p458]
[current-char p457]
[match-advance-string p449]
[must p460]
[PARSE-PrimaryNoFloat p428]
[push-reduction p462]
[pop-stack-2 p523]
[pop-stack-1 p522]
[PARSE-Float p429]
[PARSE-Primary p428]
[$boot p??]
```

```
(defun |PARSE-Selector| ()
  (declare (special $boot))
  (or (and nonblank (eq (current-symbol) '|.|)
    (char-ne (current-char) '| |) (match-advance-string ".")
    (must (|PARSE-PrimaryNoFloat|))
    (must (or (and $boot
      (push-reduction '|PARSE-Selector|
        (list 'elt (pop-stack-2) (pop-stack-1))))
      (push-reduction '|PARSE-Selector|
        (list (pop-stack-2) (pop-stack-1))))))
    (and (or (|PARSE-Float|)
      (and (match-advance-string ".")
        (must (|PARSE-Primary|))))
      (must (or (and $boot
        (push-reduction '|PARSE-Selector|
```

```

      (list 'elt (pop-stack-2) (pop-stack-1))))
(push-reduction '|PARSE-Selector|
  (list (pop-stack-2) (pop-stack-1))))))

```

8.2.39 defun PARSE-PrimaryNoFloat

[PARSE-Primary1 p428]
 [optional p461]
 [PARSE-TokTail p424]

— defun PARSE-PrimaryNoFloat —

```

(defun |PARSE-PrimaryNoFloat| ()
  (and (|PARSE-Primary1|) (optional (|PARSE-TokTail|))))

```

8.2.40 defun PARSE-Primary

[PARSE-Float p429]
 [PARSE-PrimaryNoFloat p428]

— defun PARSE-Primary —

```

(defun |PARSE-Primary| ()
  (or (|PARSE-Float|) (|PARSE-PrimaryNoFloat|)))

```

8.2.41 defun PARSE-Primary1

[PARSE-VarForm p434]
 [optional p461]
 [current-symbol p454]
 [PARSE-Primary1 p428]
 [must p460]
 [pop-stack-2 p523]
 [pop-stack-1 p522]
 [push-reduction p462]
 [PARSE-Quad p433]

```

[PARSE-String p433]
[PARSE-IntegerTok p432]
[PARSE-FormalParameter p433]
[match-string p448]
[PARSE-Data p436]
[match-advance-string p449]
[PARSE-Expr p420]
[PARSE-Sequence p439]
[PARSE-Enclosure p432]
[$boot p??]

```

— **defun PARSE-Primary1** —

```

(defun |PARSE-Primary1| ()
  (declare (special $boot))
  (or (and (|PARSE-VarForm|)
    (optional
      (and nonblank (eq (current-symbol) '|(|)
        (must (|PARSE-Primary1|))
        (push-reduction '|PARSE-Primary1|
          (list (pop-stack-2) (pop-stack-1))))))
    (|PARSE-Quad|) (|PARSE-String|) (|PARSE-IntegerTok|)
    (|PARSE-FormalParameter|)
    (and (match-string "'")
      (must (or (and $boot (|PARSE-Data|))
        (and (match-advance-string "'")
          (must (|PARSE-Expr| 999))
          (push-reduction '|PARSE-Primary1|
            (list 'quote (pop-stack-1)))))))
    (|PARSE-Sequence|) (|PARSE-Enclosure|))))

```

—————

8.2.42 **defun PARSE-Float**

```

[PARSE-FloatBase p430]
[must p460]
[PARSE-FloatExponent p431]
[push-reduction p462]
[make-float p??]
[pop-stack-4 p523]
[pop-stack-3 p523]
[pop-stack-2 p523]
[pop-stack-1 p522]

```

— **defun PARSE-Float** —

```
(defun |PARSE-Float| ()
  (and (|PARSE-FloatBase|)
    (must (or (and nonblank (|PARSE-FloatExponent|))
      (push-reduction '|PARSE-Float| 0)))
    (push-reduction '|PARSE-Float|
      (make-float (pop-stack-4) (pop-stack-2) (pop-stack-2)
        (pop-stack-1))))))
```

8.2.43 defun PARSE-FloatBase

```
[current-symbol p454]
[char-eq p458]
[current-char p457]
[char-ne p458]
[next-char p457]
[PARSE-IntegerTok p432]
[must p460]
[PARSE-FloatBasePart p430]
[PARSE-IntegerTok p432]
[push-reduction p462]
[PARSE-FloatBase digitp (vol5)]
```

— defun PARSE-FloatBase —

```
(defun |PARSE-FloatBase| ()
  (or (and (integerp (current-symbol)) (char-eq (current-char) ".")
    (char-ne (next-char) ".") (|PARSE-IntegerTok|)
    (must (|PARSE-FloatBasePart|)))
    (and (integerp (current-symbol))
      (char-eq (char-upcase (current-char)) 'e)
      (|PARSE-IntegerTok|) (push-reduction '|PARSE-FloatBase| 0)
      (push-reduction '|PARSE-FloatBase| 0))
    (and (digitp (current-char)) (eq (current-symbol) '|.|)
      (push-reduction '|PARSE-FloatBase| 0)
      (|PARSE-FloatBasePart|))))
```

8.2.44 defun PARSE-FloatBasePart

```
[match-advance-string p449]
[must p460]
```

— defun PARSE-FloatBasePart —

— defun PARSE-FloatExponent —

[illegible]

```

(push-reduction ' |PARSE-FloatExponent| 0)))
(and (identp (current-symbol))
  (setq g1 (floatexpid (current-symbol)))
  (action (advance-token))
  (push-reduction ' |PARSE-FloatExponent| g1))))

```

8.2.46 defun PARSE-Enclosure

```

[match-advance-string p449]
[must p460]
[PARSE-Expr p420]
[push-reduction p462]
[pop-stack-1 p522]

```

— defun PARSE-Enclosure —

```

(defun |PARSE-Enclosure| ()
  (or (and (match-advance-string "(")
    (must (or (and (|PARSE-Expr| 6)
      (must (match-advance-string ")"))
      (and (match-advance-string ")")
        (push-reduction ' |PARSE-Enclosure|
          (list ' |@Tuple|))))))
    (and (match-advance-string "{")
      (must (or (and (|PARSE-Expr| 6)
        (must (match-advance-string "}"))
        (push-reduction ' |PARSE-Enclosure|
          (cons ' |brace|
            (list (list ' |construct| (pop-stack-1))))))
        (and (match-advance-string "}")
          (push-reduction ' |PARSE-Enclosure|
            (list ' |brace|)))))))))

```

8.2.47 defun PARSE-IntegerTok

```

[parse-number p520]

```

— defun PARSE-IntegerTok —

```

(defun |PARSE-IntegerTok| () (parse-number))

```

8.2.48 defun PARSE-FormalParameter

[PARSE-FormalParameterTok p433]

— defun PARSE-FormalParameter —

```
(defun |PARSE-FormalParameter| () (|PARSE-FormalParameterTok|))
```

8.2.49 defun PARSE-FormalParameterTok

[parse-argument-designator p520]

— defun PARSE-FormalParameterTok —

```
(defun |PARSE-FormalParameterTok| () (parse-argument-designator))
```

8.2.50 defun PARSE-Quad

[match-advance-string p449]

[push-reduction p462]

[PARSE-GlyphTok p438]

[\$boot p??]

— defun PARSE-Quad —

```
(defun |PARSE-Quad| ()
  (or (and (match-advance-string "$")
            (push-reduction '|PARSE-Quad| '$))
      (and $boot (|PARSE-GlyphTok| '|.|)
            (push-reduction '|PARSE-Quad| '|.|))))
```

8.2.51 defun PARSE-String

[parse-spadstring p518]

— **defun PARSE-String** —

```
(defun |PARSE-String| () (parse-spadstring))
```

8.2.52 **defun PARSE-VarForm**

[PARSE-Name p435]
 [optional p461]
 [PARSE-Scripts p434]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-VarForm** —

```
(defun |PARSE-VarForm| ()
  (and (|PARSE-Name|)
    (optional
      (and (|PARSE-Scripts|)
        (push-reduction ' |PARSE-VarForm|
          (list ' |Scripts| (pop-stack-2) (pop-stack-1))))))
    (push-reduction ' |PARSE-VarForm| (pop-stack-1))))
```

8.2.53 **defun PARSE-Scripts**

[match-advance-string p449]
 [must p460]
 [PARSE-ScriptItem p435]

— **defun PARSE-Scripts** —

```
(defun |PARSE-Scripts| ()
  (and nonblank (match-advance-string "[" (must (|PARSE-ScriptItem|))
    (must (match-advance-string "]"")))))
```

8.2.54 defun PARSE-ScriptItem

[PARSE-Expr p420]
 [optional p461]
 [star p461]
 [match-advance-string p449]
 [must p460]
 [PARSE-ScriptItem p435]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-ScriptItem** —

```
(defun |PARSE-ScriptItem| ()
  (or (and (|PARSE-Expr| 90)
    (optional
      (and (star repeater
        (and (match-advance-string ";")
          (must (|PARSE-ScriptItem|))))
        (push-reduction '|PARSE-ScriptItem|
          (cons '|;|
            (cons (pop-stack-2)
              (append (pop-stack-1) nil)))))))
      (and (match-advance-string ";") (must (|PARSE-ScriptItem|))
        (push-reduction '|PARSE-ScriptItem|
          (list '|PrefixSC| (pop-stack-1)))))))
```

—————

8.2.55 defun PARSE-Name

[parse-identifier p519]
 [push-reduction p462]
 [pop-stack-1 p522]

— **defun PARSE-Name** —

```
(defun |PARSE-Name| ()
  (and (parse-identifier) (push-reduction '|PARSE-Name| (pop-stack-1))))
```

—————

8.2.56 defun PARSE-Data

[action p461]
 [PARSE-Sexpr p436]
 [push-reduction p462]
 [translabel p515]
 [pop-stack-1 p522]
 [labasoc p??]

— **defun PARSE-Data** —

```
(defun |PARSE-Data| ()
  (declare (special lablasoc))
  (and (action (setq lablasoc nil)) (|PARSE-Sexpr|)
    (push-reduction '|PARSE-Data|
      (list 'quote (translabel (pop-stack-1) lablasoc))))))
```

—————

8.2.57 defun PARSE-Sexpr

[PARSE-Sexpr1 p436]

— **defun PARSE-Sexpr** —

```
(defun |PARSE-Sexpr| ()
  (and (action (advance-token)) (|PARSE-Sexpr1|)))
```

—————

8.2.58 defun PARSE-Sexpr1

[PARSE-AnyId p438]
 [optional p461]
 [PARSE-NBGlyphTok p437]
 [must p460]
 [PARSE-Sexpr1 p436]
 [action p461]
 [pop-stack-2 p523]
 [nth-stack p524]
 [match-advance-string p449]
 [push-reduction p462]
 [PARSE-IntegerTok p432]
 [pop-stack-1 p522]

[PARSE-String p433]
 [bang p??]
 [star p461]
 [PARSE-GlyphTok p438]

— defun PARSE-Sexpr1 —

```
(defun |PARSE-Sexpr1| ()
  (or (and (|PARSE-AnyId|)
    (optional
      (and (|PARSE-NBGlyphTok| '=) (must (|PARSE-Sexpr1|))
        (action (setq lablasoc
          (cons (cons (pop-stack-2)
            (nth-stack 1))
            lablasoc))))))
    (and (match-advance-string "'") (must (|PARSE-Sexpr1|))
      (push-reduction '|PARSE-Sexpr1|
        (list 'quote (pop-stack-1))))
    (|PARSE-IntegerTok|)
    (and (match-advance-string "-") (must (|PARSE-IntegerTok|))
      (push-reduction '|PARSE-Sexpr1| (- (pop-stack-1))))
    (|PARSE-String|)
    (and (match-advance-string "<")
      (bang fil_test (optional (star repeater (|PARSE-Sexpr1|))))
      (must (match-advance-string ">"))
      (push-reduction '|PARSE-Sexpr1| (list2vec (pop-stack-1))))
    (and (match-advance-string "(")
      (bang fil_test
        (optional
          (and (star repeater (|PARSE-Sexpr1|))
            (optional
              (and (|PARSE-GlyphTok| '|.|)
                (must (|PARSE-Sexpr1|))
                (push-reduction '|PARSE-Sexpr1|
                  (nconc (pop-stack-2) (pop-stack-1))))))))
      (must (match-advance-string ")")))))
```

—————

8.2.59 defun PARSE-NBGlyphTok

[match-current-token p453]
 [action p461]
 [advance-token p456]
 [tok p410]

— defun PARSE-NBGlyphTok —

```
(defun |PARSE-NBGlyphTok| (|tok|)
  (declare (special |tok|))
  (and (match-current-token 'gliph |tok|) nonblank (action (advance-token)))))
```

8.2.60 defun PARSE-GlyphTok

```
[match-current-token p453]
[action p461]
[advance-token p456]
[tok p410]
```

— defun PARSE-GlyphTok —

```
(defun |PARSE-GlyphTok| (|tok|)
  (declare (special |tok|))
  (and (match-current-token 'gliph |tok|) (action (advance-token)))))
```

8.2.61 defun PARSE-AnyId

```
[parse-identifier p519]
[match-string p448]
[push-reduction p462]
[current-symbol p454]
[action p461]
[advance-token p456]
[parse-keyword p520]
```

— defun PARSE-AnyId —

```
(defun |PARSE-AnyId| ()
  (or (parse-identifier)
      (or (and (match-string "$")
                (push-reduction '|PARSE-AnyId| (current-symbol))
                (action (advance-token)))
          (parse-keyword)))))
```

8.2.62 defun PARSE-Sequence

[PARSE-OpenBracket p440]
 [must p460]
 [PARSE-Sequence1 p439]
 [match-advance-string p449]
 [PARSE-OpenBrace p440]
 [push-reduction p462]
 [pop-stack-1 p522]

— **defun PARSE-Sequence** —

```
(defun |PARSE-Sequence| ()
  (or (and (|PARSE-OpenBracket|) (must (|PARSE-Sequence1|))
    (must (match-advance-string "]")))
    (and (|PARSE-OpenBrace|) (must (|PARSE-Sequence1|))
    (must (match-advance-string "}")))
    (push-reduction '|PARSE-Sequence|
      (list '|brace| (pop-stack-1))))))
```

8.2.63 defun PARSE-Sequence1

[PARSE-Expression p419]
 [push-reduction p462]
 [pop-stack-2 p523]
 [pop-stack-1 p522]
 [optional p461]
 [PARSE-IteratorTail p441]

— **defun PARSE-Sequence1** —

```
(defun |PARSE-Sequence1| ()
  (and (or (and (|PARSE-Expression|)
    (push-reduction '|PARSE-Sequence1|
      (list (pop-stack-2) (pop-stack-1))))
    (push-reduction '|PARSE-Sequence1| (list (pop-stack-1))))
    (optional
      (and (|PARSE-IteratorTail|)
        (push-reduction '|PARSE-Sequence1|
          (cons 'collect
            (append (pop-stack-1)
              (list (pop-stack-1))))))))))
```

8.2.64 defun PARSE-OpenBracket

[getToken p452]
 [current-symbol p454]
 [eqcar p??]
 [push-reduction p462]
 [action p461]
 [advance-token p456]

— defun PARSE-OpenBracket —

```
(defun |PARSE-OpenBracket| ()
  (let (g1)
    (and (eq (|getToken| (setq g1 (current-symbol)))) '[])
    (must (or (and (eqcar g1 '|elt|)
                  (push-reduction '|PARSE-OpenBracket|
                                (list '|elt| (second g1) '|construct|)))
              (push-reduction '|PARSE-OpenBracket| '|construct|)))
    (action (advance-token))))))
```

—————

8.2.65 defun PARSE-OpenBrace

[getToken p452]
 [current-symbol p454]
 [eqcar p??]
 [push-reduction p462]
 [action p461]
 [advance-token p456]

— defun PARSE-OpenBrace —

```
(defun |PARSE-OpenBrace| ()
  (let (g1)
    (and (eq (|getToken| (setq g1 (current-symbol)))) '{})
    (must (or (and (eqcar g1 '|elt|)
                  (push-reduction '|PARSE-OpenBrace|
                                (list '|elt| (second g1) '|brace|)))
              (push-reduction '|PARSE-OpenBrace| '|construct|)))
    (action (advance-token))))))
```

—————

8.2.66 defun PARSE-IteratorTail

[match-advance-string p449]
 [bang p??]
 [optional p461]
 [star p461]
 [PARSE-Iterator p441]

— **defun PARSE-IteratorTail** —

```
(defun |PARSE-IteratorTail| ()
  (or (and (match-advance-string "repeat")
    (bang fil_test (optional (star repeater (|PARSE-Iterator|))))))
    (star repeater (|PARSE-Iterator|))))
```

—————

8.2.67 defun PARSE-Iterator

[match-advance-string p449]
 [must p460]
 [PARSE-Primary p428]
 [PARSE-Expression p419]
 [PARSE-Expr p420]
 [pop-stack-3 p523]
 [pop-stack-2 p523]
 [pop-stack-1 p522]
 [optional p461]

— **defun PARSE-Iterator** —

```
(defun |PARSE-Iterator| ()
  (or (and (match-advance-string "for") (must (|PARSE-Primary|))
    (must (match-advance-string "in"))
    (must (|PARSE-Expression|))
    (must (or (and (match-advance-string "by")
      (must (|PARSE-Expr| 200))
      (push-reduction '|PARSE-Iterator|
        (list 'inby (pop-stack-3)
          (pop-stack-2) (pop-stack-1))))
      (push-reduction '|PARSE-Iterator|
        (list 'in (pop-stack-2) (pop-stack-1))))))
    (optional
      (and (match-advance-string "|")
        (must (|PARSE-Expr| 111))
        (push-reduction '|PARSE-Iterator|
```

```

                (list '|\\| (pop-stack-1))))))
    (and (match-advance-string "while") (must (|PARSE-Expr| 190))
      (push-reduction '|PARSE-Iterator|
        (list 'while (pop-stack-1))))
    (and (match-advance-string "until") (must (|PARSE-Expr| 190))
      (push-reduction '|PARSE-Iterator|
        (list 'until (pop-stack-1))))))

```

8.2.68 The PARSE implicit routines

These symbols are not explicitly referenced in the source. Nevertheless, they are called during runtime. For example, PARSE-SemiColon is called in the chain:

```

PARSE-Enclosure {loc0=nil,loc1="(V ==> Vector; " } [ihs=35]
PARSE-Expr
  PARSE-LedPart
    PARSE-Operation
      PARSE-getSemanticForm
        PARSE-SemiColon

```

so there is a bit of indirection involved in the call.

8.2.69 defun PARSE-Suffix

```

[push-reduction p462]
[current-symbol p454]
[action p461]
[advance-token p456]
[optional p461]
[PARSE-TokTail p424]
[pop-stack-1 p522]

```

— defun PARSE-Suffix —

```

(defun |PARSE-Suffix| ()
  (and (push-reduction '|PARSE-Suffix| (current-symbol))
    (action (advance-token)) (optional (|PARSE-TokTail|))
    (push-reduction '|PARSE-Suffix|
      (list (pop-stack-1) (pop-stack-1))))))

```

8.2.70 defun PARSE-SemiColon

```
[match-advance-string p449]
[must p460]
[PARSE-Expr p420]
[push-reduction p462]
[pop-stack-2 p523]
[pop-stack-1 p522]
```

— **defun PARSE-SemiColon** —

```
(defun |PARSE-SemiColon| ()
  (and (match-advance-string ";")
    (must (or (|PARSE-Expr| 82)
              (push-reduction '|PARSE-SemiColon| '|/throwAway|)))
    (push-reduction '|PARSE-SemiColon|
      (list '|;| (pop-stack-2) (pop-stack-1)))))
```

—————

8.2.71 defun PARSE-Return

```
[match-advance-string p449]
[must p460]
[PARSE-Expression p419]
[push-reduction p462]
[pop-stack-1 p522]
```

— **defun PARSE-Return** —

```
(defun |PARSE-Return| ()
  (and (match-advance-string "return") (must (|PARSE-Expression|))
    (push-reduction '|PARSE-Return|
      (list '|return| (pop-stack-1)))))
```

—————

8.2.72 defun PARSE-Exit

```
[match-advance-string p449]
[must p460]
[PARSE-Expression p419]
[push-reduction p462]
[pop-stack-1 p522]
```

— defun PARSE-Exit —

```
(defun |PARSE-Exit| ()
  (and (match-advance-string "exit")
        (must (or (|PARSE-Expression|)
                   (push-reduction '|PARSE-Exit| '$NoValue|))))
  (push-reduction '|PARSE-Exit|
    (list '|exit| (pop-stack-1)))))
```

— —

8.2.73 defun PARSE-Leave

[match-advance-string p449]
 [PARSE-Expression p419]
 [must p460]
 [push-reduction p462]
 [PARSE-Label p427]
 [pop-stack-1 p522]

— defun PARSE-Leave —

```
(defun |PARSE-Leave| ()
  (and (match-advance-string "leave")
        (must (or (|PARSE-Expression|)
                   (push-reduction '|PARSE-Leave| '$NoValue|))))
  (must (or (and (match-advance-string "from")
                 (must (|PARSE-Label|))
                 (push-reduction '|PARSE-Leave|
                   (list '|leaveFrom| (pop-stack-1) (pop-stack-1)))))
        (push-reduction '|PARSE-Leave|
          (list '|leave| (pop-stack-1)))))
```

— —

8.2.74 defun PARSE-Seg

[PARSE-GlyphTok p438]
 [bang p??]
 [optional p461]
 [PARSE-Expression p419]
 [push-reduction p462]
 [pop-stack-2 p523]

[pop-stack-1 p522]

— **defun PARSE-Seg** —

```
(defun |PARSE-Seg| ()
  (and (|PARSE-GlyphTok| '|\...|)
    (bang fil_test (optional (|PARSE-Expression|)))
    (push-reduction '|PARSE-Seg|
      (list 'segment (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.75 **defun PARSE-Conditional**

[match-advance-string p449]
 [must p460]
 [PARSE-Expression p419]
 [bang p??]
 [optional p461]
 [PARSE-ElseClause p445]
 [push-reduction p462]
 [pop-stack-3 p523]
 [pop-stack-2 p523]
 [pop-stack-1 p522]

— **defun PARSE-Conditional** —

```
(defun |PARSE-Conditional| ()
  (and (match-advance-string "if") (must (|PARSE-Expression|))
    (must (match-advance-string "then")) (must (|PARSE-Expression|))
    (bang fil_test
      (optional
        (and (match-advance-string "else")
          (must (|PARSE-ElseClause|))))))
    (push-reduction '|PARSE-Conditional|
      (list '|if| (pop-stack-3) (pop-stack-2) (pop-stack-1))))))
```

—————

8.2.76 **defun PARSE-ElseClause**

[current-symbol p454]
 [PARSE-Conditional p445]
 [PARSE-Expression p419]

— defun PARSE-ElseClause —

```
(defun |PARSE-ElseClause| ()
  (or (and (eq (current-symbol) '|if|) (|PARSE-Conditional|))
      (|PARSE-Expression|)))
```

8.2.77 defun PARSE-Loop

```
[star p461]
[PARSE-Iterator p441]
[must p460]
[match-advance-string p449]
[PARSE-Expr p420]
[push-reduction p462]
[pop-stack-2 p523]
[pop-stack-1 p522]
```

— defun PARSE-Loop —

```
(defun |PARSE-Loop| ()
  (or (and (star repeater (|PARSE-Iterator|))
          (must (match-advance-string "repeat"))
          (must (|PARSE-Expr| 110))
          (push-reduction '|PARSE-Loop|
                        (cons 'repeat
                            (append (pop-stack-2) (list (pop-stack-1))))))
      (and (match-advance-string "repeat") (must (|PARSE-Expr| 110))
          (push-reduction '|PARSE-Loop|
                        (list 'repeat (pop-stack-1))))))
```

8.2.78 defun PARSE-LabelExpr

```
[PARSE-Label p427]
[must p460]
[PARSE-Expr p420]
[push-reduction p462]
[pop-stack-2 p523]
[pop-stack-1 p522]
```

— defun PARSE-LabelExpr —

```
(defun |PARSE-LabelExpr| ()
  (and (|PARSE-Label|) (must (|PARSE-Expr| 120))
    (push-reduction '|PARSE-LabelExpr|
      (list 'label (pop-stack-2) (pop-stack-1))))))
```

8.2.79 defun PARSE-FloatTok

```
[parse-number p520]
[push-reduction p462]
[pop-stack-1 p522]
[bfp- p??]
[$boot p??]
```

— defun PARSE-FloatTok —

```
(defun |PARSE-FloatTok| ()
  (declare (special $boot))
  (and (parse-number)
    (push-reduction '|PARSE-FloatTok|
      (if $boot (pop-stack-1) (bfp- (pop-stack-1))))))
```

8.3 The PARSE support routines

This section is broken up into 3 levels:

- String grabbing: Match String, Match Advance String
- Token handling: Current Token, Next Token, Advance Token
- Character handling: Current Char, Next Char, Advance Char
- Line handling: Next Line, Print Next Line
- Error Handling
- Floating Point Support
- Dollar Translation

8.3.1 String grabbing

String grabbing is the art of matching initial segments of the current line, and removing them from the line before the get tokenized if they match (or removing the corresponding current tokens).

8.3.2 defun match-string

The match-string function returns length of X if X matches initial segment of inputstream.

[unget-tokens p452]
 [skip-blanks p448]
 [line-past-end-p p605]
 [current-char p457]
 [initial-substring-p p450]
 [subseq p??]
 [\$line p603]
 [line p603]

— defun match-string —

```
(defun match-string (x)
  (unget-tokens) ; So we don't get out of synch with token stream
  (skip-blanks)
  (if (and (not (line-past-end-p current-line)) (current-char) )
      (initial-substring-p x
        (subseq (line-buffer current-line) (line-current-index current-line))))))
```

—————

8.3.3 defun skip-blanks

[current-char p457]
 [token-lookahead-type p449]
 [advance-char p??]

— defun skip-blanks —

```
(defun skip-blanks ()
  (loop (let ((cc (current-char)))
        (if (not cc) (return nil))
        (if (eq (token-lookahead-type cc) 'white)
            (if (not (advance-char)) (return nil))
            (return t)))))
```

— **initvars** —

```
(defvar Escape-Character #\\ "Superquoting character.")
```

8.3.4 defun token-lookahead-type

[Escape-Character p??]

— **defun token-lookahead-type** —

```
(defun token-lookahead-type (char)
  "Predicts the kind of token to follow, based on the given initial character."
  (declare (special Escape-Character))
  (cond
    ((not char)                                     'eof)
    ((or (char= char Escape-Character) (alpha-char-p char)) 'id)
    ((digitp char)                                     'num)
    ((char= char #\')                                  'string)
    ((char= char #\[)                                  'bstring)
    ((member char '(#\Space #\Tab #\Return) :test #'char=) 'white)
    (t                                                'special-char)))
```

8.3.5 defun match-advance-string

The match-string function returns length of X if X matches initial segment of inputstream. If it is successful, advance inputstream past X. [quote-if-string p450]

[current-token p455]
 [match-string p448]
 [line-current-index p??]
 [line-past-end-p p605]
 [line-current-char p??]
 [\$token p90]
 [\$line p603]

— **defun match-advance-string** —

```
(defun match-advance-string (x)
```

```

(let ((y (if (>= (length (string x))
               (length (string (quote-if-string (current-token))))
             (match-string x)
             nil))) ; must match at least the current token
  (when y
    (incf (line-current-index current-line) y)
    (if (not (line-past-end-p current-line))
        (setf (line-current-char current-line)
              (elt (line-buffer current-line)
                  (line-current-index current-line)))
        (setf (line-current-char current-line) #\space))
    (setq prior-token
          (make-token :symbol (intern (string x))
                     :type 'identifier
                     :nonblank nonblank))
    t)))

```

8.3.6 defun initial-substring-p

[string-not-greaterp p??]

— defun initial-substring-p —

```

(defun initial-substring-p (part whole)
  "Returns length of part if part matches initial segment of whole."
  (let ((x (string-not-greaterp part whole)))
    (and x (= x (length part)) x)))

```

8.3.7 defun quote-if-string

[token-type p??]
 [strconc p??]
 [token-symbol p??]
 [underscore p452]
 [token-nonblank p??]
 [pack p??]
 [escape-keywords p451]
 [\$boot p??]
 [\$spad p549]

— defun quote-if-string —

```

(defun quote-if-string (token)
  (declare (special $boot $spad))
  (when token ;only use token-type on non-null tokens
    (case (token-type token)
      (bstring (strconc "[" (token-symbol token) "]*"))
      (string (strconc "'" (token-symbol token) "'"))
      (spadstring (strconc "\" (underscore (token-symbol token)) "\"))
      (number (format nil "~v,'OD" (token-nonblank token)
                        (token-symbol token)))
      (special-char (string (token-symbol token)))
      (identifier (let ((id (symbol-name (token-symbol token)))
                        (pack (package-name (symbol-package
                                              (token-symbol token)))))
                    (if (or $boot $spad)
                        (if (string= pack "BOOT")
                            (escape-keywords (underscore id) (token-symbol token))
                            (concatenate 'string
                                           (underscore pack) "'" (underscore id)))
                        id)))
                    (token-symbol token))))))

```

8.3.8 defun escape-keywords

[\$keywords p??]

— defun escape-keywords —

```

(defun escape-keywords (pname id)
  (declare (special keywords))
  (if (member id keywords)
      (concatenate 'string "_" pname)
      pname))

```

8.3.9 defun isTokenDelimiter

NIL needed below since END_UNIT is not generated by current parser [current-symbol p454]

— defun isTokenDelimiter —

```

(defun |isTokenDelimiter| ()
  (member (current-symbol) '(\ end\_unit nil)))

```

8.3.10 defun underscore

[vector-push p??]

— defun underscore —

```
(defun underscore (string)
  (if (every #'alpha-char-p string)
      string
      (let* ((size (length string))
              (out-string (make-array (* 2 size)
                                      :element-type 'string-char
                                      :fill-pointer 0))
              next-char)
        (dotimes (i size)
          (setq next-char (char string i))
          (unless (alpha-char-p next-char) (vector-push #\_ out-string))
          (vector-push next-char out-string))
        out-string)))
```

8.3.11 Token Handling

8.3.12 defun getToken

[eqcar p??]

— defun getToken —

```
(defun |getToken| (x)
  (if (eqcar x '|elt|) (third x) x))
```

8.3.13 defun unget-tokens

```
[quote-if-string p450]
[line-current-segment p606]
[strconc p??]
[line-number p??]
```

[token-nonblank p??]
 [line-new-line p606]
 [line-number p??]
 [valid-tokens p91]

— **defun unget-tokens** —

```
(defun unget-tokens ()
  (case valid-tokens
    (0 t)
    (1 (let* ((cursym (quote-if-string current-token))
              (curline (line-current-segment current-line))
              (revised-line (strconc cursym curline (copy-seq " "))))
        (line-new-line revised-line current-line (line-number current-line))
        (setq nonblank (token-nonblank current-token))
        (setq valid-tokens 0)))
    (2 (let* ((cursym (quote-if-string current-token))
              (nextsym (quote-if-string next-token))
              (curline (line-current-segment Current-Line))
              (revised-line
               (strconc (if (token-nonblank current-token) " " " ")
                        cursym
                        (if (token-nonblank next-token) " " " ")
                        nextsym curline " ")))
        (setq nonblank (token-nonblank current-token))
        (line-new-line revised-line current-line (line-number current-line))
        (setq valid-tokens 0)))
    (t (error "How many tokens do you think you have?"))))
```

—

8.3.14 **defun match-current-token**

This returns the current token if it has EQ type and (optionally) equal symbol. [current-token p455]
 [match-token p454]

— **defun match-current-token** —

```
(defun match-current-token (type &optional (symbol nil))
  (match-token (current-token) type symbol))
```

—

8.3.15 defun match-token

[token-type p??]
[token-symbol p??]

— defun match-token —

```
(defun match-token (token type &optional (symbol nil))
  (when (and token (eq (token-type token) type))
    (if symbol
      (when (equal symbol (token-symbol token)) token)
      token)))
```

—————

8.3.16 defun match-next-token

This returns the next token if it has equal type and (optionally) equal symbol. [next-token p456]
[match-token p454]

— defun match-next-token —

```
(defun match-next-token (type &optional (symbol nil))
  (match-token (next-token) type symbol))
```

—————

8.3.17 defun current-symbol

[make-symbol-of p454]
[current-token p455]

— defun current-symbol —

```
(defun current-symbol ()
  (make-symbol-of (current-token)))
```

—————

8.3.18 defun make-symbol-of

[\$token p90]

— **defun make-symbol-of** —

```
(defun make-symbol-of (token)
  (let ((u (and token (token-symbol token))))
    (cond
      ((not u) nil)
      ((characterp u) (intern (string u)))
      (u))))
```

8.3.19 **defun current-token**

This returns the current token getting a new one if necessary. [try-get-token p455]
 [valid-tokens p91]
 [current-token p455]

— **defun current-token** —

```
(defun current-token ()
  (declare (special valid-tokens current-token))
  (if (> valid-tokens 0)
      current-token
      (try-get-token current-token)))
```

8.3.20 **defun try-get-token**

[get-token p457]
 [valid-tokens p91]

— **defun try-get-token** —

```
(defun try-get-token (token)
  (declare (special valid-tokens))
  (let ((tok (get-token token)))
    (when tok
      (incf valid-tokens)
      token)))
```

8.3.21 defun next-token

This returns the token after the current token, or NIL if there is none after. [try-get-token p455]

```
[current-token p455]
[valid-tokens p91]
[next-token p456]
```

— defun next-token —

```
(defun next-token ()
  (declare (special valid-tokens next-token))
  (current-token)
  (if (> valid-tokens 1)
      next-token
      (try-get-token next-token)))
```

—————

8.3.22 defun advance-token

This makes the next token be the current token. [current-token p455]

```
[copy-token p??]
[try-get-token p455]
[valid-tokens p91]
[current-token p455]
```

— defun advance-token —

```
(defun advance-token ()
  (current-token) ;don't know why this is needed
  (case valid-tokens
    (0 (try-get-token (current-token)))
    (1 (decf valid-tokens)
        (setq prior-token (copy-token current-token))
        (try-get-token current-token))
    (2 (setq prior-token (copy-token current-token))
        (setq current-token (copy-token next-token))
        (decf valid-tokens))))
```

—————

8.3.23 defvar \$XTokenReader— **initvars** —

```
(defvar XTokenReader 'get-meta-token "Name of tokenizing function")
```

—————

8.3.24 defun get-token

[XTokenReader p457]
 [XTokenReader p457]

— **defun get-token** —

```
(defun get-token (token)
  (funcall XTokenReader token))
```

—————

8.3.25 Character handling**8.3.26 defun current-char**

This returns the current character of the line, initially blank for an unread line. [line p603]
 [current-line p604]

— **defun current-char** —

```
(defun current-char ()
  (if (line-past-end-p current-line)
      #\return
      (line-current-char current-line)))
```

—————

8.3.27 defun next-char

This returns the character after the current character, blank if at end of line. The blank-at-end-of-line assumption is allowable because we assume that end-of-line is a token separator, which blank is equivalent to. [line-at-end-p p604]

```
[line-next-char p605]
[current-line p604]
```

— **defun next-char** —

```
(defun next-char ()
  (if (line-at-end-p current-line)
      #\return
      (line-next-char current-line)))
```

8.3.28 **defun char-eq**

— **defun char-eq** —

```
(defun char-eq (x y)
  (char= (character x) (character y)))
```

8.3.29 **defun char-ne**

— **defun char-ne** —

```
(defun char-ne (x y)
  (char/= (character x) (character y)))
```

8.3.30 **Error handling**

8.3.31 **defvar \$meta-error-handler**

— **initvars** —

```
(defvar meta-error-handler 'meta-meta-error-handler)
```

8.3.32 defun meta-syntax-error

[meta-error-handler p458]
 [meta-error-handler p458]

— **defun meta-syntax-error** —

```
(defun meta-syntax-error (&optional (wanted nil) (parsing nil))
  (declare (special meta-error-handler))
  (funcall meta-error-handler wanted parsing))
```

—————

8.3.33 Floating Point Support**8.3.34 defun floatexpid**

TPDHERE: The use of and in spadreduce is undefined. rewrite this to loop

[floatexpid identp (vol5)]
 [floatexpid pname (vol5)]
 [spadreduce p??]
 [collect p373]
 [step p??]
 [maxindex p??]
 [floatexpid digitp (vol5)]

— **defun floatexpid** —

```
(defun floatexpid (x &aux s)
  (when (and (identp x) (char= (char-upcase (elt (setq s (pname x)) 0)) #\E)
    (> (length s) 1)
    (spadreduce and 0 (collect (step i 1 1 (maxindex s))
                              (digitp (elt s i))))))
  (read-from-string s t nil :start 1)))
```

—————

8.3.35 Dollar Translation**8.3.36 defun dollarTran**

[\$InteractiveMode p??]

— **defun dollarTran** —

```
(defun |dollarTran| (dom rand)
  (let ((eltWord (if |$InteractiveMode| '|$elt| '|elt|)))
    (declare (special |$InteractiveMode|))
    (if (and (not (atom rand)) (cdr rand))
        (cons (list eltWord dom (car rand)) (cdr rand))
        (list eltWord dom rand))))
```

8.3.37 Applying metagrammatical elements of a production (e.g., Star).

- **must** means that if it is not present in the token stream, it is a syntax error.
- **optional** means that if it is present in the token stream, that is a good thing, otherwise don't worry (like [foo] in BNF notation).
- **action** is something we do as a consequence of successful parsing; it is inserted at the end of the conjunction of requirements for a successful parse, and so should return T.
- **sequence** consists of a head, which if recognized implies that the tail must follow. Following tail are actions, which are performed upon recognizing the head and tail.

8.3.38 defmacro Bang

If the execution of prod does not result in an increase in the size of the stack, then stack a NIL. Return the value of prod.

— defmacro bang —

```
(defmacro bang (lab prod)
  `(progn
    (setf (stack-updated reduce-stack) nil)
    (let* ((prodvalue ,prod) (updated (stack-updated reduce-stack)))
      (unless updated (push-reduction ',lab nil))
      prodvalue)))
```

8.3.39 defmacro must

[meta-syntax-error p459]

— defmacro must —

```
(defmacro must (dothis &optional (this-is nil) (in-rule nil))
  `(or ,dothis (meta-syntax-error ,this-is ,in-rule)))
```

8.3.40 defun action

— defun action —

```
(defun action (dothis) (or dothis t))
```

8.3.41 defun optional

— defun optional —

```
(defun optional (dothis) (or dothis t))
```

8.3.42 defmacro star

Succeeds if there are one or more of PROD, stacking as one unit the sub-reductions of PROD and labelling them with LAB. E.G., (Star IDs (parse-id)) with A B C will stack (3 IDs (A B C)), where (parse-id) would stack (1 ID (A)) when applied once. [stack-size p??]
 [push-reduction p462]
 [pop-stack-1 p522]

— defmacro star —

```
(defmacro star (lab prod)
  `(prog ((oldstacksize (stack-size reduce-stack)))
    (if (not ,prod) (return nil))
  loop
    (if (not ,prod)
      (let* ((newstacksize (stack-size reduce-stack))
              (number-of-new-reductions (- newstacksize oldstacksize)))
        (if (> number-of-new-reductions 0)
          (return (do ((i 0 (1+ i)) (accum nil))
```

```

                ((= i number-of-new-reductions)
                 (push-reduction ',lab accum)
                 (return t))
              (push (pop-stack-1) accum)))
    (return t)))
  (go loop))))

```

8.3.43 Stacking and retrieving reductions of rules.

8.3.44 `defvar $reduce-stack`

Stack of results of reduced productions. [`$stack` p88]

— `initvars` —

```

(defvar reduce-stack (make-stack) )

```

8.3.45 `defmacro reduce-stack-clear`

— `defmacro reduce-stack-clear` —

```

(defmacro reduce-stack-clear () '(stack-load nil reduce-stack))

```

8.3.46 `defun push-reduction`

[`stack-push` p89]
 [`make-reduction` p??]
 [`reduce-stack` p462]

— `defun push-reduction` —

```

(defun push-reduction (rule redn)
  (stack-push (make-reduction :rule rule :value redn) reduce-stack))

```

Chapter 9

Comment handlers

9.0.47 defun recordSignatureDocumentation

[recordDocumentation p464]
[postTransform p359]

— defun recordSignatureDocumentation —

```
(defun |recordSignatureDocumentation| (opSig lineno)
  (|recordDocumentation| (cdr (|postTransform| opSig)) lineno))
```

—————

9.0.48 defun recordAttributeDocumentation

[opOf p??]
[pname p??]
[upper-case-p p??]
[recordDocumentation p464]
[ifcdr p??]
[postTransform p359]

— defun recordAttributeDocumentation —

```
(defun |recordAttributeDocumentation| (arg lineno)
  (let (att name)
    (setq att (cadr arg))
    (setq name (|opOf| att))
    (cond
      ((upper-case-p (elt (pname name) 0)) nil)
```

```
(t
  (|recordDocumentation|
    (list name (cons '|attribute| (ifcdr (|postTransform| att)))) lineno))))
```

9.0.49 defun recordDocumentation

```
[recordHeaderDocumentation p464]
[collectComBlock p465]
[$maxSignatureLineNumber p??]
[$docList p??]
```

— defun recordDocumentation —

```
(defun |recordDocumentation| (key lineno)
  (let (u)
    (declare (special |$docList| |$maxSignatureLineNumber|))
    (|recordHeaderDocumentation| lineno)
    (setq u (|collectComBlock| lineno))
    (setq |$maxSignatureLineNumber| lineno)
    (setq |$docList| (cons (cons key u) |$docList|))))
```

9.0.50 defun recordHeaderDocumentation

```
[assocright p??]
[$maxSignatureLineNumber p??]
[$comblocklist p525]
[$headerDocumentation p??]
[$headerDocumentation p??]
[$comblocklist p525]
```

— defun recordHeaderDocumentation —

```
(defun |recordHeaderDocumentation| (lineno)
  (let (al)
    (declare (special |$headerDocumentation| |$maxSignatureLineNumber|
                      $comblocklist))
    (when (eql |$maxSignatureLineNumber| 0)
      (setq al
        (loop for p in $comblocklist
              when (or (null (car p)) (null lineno) (> lineno (car p)))
```

```

      collect p))
  (setq $comblocklist (setdifference $comblocklist al))
  (setq |$headerDocumentation| (assocright al))
  (when |$headerDocumentation| (setq |$maxSignatureLineNumber| 1))
  |$headerDocumentation|)))

```

9.0.51 defun collectComBlock

[collectAndDeleteAssoc p465]
 [\$comblocklist p525]

— defun collectComBlock —

```

(defun |collectComBlock| (x)
  (let (val u)
    (declare (special $comblocklist))
    (cond
      ((and (consp $comblocklist)
            (consp (qcar $comblocklist))
            (equal (qcaar $comblocklist) x))
       (setq val (qcdar $comblocklist))
       (setq u (append val (|collectAndDeleteAssoc| x)))
       (setq $comblocklist (cdr $comblocklist)))
      (t (|collectAndDeleteAssoc| x))))
  u)

```

9.0.52 defun collectAndDeleteAssoc

$u \text{ is } (.. (x . a) .. (x . b) ..) \implies (a \ b \ ..)$

deleting entries from u assumes that the first element is useless [\$comblocklist p525]

— defun collectAndDeleteAssoc —

```

(defun |collectAndDeleteAssoc| (x)
  (let (r res s)
    (declare (special $comblocklist))
    (maplist
      #'(lambda (y)
          (when (setq s (cdr y))
            (do ()

```

```

      ((null (and s (consp (car s)) (equal (qcar (car s)) x))) nil)
      (setq r (qcdr (car s)))
      (setq res (append res r))
      (setq s (cdr s))
      (rplacd y s))))
    $comblocklist)
  res))

```

9.0.53 defun finalizeDocumentation

```

[bright p??]
[sayMSG p??]
[stringimage p??]
[strconc p??]
[sayKeyedMsg p??]
[form2String p??]
[formatOpSignature p??]
[transDocList p468]
[msubst p??]
[assocleft p??]
[remdup p??]
[macroExpand p136]
[sublislis p??]
[$e p??]
[$lisplibForm p??]
[$docList p??]
[$op p??]
[$comblocklist p525]
[$FormalMapVariableList p250]

```

— defun finalizeDocumentation —

```

(defun |finalizeDocumentation| ()
  (labels (
    (fn (x env)
      (declare (special |$lisplibForm| |$FormalMapVariableList|))
      (cond
        ((atom x) (list x nil))
        (t
         (when (> (|#| x) 2) (setq x (take 2 x)))
         (sublislis |$FormalMapVariableList| (cdr |$lisplibForm|)
                   (|macroExpand| x env))))))
    (hn (u)
      ; ((op,sig,doc), ...) --> ((op ((sig doc) ...)) ...))

```

```

(let (opList op1 sig doc)
  (setq oplist (remdup (assocleft u)))
  (loop for op in opList
    collect
      (cons op
        (loop for item in u
          do (setq op1 (first item))
              (setq sig (second item))
              (setq doc (third item))
          when (equal op op1)
            collect
              (list sig doc))))))
(let (unusedCommentLineNumbers docList u noHeading attributes
      signatures name bigcnt op s litcnt a n r sig)
  (declare (special |$e| |$lisplibForm| |$docList| |$op| |$comblocklist|))
  (setq unusedCommentLineNumbers
    (loop for x in $comblocklist
      when (cdr x)
        collect x))
  (setq docList (msubst '$ '% (|transDocList| |$op| |$docList|)))
  (cond
    ((setq u
      (loop for item in docList
        when (null (cdr item))
          collect (car item)))
      (loop for y in u
        do
          (cond
            ((eq y '|constructor|) (setq noHeading t))
            ((and (consp y) (consp (qcdr y)) (eq (qcddr y) nil)
              (consp (qcadr y)) (eq (qcaadr y) '|attribute|))
              (setq attributes (cons (cons (qcar y) (qcdadr y)) attributes)))
            (t (setq signatures (cons y signatures))))))
      (setq name (CAR |$lisplibForm|))
      (when (or noHeading signatures attributes unusedCommentLineNumbers)
        (|sayKeyedMsg| 'S2CD0001 nil)
        (setq bigcnt 1)
        (when (or noHeading signatures attributes)
          (|sayKeyedMsg| 'S2CD0002 (list (strconc (stringimage bigcnt) ".") name))
          (setq bigcnt (1+ bigcnt))
          (setq litcnt 1)
          (when noHeading
            (|sayKeyedMsg| 'S2CD0003
              (list (strconc "(" (stringimage litcnt) ")") name))
            (setq litcnt (1+ litcnt)))
          (when signatures
            (|sayKeyedMsg| 'S2CD0004
              (list (strconc "(" (stringimage litcnt) ")"))
              (setf litcnt (1+ litcnt))
              (loop for item in signatures

```

```

do
  (setq op (first item))
  (setq sig (second item))
  (setq s (|formatOpSignature| op sig))
  (|sayMSG|
   (if (atom s)
       (list '|%x9| s)
       (cons '|%x9| s))))))
(when attributes
  (|sayKeyedMsg| 'S2CD0005
   (list (strconc "(" (stringimage litcnt) ")"))))
  (setq litcnt (1+ litcnt))
  (DO ((G166491 attributes
        (CDR G166491))
      (x NIL)
      ((OR (ATOM G166491)
            (PROGN
             (SETQ x (CAR G166491))
             NIL))
       NIL)
      (SEQ (EXIT
            (PROGN
             (setq a (|form2String| x))
             (|sayMSG|
              (COND
               ((ATOM a)
                (CONS '|%x9| (CONS a NIL)))
               ('T (CONS '|%x9| a))))))))))
  (when unusedCommentLineNumbers
    (|sayKeyedMsg| 'S2CD0006
     (list (strconc (stringimage bigcnt) ".") name))
    (loop for item in unusedCommentLineNumbers
          do
            (setq r (second item))
            (|sayMSG| (cons " " (append (|bright| n) (list " " r))))))
  (hn
   (loop for item in docList
         collect (append (fn (car item) |$e|) (cdr item)))))

```

9.1 Transformation of ++ comments

9.1.1 defun transDocList

```

[sayBrightly p??]
[transDoc p469]

```

```
[checkDocError p478]
[checkDocError1 p478]
[$constructorName p??]
```

— defun transDocList —

```
(defun |transDocList| (|$constructorName| doclist)
  (declare (special |$constructorName|))
  (let (commentList conEntry acc)
    (|sayBrightly|
      (list " Processing " |$constructorName| " for Browser database:"))
    (setq commentList (|transDoc| |$constructorName| doclist))
    (setq acc nil)
    (loop for entry in commentList
      do
        (cond
          ((and (consp entry) (eq (qcar entry) '|constructor|)
              (consp (qcdr entry)) (eq (qcddr entry) nil))
            (if conEntry
              (|checkDocError| (list "Spurious comments: " (qcadr entry)))
              (setq conEntry entry)))
            (t (setq acc (cons entry acc))))))
    (if conEntry
      (cons conEntry acc)
      (progn
        (|checkDocError1| (list "Missing Description"))
        acc))))
```

—————

9.1.2 defun transDoc

```
[checkDocError1 p478]
[checkTrim p506]
[checkExtract p507]
[transformAndRecheckComments p470]
[nreverse p??]
[$x p??]
[$attribute? p??]
[$x p??]
[$attribute? p??]
[$argl p??]
```

— defun transDoc —

```
(defun |transDoc| (conname doclist)
```

```

(declare (ignore conname))
(let (|$x| |$attribute?| |$argl| rlist lines u v longline acc)
  (declare (special |$x| |$attribute?| |$argl|))
  (setq |$x| nil)
  (setq rlist (reverse doclist))
  (loop for item in rlist
    do
      (setq |$x| (car item))
      (setq lines (cdr item))
      (setq |$attribute?|
        (and (consp |$x|) (consp (qcdr |$x|)) (eq (qcddr |$x|) nil)
              (consp (qcadr |$x|)) (eq (qcdadr |$x|) nil)
              (eq (qcaadr |$x|) '|attribute|)))
      (cond
        ((null lines)
         (unless |$attribute?| (|checkDocError1| (list "Not documented!!!!"))))
        (t
         (setq u
              (|checkTrim| |$x|
               (cond
                ((stringp lines) (list lines))
                ((eq |$x| '|constructor|) (car lines))
                (t lines))))
              (setq |$argl| nil) ;; possibly unused -- tpd
              (setq longline
                   (cond
                    ((eq |$x| '|constructor|)
                     (setq v
                          (or
                           (|checkExtract| "Description:" u)
                           (and u (|checkExtract| "Description:"
                                                  (cons (strconc "Description: " (car u)) (cdr u))))))
                     (|transformAndRecheckComments| '|constructor| (or v u)))
                    (t (|transformAndRecheckComments| |$x| u))))
              (setq acc (cons (list |$x| longline) acc))))
        (nreverse acc)))

```

9.1.3 defun transformAndRecheckComments

```

[sayBrightly p??]
[$exposeFlagHeading p??]
[$checkingXmptex? p??]
[$x p??]
[$name p??]
[$origin p??]

```

```
[$recheckingFlag p??]
[$exposeFlagHeading p??]
```

— **defun transformAndRecheckComments** —

```
(defun |transformAndRecheckComments| (name lines)
  (let (|$x| |$name| |$origin| |$recheckingFlag| |$exposeFlagHeading| u)
    (declare (special |$x| |$name| |$origin| |$recheckingFlag|
                      |$exposeFlagHeading| |$exposeFlag| |$checkingXmptex?|))
    (setq |$checkingXmptex?| nil)
    (setq |$x| name)
    (setq |$name| ' |GlossaryPage|)
    (setq |$origin| ' |gloss|)
    (setq |$recheckingFlag| nil)
    (setq |$exposeFlagHeading| (list "-----" name "-----"))
    (unless |$exposeFlag| (|sayBrightly| |$exposeFlagHeading|))
    (setq u (|checkComments| name lines))
    (setq |$recheckingFlag| t)
    (|checkRewrite| name (list u))
    (setq |$recheckingFlag| nil)
    u))
```

—————

9.1.4 defun checkRewrite

```
[checkRemoveComments p500]
[checkAddIndented p??]
[checkGetArgs p504]
[newString2Words p503]
[checkAddSpaces p512]
[checkSplit2Words p482]
[checkAddMacros p501]
[checkTexht p476]
[checkArguments p486]
[checkFixCommonProblem p508]
[checkRecordHash p472]
[checkDecorateForHt p477]
[$checkErrorFlag p??]
[$argl p??]
[$checkingXmptex? p??]
```

— **defun checkRewrite** —

```
(defun |checkRewrite| (name lines)
  (declare (ignore name)))
```

```

(prog (|$checkErrorFlag| margin w verbatim u2 okBefore u)
(declare (special |$checkErrorFlag| |$argl| |$checkingXmptex?|))
(setq |$checkErrorFlag| t)
(setq margin 0)
(setq lines (|checkRemoveComments| lines))
(setq u lines)
(when |$checkingXmptex?|
  (setq u
    (loop for x in u
      collect (|checkAddIndented| x margin))))
(setq |$argl| (|checkGetArgs| (car u)))
(setq u2 nil)
(setq verbatim nil)
(loop for x in u
  do
    (setq w (|newString2Words| x))
    (cond
      (verbatim
        (cond
          ((and w (equal (car w) "\\end{verbatim}"))
            (setq verbatim nil)
            (setq u2 (append u2 w)))
          (t
            (setq u2 (append u2 (list x)))))
          ((and w (equal (car w) "\\begin{verbatim}"))
            (setq verbatim t)
            (setq u2 (append u2 w)))
          (t (setq u2 (append u2 w)))))
    (setq u u2)
    (setq u (|checkAddSpaces| u))
    (setq u (|checkSplit2Words| u))
    (setq u (|checkAddMacros| u))
    (setq u (|checkTexht| u))
    (setq okBefore (null |$checkErrorFlag|))
    (|checkArguments| u)
    (when |$checkErrorFlag| (setq u (|checkFixCommonProblem| u)))
    (|checkRecordHash| u)
    (|checkDecorateForHt| u)))

```

9.1.5 defun checkRecordHash

```

[member p??]
[checkLookForLeftBrace p487]
[checkLookForRightBrace p487]
[ifcdr p??]

```

```

[intern p??]
[hget p??]
[hput p??]
[checkGetLispFunctionName p??]
[checkGetStringBeforeRightBrace p??]
[checkGetParse p475]
[checkDocError p478]
[opOf p??]
[spadSysChoose p??]
[checkNumOfArgs p499]
[checkIsValidType p??]
[form2HtString p??]
[getl p??]
[$charBack p??]
[$HTlinks p??]
[$htHash p??]
[$HTlisplinks p??]
[$lispHash p??]
[$glossHash p??]
[$currentSysList p??]
[$setOptions p??]
[$sysHash p??]
[$name p??]
[$origin p??]
[$sysHash p??]
[$glossHash p??]
[$lispHash p??]
[$htHash p??]

```

— defun checkRecordHash —

```

(defun |checkRecordHash| (u)
  (let (p q htname entry s parse n key x)
    (declare (special |$origin| |$name| |$sysHash| |$setOptions| |$glossHash|
                      |$currentSysList| |$lispHash| |$HTlisplinks| |$htHash|
                      |$HTlinks| |$charBack|))
    (loop while u
      do
        (setq x (car u))
        (when (and (stringp x) (equal (elt x 0) |$charBack|))
          (cond
            ((and (|member| x |$HTlinks|)
                  (setq u (|checkLookForLeftBrace| (ifcdr u)))
                  (setq u (|checkLookForRightBrace| (ifcdr u)))
                  (setq u (|checkLookForLeftBrace| (ifcdr u)))
                  (setq u (ifcdr u)))
              (setq htname (|intern| (ifcar u)))

```

```

(setq entry (or (hget |$htHash| htname) (list nil)))
(hput |$htHash| htname
 (cons (car entry) (cons (cons |$name| |$origin|) (cdr entry)))))
((and (|member| x |$HTlispLinks|)
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (|checkLookForRightBrace| (ifcdr u)))
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (ifcdr u)))
 (setq htname
  (|intern|
   (|checkGetLispFunctionName|
    (|checkGetStringBeforeRightBrace| u))))
(setq entry (or (hget |$lispHash| htname) (list nil)))
(hput |$lispHash| htname
 (cons (car entry) (cons (cons |$name| |$origin|) (cdr entry)))))
((and (or (setq p (|member| x '("\gloss" "\spadglos")))
          (setq q (|member| x '("\glossSee" "\spadglosSee"))))
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (ifcdr u)))
 (when q
  (setq u (|checkLookForRightBrace| u))
  (setq u (|checkLookForLeftBrace| (ifcdr u)))
  (setq u (ifcdr u)))
 (setq htname (|intern| (|checkGetStringBeforeRightBrace| u)))
 (setq entry
  (or (hget |$glossHash| htname) (list nil)))
  (hput |$glossHash| htname
   (cons (car entry) (cons (cons |$name| |$origin|) (cdr entry)))))
((and (boot-equal x "\spadsys")
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (ifcdr u)))
 (setq s (|checkGetStringBeforeRightBrace| u))
 (when (char= (elt s 0) #\)) (setq s (substring s 1 nil)))
 (setq parse (|checkGetParse| s))
 (cond
  ((null parse)
   (|checkDocError| (list "Unparseable \spadtype: " s)))
  ((null (|member| (|opOf| parse) |$currentSysList|))
   (|checkDocError| (list "Bad system command: " s)))
  ((or (atom parse)
       (null (and (consp parse) (eq (qcar parse) '|set|)
                  (consp (qcdr parse))
                  (eq (qcddr parse) nil))))
   '|ok|)
  ((null (|spadSysChoose| |$setOptions| (qcadr parse)))
   (progn
    (|checkDocError| (list "Incorrect \spadsys: " s))
    (setq entry (or (hget |$sysHash| htname) (list nil)))
    (hput |$sysHash| htname
     (cons (car entry) (cons (cons |$name| |$origin|) (cdr entry)))))))

```

```

((and (boot-equal x "\\spadtype")
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (ifcdr u)))
 (setq s (|checkGetStringBeforeRightBrace| u))
 (setq parse (|checkGetParse| s))
 (cond
  ((null parse)
   (|checkDocError| (list "Unparseable \\spadtype: " s)))
  (t
   (setq n (|checkNumOfArgs| parse))
   (cond
    ((null n)
     (|checkDocError| (list "Unknown \\spadtype: " s)))
    ((and (atom parse) (> n 0))
     '|skip|)
    ((null (setq key (|checkIsValidType| parse)))
     (|checkDocError| (list "Unknown \\spadtype: " s)))
    ((atom key) '|ok|)
    (t
     (|checkDocError|
      (list "Wrong number of arguments: " (|form2HtString| key)))))))
((and (|member| x '("\\spadop" "\\keyword"))
      (setq u (|checkLookForLeftBrace| (ifcdr u)))
      (setq u (ifcdr u)))
 (setq x (|intern| (|checkGetStringBeforeRightBrace| u)))
 (when (null (or (get1 x '|Led|) (get1 x '|Nud|)))
  (|checkDocError| (list "Unknown \\spadop: " x))))
(pop u)
'|done|))

```

9.1.6 defun checkGetParse

[ncParseFromString p??]
[removeBackslashes p476]

— defun checkGetParse —

```

(defun |checkGetParse| (s)
  (|ncParseFromString| (|removeBackslashes| s)))

```

9.1.7 defun removeBackslashes

```
[charPosition p??]
[removeBackslashes p476]
[strconc p??]
[length p??]
[$charBack p??]
```

— defun removeBackslashes —

```
(defun |removeBackslashes| (s)
  (let (k)
    (declare (special |$charBack|))
    (cond
      ((string= s "") "")
      ((> (|#| s) (setq k (|charPosition| |$charBack| s 0)))
        (if (eql k 0)
          (|removeBackslashes| (substring s 1 nil))
          (strconc (substring s 0 k)
                    (|removeBackslashes| (substring s (1+ k) nil))))))
    (t s))))
```

—————

9.1.8 defun checkTexht

```
[ifcar p??]
[checkDocError p478]
[nequal p??]
[$charRbrace p??]
[$charLbrace p??]
```

— defun checkTexht —

```
(defun |checkTexht| (u)
  (let (count y x acc)
    (declare (special |$charRbrace| |$charLbrace|))
    (setq count 0)
    (loop while u
      do
        (setq x (car u))
        (when (and (string= x "\\texht") (setq u (ifcdr u)))
          (when (null (equal (ifcar u) |$charLbrace|))
            (|checkDocError| "First left brace after \\texht missing"))
          ; drop first argument including braces of texht
          (setq count 1)))
```

```

(do ()
  ((null (or (nequal (setq y (ifcar (setq u (cdr u)))) |$charRbrace|)
             (> count 1)))
   nil)
  (when (equal y |$charLbrace|) (setq count (1+ count)))
  (when (equal y |$charRbrace|) (setq count (1- count))))
; drop first right brace of 1st arg
(setq x (ifcar (setq u (cdr u))))
(when (and (string= x "\\httex") (setq u (ifcdr u))
           (equal (ifcar u) |$charLbrace|))
  (setq acc (cons (ifcar u) acc))
  (do ()
    ((null (nequal (setq y (ifcar (setq u (cdr u)))) |$charRbrace|))
     nil)
    (setq acc (cons y acc)))
  (setq acc (cons (ifcar u) acc)) ; left brace: add it
  (setq x (ifcar (setq u (cdr u)))) ; left brace: forget it
  (do ()
    ((null (nequal (ifcar (setq u (cdr u)))) |$charRbrace|))
     nil)
    ' |skip|)
  ; forget right brace; move to next character
  (setq x (ifcar (setq u (cdr u)))))
(setq acc (cons x acc))
(pop u)
(nreverse acc)))

```

9.1.9 defun checkDecorateForHt

```

[checkDocError p478]
[member p??]
[$checkingXmptex? p??]
[$charRbrace p??]
[$charLbrace p??]

```

— defun checkDecorateForHt —

```

(defun |checkDecorateForHt| (u)
  (let (x count spadflag)
    (declare (special |$checkingXmptex?| |$charRbrace| |$charLbrace|))
    (setq count 0)
    (setq spadflag nil)
    (loop while u
      do
        (setq x (car u))

```

```

(when (equal x "\\em")
  (if (> count 0)
    (setq spadflag (1- count))
    (|checkDocError| (list "\\em must be enclosed in braces"))))
(cond
  ((|member| x '("\\s" "\\spadop" "\\spadtype" "\\spad" "\\spadpaste"
    "\\spadcommand" "\\footnote"))
    (setq spadflag count))
  ((equal x |$charLbrace|)
    (setq count (1+ count)))
  ((equal x |$charRbrace|)
    (setq count (1- count)))
  (when (equal spadflag count) (setq spadflag nil)))
((and (null spadflag) (|member| x '("+ " *" "=" "==" "->"))
  (when |$checkingXmptex?|
    (|checkDocError| (list '|Symbol| x " appearing outside \\spad{}"))))
  (t nil))
(when (or (equal x "$") (equal x "%"))
  (|checkDocError| (list "Unescaped " x)))
(pop u)
u))

```

9.1.10 defun checkDocError1

[checkDocError p478]
 [\$compileDocumentation p174]

— defun checkDocError1 —

```

(defun |checkDocError1| (u)
  (declare (special |$compileDocumentation|))
  (if (and (boundp '|$compileDocumentation|) |$compileDocumentation|)
    nil
    (|checkDocError| u)))

```

9.1.11 defun checkDocError

[checkDocMessage p479]
 [concat p??]
 [saybrightly1 p??]
 [sayBrightly p??]

```

[$checkErrorFlag p??]
[$recheckingFlag p??]
[$constructorName p??]
[$exposeFlag p??]
[$exposeFlagHeading p??]
[$outStream p??]
[$checkErrorFlag p??]
[$exposeFlagHeading p??]

```

— **defun checkDocError** —

```

(defun |checkDocError| (u)
  (let (msg)
    (declare (special |$outStream| |$exposeFlag| |$exposeFlagHeading|
                     |$constructorName| |$recheckingFlag| |$checkErrorFlag|))
    (setq |$checkErrorFlag| t)
    (setq msg
      (cond
        (|$recheckingFlag|
         (if |$constructorName|
             (|checkDocMessage| u)
             (concat| "> " u)))
        (|$constructorName| (|checkDocMessage| u))
        (t u)))
    (when (and |$exposeFlag| |$exposeFlagHeading|)
      (saybrightly1 |$exposeFlagHeading| |$outStream|)
      (|sayBrightly| |$exposeFlagHeading|)
      (setq |$exposeFlagHeading| nil))
    (|sayBrightly| msg)
    (when |$exposeFlag| (saybrightly1 msg |$outStream|))))

```

—————

9.1.12 defun checkDocMessage

```

[getdatabase p??]
[whoOwns p480]
[concat p??]
[$x p??]
[$constructorName p??]

```

— **defun checkDocMessage** —

```

(defun |checkDocMessage| (u)
  (let (sourcefile person middle)
    (declare (special |$constructorName| |$x|))

```

```

(setq sourcefile (getdatabase |$constructorName| 'sourcefile))
(setq person (or (|whoOwns| |$constructorName|) "---"))
(setq middle
  (if (boundp '|$x|)
    (list "(" |$x| "): ")
    (list ": ")))
(|concat| person ">" sourcefile "-->" |$constructorName| middle u)))

```

9.1.13 defun whoOwns

This function always returns nil in the current system. Since it has no side effects we define it to return nil. [getdatabase p??]

```

[strconc p??]
[awk p??]
[shut p??]
[$exposeFlag p??]

```

— defun whoOwns —

```

(defun |whoOwns| (con) nil)
; (let (filename quoteChar instream value)
; (declare (special |$exposeFlag|))
; (cond
; ((null |$exposeFlag|) nil)
; (t
; (setq filename (getdatabase con 'sourcefile))
; (setq quoteChar #\")
; (obey (strconc "awk '$2 == \" quoteChar filename quoteChar
;           \" {print $1}' whofiles > /tmp/temp"))
; (setq instream (make-instream "/tmp/temp"))
; (setq value (unless (eofp instream) (readline instream)))
; (shut instream)
; value))))

```

9.1.14 defun checkComments

```

[checkGetMargin p493]
[nequal p??]
[checkTransformFirsts p488]
[checkIndentedLines p502]
[checkGetArgs p504]

```

```

[newString2Words p503]
[checkAddSpaces p512]
[checkIeEg p494]
[checkSplit2Words p482]
[checkBalance p483]
[checkArguments p486]
[checkFixCommonProblems p??]
[checkDecorate p508]
[strconc p??]
[checkAddPeriod p483]
[pp p??]
[$attribute? p??]
[$checkErrorFlag p??]
[$argl p??]
[$checkErrorFlag p??]

```

— defun checkComments —

```

(defun |checkComments| (nameSig lines)
  (let (|$checkErrorFlag| margin w verbatim u2 okBefore u v res)
    (declare (special |$checkErrorFlag| |$argl| |$attribute?|))
    (setq |$checkErrorFlag| nil)
    (setq margin (|checkGetMargin| lines))
    (cond
      ((and (or (null (boundp '|$attribute?|)) (null |$attribute?|))
            (nequal nameSig '|constructor|)))
        (setq lines
          (cons
            (|checkTransformFirst| (car nameSig) (car lines) margin)
            (cdr lines))))))
    (setq u (|checkIndentedLines| lines margin))
    (setq |$argl| (|checkGetArgs| (car u)))
    (setq u2 nil)
    (setq verbatim nil)
    (loop for x in u
      do (setq w (|newString2Words| x))
        (cond
          (verbatim
            (cond
              ((and w (equal (car w) "\\end{verbatim}"))
                (setq verbatim nil)
                (setq u2 (append u2 w)))
              (t
                (setq u2 (append u2 (list x))))))
            ((and w (equal (car w) "\\begin{verbatim}"))
              (setq verbatim t)
              (setq u2 (append u2 w)))
              (t (setq u2 (append u2 w))))))

```

```

(setq u u2)
(setq u (|checkAddSpaces| u))
(setq u (|checkIsEg| u))
(setq u (|checkSplit2Words| u))
(|checkBalance| u)
(setq okBefore (null |$checkErrorFlag|))
(|checkArguments| u)
(when |$checkErrorFlag| (setq u (|checkFixCommonProblem| u)))
(setq v (|checkDecorate| u))
(setq res
  (let ((result ""))
    (loop for y in v
      do (setq result (strconc result y)))
    result))
(setq res (|checkAddPeriod| res))
(when |$checkErrorFlag| (|pp| res))
res))

```

9.1.15 defun checkSplit2Words

[checkSplitBrace p495]

— defun checkSplit2Words —

```

(defun |checkSplit2Words| (u)
  (let (x verbatim z acc)
    (setq acc nil)
    (loop while u
      do
        (setq x (car u))
        (setq acc
          (cond
            ((string= x "\\end{verbatim}")
             (setq verbatim nil)
             (cons x acc))
            (verbatim (cons x acc))
            ((string= x "\\begin{verbatim}")
             (setq verbatim t)
             (cons x acc))
            ((setq z (|checkSplitBrace| x))
             (append (nreverse z) acc))
            (t (cons x acc))))
        (pop u))
    (nreverse acc)))

```

9.1.16 defun checkAddPeriod

```
[setelt p??]
[maxindex p??]
```

— **defun checkAddPeriod** —

```
(defun |checkAddPeriod| (s)
  (let (m lastChar)
    (setq m (maxindex s))
    (setq lastChar (elt s m))
    (cond
      ((or (char= lastChar #\!) (char= lastChar #\?) (char= lastChar #\.) s)
        ((or (char= lastChar #\,) (char= lastChar #\;))
          (setelt s m #\.)
          s)
        (t s))))
```

9.1.17 defun checkBalance

```
[checkBeginEnd p484]
[assoc p??]
[rassoc p??]
[nequal p??]
[checkDocError p478]
[checkSayBracket p486]
[nreverse p??]
[$checkPrenAlist p??]
```

— **defun checkBalance** —

```
(defun |checkBalance| (u)
  (let (x openClose open top restStack stack)
    (declare (special |$checkPrenAlist|))
    (|checkBeginEnd| u)
    (setq stack nil)
    (loop while u
      do
        (setq x (car u))
        (cond
          ((setq openClose (|assoc| x |$checkPrenAlist|))
            (setq stack (cons (car openClose) stack))))
```

```

((setq open (|rassoc| x |$checkPrenAlist|))
 (cond
  ((consp stack)
   (setq top (qcar stack))
   (setq restStack (qcdr stack))
   (when (nequal open top)
    (|checkDocError|
     (list "Mismatch: left " (|checkSayBracket| top)
           " matches right " (|checkSayBracket| open))))
   (setq stack restStack))
  (t
   (|checkDocError|
    (list "Missing left " (|checkSayBracket| open))))))
(pop u))
(when stack
 (loop for x in (nreverse stack)
  do
   (|checkDocError| (list "Missing right " (|checkSayBracket| x)))))
u))

```

9.1.18 defun checkBeginEnd

```

[length p??]
[hget p??]
[ifcar p??]
[ifcdr p??]
[substring? p??]
[checkDocError p478]
[member p??]
[$charRbrace p??]
[$charLbrace p??]
[$beginEndList p??]
[$htMacroTable p??]
[$charBack p??]

```

— defun checkBeginEnd —

```

(defun |checkBeginEnd| (u)
 (let (x y beginEndStack)
 (declare (special |$charRbrace| |$charLbrace| |$beginEndList| |$charBack|
                  |$htMacroTable|))
 (loop while u
  do
   (setq x (car u))
   (cond

```

```

((and (stringp x) (equal (elt x 0) |$charBack|) (> (|#| x) 2)
  (null (hget |$htMacroTable| x)) (null (equal x "\\spadignore")))
  (equal (ifcar (ifcdr u)) |$charLbrace|)
  (null (or (|substring?| "\\radiobox" x 0)
    (|substring?| "\\inputbox" x 0))))
  (|checkDocError| (list '|Unexpected HT command: | x)))
(equal x "\\beginitems")
(setq beginEndStack (cons '|items| beginEndStack)))
(equal x "\\begin")
(cond
  ((and (consp u) (consp (qcdr u)) (equal (qcar (qcdr u)) |$charLbrace|)
    (consp (qcddr u)) (equal (car (qcddr u)) |$charRbrace|))
    (setq y (qcaddr u))
    (cond
      ((null (|member| y |$beginEndList|))
        (|checkDocError| (list "Unknown begin type: \\begin{" y "}"))))
      (setq beginEndStack (cons y beginEndStack))
      (setq u (qcddr u)))
    (t (|checkDocError| (list "Improper \\begin command")))))
(equal x "\\item")
(cond
  ((|member| (ifcar beginEndStack) '("items" "menu")) nil)
  (null beginEndStack)
  (|checkDocError| (list "\\item appears outside a \\begin-\\end")))
(t
  (|checkDocError|
    (list "\\item appears within a \\begin{"
      (ifcar beginEndStack) ".")
    )))
(equal x "\\end")
(cond
  ((and (consp u) (consp (qcdr u)) (equal (qcar (qcdr u)) |$charLbrace|)
    (consp (qcddr u)) (equal (car (qcddr u)) |$charRbrace|))
    (setq y (qcaddr u))
    (cond
      ((equal y (ifcar beginEndStack))
        (setq beginEndStack (cdr beginEndStack))
        (setq u (qcddr u)))
      (t
        (|checkDocError|
          (list "Trying to match \\begin{" (ifcar beginEndStack)
            "} with \\end{" y "}")))))
    (t
      (|checkDocError| (list "Improper \\end command")))))
(pop u))
(cond
  (beginEndStack
    (|checkDocError| (list "Missing \\end{" (car beginEndStack) "}")))
  (t '|ok|)))

```

9.1.19 defun checkSayBracket

— defun checkSayBracket —

```
(defun |checkSayBracket| (x)
  (cond
    ((or (char= x #\() (char= x #\))) "pren")
    ((or (char= x #\{) (char= x #\})) "brace")
    (t "bracket")))
```

9.1.20 defun checkArguments

```
[hget p??]
[checkHTargs p486]
[$htMacroTable p??]
```

— defun checkArguments —

```
(defun |checkArguments| (u)
  (let (x k)
    (declare (special |$htMacroTable|))
    (loop while u
      do (setq x (car u))
        (cond
          ((null (setq k (hget |$htMacroTable| x))) '|skip|)
          ((eq k 0) '|skip|)
          ((> k 0) (|checkHTargs| x (cdr u) k nil))
          (t (|checkHTargs| x (cdr u) (- k) t)))
        (pop u))
    u))
```

9.1.21 defun checkHTargs

Note that u should start with an open brace. [checkLookForLeftBrace p487]
 [checkLookForRightBrace p487]
 [checkDocError p478]
 [checkHTargs p486]

```
[ifcdr p??]
```

— **defun checkHTargs** —

```
(defun |checkHTargs| (keyword u nargs inteerValue?)
  (cond
    ((eq1 nargs 0) '|ok|)
    ((null (setq u (|checkLookForLeftBrace| u)))
     (|checkDocError| (list "Missing argument for " keyword)))
    ((null (setq u (|checkLookForRightBrace| (ifcdr u))))
     (|checkDocError| (list "Missing right brace for " keyword)))
    (t
     (|checkHTargs| keyword (cdr u) (1- nargs) inteerValue?))))
```

—————

9.1.22 defun checkLookForLeftBrace

```
[nequal p??]
[$charBlank p??]
[$charLbrace p??]
```

— **defun checkLookForLeftBrace** —

```
(defun |checkLookForLeftBrace| (u)
  (declare (special |$charBlank| |$charLbrace|))
  (loop while u
    do
      (cond
        ((equal (car u) |$charLbrace|) (return (car u)))
        ((nequal (car u) |$charBlank|) (return nil))
        (t (pop u))))
  u)
```

—————

9.1.23 defun checkLookForRightBrace

This returns a line beginning with right brace [\$charLbrace p??]
[\$charRbrace p??]

— **defun checkLookForRightBrace** —

```
(defun |checkLookForRightBrace| (u)
  (let (found count)
```

```
(declare (special |$charLbrace| |$charRbrace|))
(setq count 0)
(loop while u
  do
    (cond
      ((equal (car u) |$charRbrace|)
        (if (eql count 0)
          (return (setq found u))
          (setq count (1- count)))))
      ((equal (car u) |$charLbrace|)
        (setq count (1+ count)))))
    (pop u))
found))
```

9.1.24 defun checkTransformFirsts

```
[pname p??]
[leftTrim p??]
[fillerSpaces p??]
[checkTransformFirsts p488]
[maxindex p??]
[checkSkipToken p492]
[checkSkipBlanks p491]
[getMatchingRightPren p492]
[nequal p??]
[checkDocError p478]
[strconc p??]
[getl p??]
[lassoc p??]
[$checkPrenAlist p??]
[$charBack p??]
```

— defun checkTransformFirsts —

```
(defun |checkTransformFirsts| (opname u margin)
  (prog (namestring s m infixOp p open close z n i prefixOp j k firstWord)
    (declare (special |$checkPrenAlist| |$charBack|))
    (return
      (progn
        ; case 1: \spad{...
        ; case 2: form(args)
        (setq namestring (pname opname))
        (cond
          ((equal namestring "Zero") (setq namestring "0"))
```

```

(equal namestring "One") (setq namestring "1"))
(t nil))
(cond
  (> margin 0)
  (setq s (|leftTrim| u))
  (strconc (|fillerSpaces| margin) (|checkTransformFirst| opname s 0)))
(t
  (setq m (maxindex u))
  (cond
    (> 2 m) u)
  ((equal (elt u 0) |$charBack|) u)
  ((alpha-char-p (elt u 0))
   (setq i (or (|checkSkipToken| u 0 m) (return u)))
   (setq j (or (|checkSkipBlanks| u i m) (return u)))
   (setq open (elt u j))
   (cond
     ((or (and (equal open #\[) (setq close #\]))
          (and (equal open #\() (setq close #\))))
      (setq k (|getMatchingRightPren| u (1+ j) open close))
      (cond
        ((nequal namestring (setq firstWord (substring u 0 i)))
         (|checkDocError|
          (list "Improper first word in comments: " firstWord))
          u)
        (null k)
        (cond
          ((equal open (|char| '[]))
           (|checkDocError|
            (list "Missing close bracket on first line: " u)))
          (t
           (|checkDocError|
            (list "Missing close parenthesis on first line: " u))))
          u)
        (t
         (strconc "\\spad{" (substring u 0 (1+ k)) "}"
                  (substring u (1+ k) nil))))))
  (t
   (setq k (or (|checkSkipToken| u j m) (return u)))
   (setq infixOp (intern (substring u j (- k j))))
   (cond
     ; case 3: form arg
     ((null (get1 infixOp '|Led|))
      (cond
        ((nequal namestring (setq firstWord (substring u 0 i)))
         (|checkDocError|
          (list "Improper first word in comments: " firstWord))
          u)
        ((and (eql (|#| (setq p (pname infixOp))) 1)
         (setq open (elt p 0))
         (setq close (lassoc open |$checkPrenAlist|))))

```

```

      (setq z (|getMatchingRightPren| u (1+ k) open close))
      (when (> z (maxindex u)) (setq z (1- k)))
      (strconc "\\spad{" (substring u 0 (1+ z)) "}"
                (substring u (1+ z) nil)))
    (t
      (strconc "\\spad{" (substring u 0 k) "}"
                (substring u k nil))))))
  (t
    (setq z (or (|checkSkipBlanks| u k m) (return u)))
    (setq n (or (|checkSkipToken| u z m) (return u)))
    (cond
      ((nequal namestring (pname infixOp))
        (|checkDocError|
          (list "Improper initial operator in comments: " infixOp))
        u)
      (t
        (strconc "\\spad{" (substring u 0 n) "}"
                  (substring u n nil)))))))))

; case 4: arg op arg
  (t
    (setq i (or (|checkSkipToken| u 0 m) (return u)))
    (cond
      ((nequal namestring (setq firstWord (substring u 0 i)))
        (|checkDocError|
          (list "Improper first word in comments: " firstWord))
        u)
      (t
        (setq prefixOp (intern (substring u 0 i)))
        (cond
          ((null (get1 prefixOp '|Nud|)) u)
          (t
            (setq j (or (|checkSkipBlanks| u i m) (return u)))
            (cond
              ((equal (elt u j) (|char| '|(|))
                (setq j
                  (|getMatchingRightPren| u (1+ j) (|char| '|(|) (|char| '|)|)))
                (cond
                  ((> j m) u)
                  (t
                    (strconc "\\spad{" (substring u 0 (1+ j)) "}"
                              (substring u (1+ j) nil))))))
              (t
                (setq k (or (|checkSkipToken| u j m) (return u)))
                (cond
                  ((nequal namestring (setq firstWord (substring u 0 i)))
                    (|checkDocError|
                      (list "Improper first word in comments: " firstWord))
                    u)
                  (t
                    (t

```

```
(strconc "\\spad{" (substring u 0 k) "}"  
              (substring u k nil)))))))))))))
```

9.1.25 defun checkSkipBlanks

[\$charBlank p??]

— defun checkSkipBlanks —

```
(defun |checkSkipBlanks| (u i m)  
  (declare (special |$charBlank|))  
  (do ()  
    ((null (and (> m i) (equal (elt u i) |$charBlank|))) nil)  
    (setq i (1+ i)))  
  (unless (= i m) i))
```

9.1.26 defun checkSkipIdentifierToken

[checkAlphabetic p491]

— defun checkSkipIdentifierToken —

```
(defun |checkSkipIdentifierToken| (u i m)  
  (do ()  
    ((null (and (> m i) (|checkAlphabetic| (elt u i)))) nil)  
    (setq i (1+ i)))  
  (unless (= i m) i))
```

9.1.27 defun checkAlphabetic

[\$charIdentifierEndings p??]

— defun checkAlphabetic —

```
(defun |checkAlphabetic| (c)  
  (declare (special |$charIdentifierEndings|))  
  (or (alpha-char-p c) (digitp c) (member c |$charIdentifierEndings|)))
```

9.1.28 defun checkSkipToken

[checkSkipIdentifierToken p491]
 [checkSkipOpToken p492]

— defun checkSkipToken —

```
(defun |checkSkipToken| (u i m)
  (if (alpha-char-p (elt u i))
      (|checkSkipIdentifierToken| u i m)
      (|checkSkipOpToken| u i m)))
```

9.1.29 defun checkSkipOpToken

[checkAlphabetic p491]
 [member p??]
 [\$charDelimiters p??]

— defun checkSkipOpToken —

```
(defun |checkSkipOpToken| (u i m)
  (declare (special |$charDelimiters|))
  (do ()
      ((null (and (> m i)
                  (null (|checkAlphabetic| (elt u i)))
                  (null (|member| (elt u i) |$charDelimiters|)))))
      nil)
  (setq i (1+ i)))
(unless (= i m) i))
```

9.1.30 defun getMatchingRightPren

[maxindex p??]

— defun getMatchingRightPren —

```
(defun |getMatchingRightPren| (u j open close)
```

```
(let (m c found count)
  (setq count 0)
  (setq m (maxindex u))
  (loop for i from j to m
    do
      (setq c (elt u i))
      (cond
        ((equal c close)
         (if (eql count 0)
             (return (setq found i))
             (setq count (1- count)))))
        ((equal c open)
         (setq count (1+ count)))))
  found))
```

9.1.31 defun checkGetMargin

[firstNonBlankPosition p493]

— defun checkGetMargin —

```
(defun |checkGetMargin| (lines)
  (let (x k margin)
    (loop while lines
      do
        (setq x (car lines))
        (setq k (|firstNonBlankPosition| x))
        (unless (= k -1) (setq margin (if margin (min margin k) k)))
        (pop lines))
    (or margin 0)))
```

9.1.32 defun firstNonBlankPosition

[nequal p??]
[maxindex p??]

— defun firstNonBlankPosition —

```
(defun |firstNonBlankPosition| (&rest therest)
  (let ((x (car therest)) (options (cdr therest)) start k)
    (declare (special |$charBlank|))
```

```

(setq start (or (ifcar options) 0))
(setq k -1)
(loop for i from start to (maxindex x)
  do (when (nequal (elt x i) |$charBlank|) (return (setq k i))))
k))

```

9.1.33 defun checkIeEg

[checkIeEgfun p494]
 [nreverse p??]

— defun checkIeEg —

```

(defun |checkIeEg| (u)
  (let (x verbatim z acc)
    (setq acc nil)
    (setq verbatim nil)
    (loop while u
      do
        (setq x (car u))
        (setq acc
          (cond
            ((equal x "\\end{verbatim}")
             (setq verbatim nil)
             (cons x acc))
            (verbatim (cons x acc))
            ((equal x "\\begin{verbatim}")
             (setq verbatim t)
             (cons x acc))
            ((setq z (|checkIeEgfun| x))
             (append (nreverse z) acc))
            (t (cons x acc))))
        (setq u (cdr u)))
    (nreverse acc)))

```

9.1.34 defun checkIeEgfun

[maxindex p??]
 [checkIeEgFun p??]
 [\$charPeriod p??]

— defun checkIeEgfun —

```

(defun |checkIeEgfun| (x)
  (let (m key firstPart result)
    (declare (special |$charPeriod|))
    (cond
      ((characterp x) nil)
      ((equal x "") nil)
      (t
       (setq m (maxindex x))
       (loop for k from 0 to (- m 3)
         do
           (cond
             ((and
              (equal (elt x (1+ k)) |$charPeriod|)
              (equal (elt x (+ k 3)) |$charPeriod|)
              (or
               (and
                (equal (elt x k) #\i)
                (equal (elt x (+ k 2)) #\e)
                (setq key "that is"))
               (and
                (equal (elt x k) #\e)
                (equal (elt x (+ k 2)) #\g)
                (setq key "for example")))))
              (progn
               (setq firstPart (when (> k 0) (cons (substring x 0 k) nil)))
               (setq result
                (append firstPart
                 (cons "\\spadignore{"
                  (cons (substring x k 4)
                   (cons "}"
                    (|checkIeEgfun| (substring x (+ k 4) nil))))))))))
              result))))))

```

9.1.35 defun checkSplitBrace

```

[charp p??]
[length p??]
[checkSplitBackslash p496]
[checkSplitBrace p495]
[checkSplitOn p498]
[checkSplitPunctuation p497]

```

— defun checkSplitBrace —

```

(defun |checkSplitBrace| (x)

```

```

(let (m u)
  (cond
    ((charp x) (list x))
    ((eql (|#| x) 1) (list (elt x 0)))
    ((and (setq u (|checkSplitBackslash| x)) (cdr u))
     (let (result)
       (loop for y in u do (append result (|checkSplitBrace| y)))
       result))
    (t
     (setq m (maxindex x))
     (cond
       ((and (setq u (|checkSplitOn| x)) (cdr u))
        (let (result)
          (loop for y in u do (append result (|checkSplitBrace| y)))
          result))
       ((and (setq u (|checkSplitPunctuation| x)) (cdr u))
        (let (result)
          (loop for y in u do (append result (|checkSplitBrace| y)))
          result))
       (t (list x))))))

```

9.1.36 defun checkSplitBackslash

[checkSplitBackslash p496]
 [maxindex p??]
 [charPosition p??]
 [\$charBack p??]

— defun checkSplitBackslash —

```

(defun |checkSplitBackslash| (x)
  (let (m k u v)
    (declare (special |$charBack|))
    (cond
      ((null (stringp x)) (list x))
      (t
       (setq m (maxindex x))
       (cond
         ((> m (setq k (|charPosition| |$charBack| x 0)))
          (cond
            ((or (eql m 1) (alpha-char-p (elt x (1+ k)))) ;starts with backslash so
             (if (> m (setq k (|charPosition| |$charBack| x 1)))
                 ; yes, another backslash
                 (cons (substring x 0 k) (|checkSplitBackslash| (substring x k nil)))
                 ; no, just return the line

```

```

      (list x)))
    ((eq1 k 0)
     ; starts with backspace but x.1 is not a letter; break it up
     (cons (substring x 0 2)
           (|checkSplitBackslash| (substring x 2 nil))))
    (t
     (setq u (substring x 0 k))
     (setq v (substring x k 2))
     (if (= (1+ k) m)
         (list u v)
         (cons u
               (cons v
                     (|checkSplitBackslash|
                      (substring x (+ k 2) nil)))))))
    (t (list x))))))

```

9.1.37 defun checkSplitPunctuation

```

[charp p??]
[maxindex p??]
[checkSplitPunctuation p497]
[charPosition p??]
[hget p??]
[$charDash p??]
[$htMacroTable p??]
[$charQuote p??]
[$charPeriod p??]
[$charSemiColon p??]
[$charComma p??]
[$charBack p??]

```

— defun checkSplitPunctuation —

```

(defun |checkSplitPunctuation| (x)
  (let (m lastchar v k u)
    (declare (special |$charDash| |$htMacroTable| |$charBack| |$charQuote|
                      |$charComma| |$charSemiColon| |$charPeriod|))
    (cond
      ((charp x) (list x))
      (t
       (setq m (maxindex x))
       (cond
         ((> 1 m) (list x))
         (t
          (setq lastchar (elt x m))

```

```

(cond
  ((and (equal lastchar |$charPeriod|)
        (equal (elt x (1- m)) |$charPeriod|))
    (cond
      ((eql m 1) (list x))
      ((and (> m 3) (equal (elt x (- m 2)) |$charPeriod|))
        (append (|checkSplitPunctuation| (substring x 0 (- m 2)))
                  (list "..."))))
      (t
       (append (|checkSplitPunctuation| (substring x 0 (1- m)))
                 (list "..."))))
    ((or (equal lastchar |$charPeriod|)
          (equal lastchar |$charSemiColon|)
          (equal lastchar |$charComma|))
      (list (substring x 0 m) lastchar))
    ((and (> m 1) (equal (elt x (1- m)) |$charQuote|))
      (list (substring x 0 (1- m)) (substring x (1- m) nil)))
    (> m (setq k (|charPosition| |$charBack| x 0)))
    (cond
      ((eql k 0)
        (cond
          ((or (eql m 1) (hget |$htMacroTable| x) (alpha-char-p (elt x 1)))
            (list x))
          (t
           (setq v (substring x 2 nil))
           (cons (substring x 0 2) (|checkSplitPunctuation| v))))))
      (t
       (setq u (substring x 0 k))
       (setq v (substring x k nil))
       (append (|checkSplitPunctuation| u)
                (|checkSplitPunctuation| v))))))
    (> m (setq k (|charPosition| |$charDash| x 1)))
    (setq u (substring x (1+ k) nil))
    (cons (substring x 0 k)
          (cons |$charDash| (|checkSplitPunctuation| u))))
    (t
     (list x))))))

```

9.1.38 defun checkSplitOn

```

[checkSplitOn p498]
[charp p??]
[maxindex p??]
[charPosition p??]
[$charBack p??]

```

[*\$charSplitList p??*]

— **defun checkSplitOn** —

```
(defun |checkSplitOn| (x)
  (let (m char k z)
    (declare (special |$charBack| |$charSplitList|))
    (cond
      ((charp x) (list x))
      (t
       (setq z |$charSplitList|)
       (setq m (maxindex x))
       (loop while z
         do
           (setq char (car z))
           (cond
             ((and (eql m 0) (equal (elt x 0) char))
              (return (setq k -1)))
             (t
              (setq k (|charPosition| char x 0))
              (cond
                ((and (> k 0) (equal (elt x (1- k)) |$charBack|)) (list x))
                ((<= k m) (return k))))))
           (pop z))
      (cond
        ((null z) (list x))
        ((eql k -1) (list char))
        ((eql k 0) (list char (substring x 1 nil)))
        ((eql k (maxindex x)) (list (substring x 0 k) char))
        (t
         (cons (substring x 0 k)
                (cons char (|checkSplitOn| (substring x (1+ k) nil))))))))))
```

—————

9.1.39 defun checkNumOfArgs

A nil return implies that the argument list length does not match [*opOf p??*]

[*constructor? p??*]

[*abbreviation? p??*]

[*getdatabase p??*]

— **defun checkNumOfArgs** —

```
(defun |checkNumOfArgs| (conform)
  (let (conname)
    (setq conname (|opOf| conform))
```

```
(when (or (|constructor?| conname) (setq conname (|abbreviation?| conname)))
  (|#| (getdatabase conname 'constructorargs))))
```

9.1.40 defun checkRemoveComments

[checkTrimCommented p500]

— defun checkRemoveComments —

```
(defun |checkRemoveComments| (lines)
  (let (line acc)
    (loop while lines
      do
        (setq line (|checkTrimCommented| (car lines)))
        (when (>= (|firstNonBlankPosition| line) 0) (push line acc))
        (pop lines))
    (nreverse acc)))
```

9.1.41 defun checkTrimCommented

[length p??]
 [htcharPosition p501]
 [nequal p??]

— defun checkTrimCommented —

```
(defun |checkTrimCommented| (line)
  (let (n k)
    (setq n (|#| line))
    (setq k (|htcharPosition| (|char| '%) line 0))
    (cond
      ((eq k 0) "")
      ((or (>= k (1- n)) (nequal (elt line (1+ k)) #\%)) line)
      ((> (|#| line) k) (substring line 0 k))
      (t line))))
```

9.1.42 defun htcharPosition

```
[length p??]
[charPosition p??]
[nequal p??]
[htcharPosition p501]
[$charBack p??]
```

— **defun htcharPosition** —

```
(defun |htcharPosition| (char line i)
  (let (m k)
    (declare (special |$charBack|))
    (setq m (|#| line))
    (setq k (|charPosition| char line i))
    (cond
      ((eql k m) k)
      (> k 0)
      (if (nequal (elt line (1- k)) |$charBack|)
          k
          (|htcharPosition| char line (1+ k))))
      (t 0))))
```

—————

9.1.43 defun checkAddMacros

```
[lassoc p??]
[nreverse p??]
[$HTmacs p??]
```

— **defun checkAddMacros** —

```
(defun |checkAddMacros| (u)
  (let (x verbatim y acc)
    (declare (special |$HTmacs|))
    (loop while u
      do
        (setq x (car u))
        (setq acc
          (cond
            ((string= x "\\end{verbatim}")
             (setq verbatim nil)
             (cons x acc))
            (verbatim
             (cons x acc))
            ((string= x "\\begin{verbatim}")
```

```

      (setq verbatim t)
      (cons x acc))
    ((setq y (lassoc x |$HTmacs|))
     (append y acc))
    (t (cons x acc))))
  (pop u)
  (nreverse acc)))

```

9.1.44 defun checkIndentedLines

[firstNonBlankPosition p493]
 [strconc p??]
 [\$charFauxNewline p??]

— defun checkIndentedLines —

```

(defun |checkIndentedLines| (u margin)
  (let (k s verbatim u2)
    (declare (special |$charFauxNewline|))
    (loop for x in u
      do
        (setq k (|firstNonBlankPosition| x))
        (cond
          ((eq1 k -1)
           (if verbatim
              (setq u2 (append u2 (list |$charFauxNewline|)))
              (setq u2 (append u2 (list "\\blankline "))))))
          (t
           (setq s (substring x k nil))
           (cond
             ((string= s "\\begin{verbatim}")
              (setq verbatim t)
              (setq u2 (append u2 (list s))))
             ((string= s "\\end{verbatim}")
              (setq verbatim nil)
              (setq u2 (append u2 (list s))))
             (verbatim
              (setq u2 (append u2 (list (substring x margin nil))))))
             ((eq1 margin k)
              (setq u2 (append u2 (list s))))
             (t
              (setq u2
                (append u2
                  (list (strconc "\\indented{" (stringimage (- k margin))
                               "}" (|checkAddSpaceSegments| s 0) "}")
                      ))))))))

```

u2))

9.1.45 defun newString2Words

[newWordFrom p503]
[nreverse0 p??]

— defun newString2Words —

```
(defun |newString2Words| (z)
  (let (m tmp1 w i result)
    (cond
      ((null (stringp z)) (list z))
      (t
       (setq m (maxindex z))
       (cond
         ((eq1 m -1) nil)
         (t
          (setq i 0)
          (do () ; [w while newWordFrom(1,i,m) is [w,i]]
              ((null (progn
                        (setq tmp1 (|newWordFrom| z i m))
                        (and (consp tmp1)
                            (progn
                             (setq w (qcar tmp1))
                             (and (consp (qcdr tmp1))
                                 (eq (qcddr tmp1) nil)
                                 (progn
                                  (setq i (qcadr tmp1))
                                  t))))))
              (nreverse0 result))
          (setq result (cons (qcar tmp1) result))))))))))
```

9.1.46 defun newWordFrom

[\$stringFauxNewline p??]
[\$charBlank p??]
[\$charFauxNewline p??]

— defun newWordFrom —

```

(defun |newWordFrom| (z i m)
  (let (ch done buf)
    (declare (special |$charFauxNewline| |$charBlank| |$stringFauxNewline|))
    (loop while (and (<= i m) (char= (elt z i) #\space)) do (incf i))
    (cond
      ((> i m) nil)
      (t
       (setq buf "")
       (setq ch (elt z i))
       (cond
         ((equal ch |$charFauxNewline|)
          (list |$stringFauxNewline| (1+ i)))
         (t
          (setq done nil)
          (loop while (and (<= i m) (null done))
            do
              (setq ch (elt z i))
              (cond
                ((or (equal ch |$charBlank|) (equal ch |$charFauxNewline|))
                 (setq done t))
                (t
                 (setq buf (strconc buf ch))
                 (setq i (1+ i))))))
          (list buf i)))))))

```

9.1.47 defun checkGetArgs

```

[maxindex p??]
[firstNonBlankPosition p493]
[checkGetArgs p504]
[stringPrefix? p??]
[getMatchingRightPren p492]
[charPosition p??]
[nequal p??]
[trimString p??]
[$charComma p??]

```

— defun checkGetArgs —

```

(defun |checkGetArgs| (u)
  (let (m k acc i)
    (declare (special |$charComma|))
    (cond
      ((null (stringp u)) nil)
      (t

```

```

(setq m (maxindex u))
(setq k (|firstNonBlankPosition| u))
(cond
  ((> k 0)
   (|checkGetArgs| (substring u k nil)))
  ((|stringPrefix?| "\\spad{" u)
   (setq k (or (|getMatchingRightPren| u 6 #{ #\}) m))
   (|checkGetArgs| (substring u 6 (- k 6))))
  ((> (setq i (|charPosition| #\ ( u 0)) m)
   nil)
   ((nequal (elt u m) #\))
   nil)
  (t
   (do ()
     ((null (> m (setq k (|charPosition| |$charComma| u (1+ i))))) nil)
     (setq acc
      (cons (|trimString| (substring u (1+ i) (1- (- k i)))) acc))
     (setq i k))
    (nreverse (cons (substring u (1+ i) (1- (- m i))) acc))))))

```

9.1.48 defun checkAddSpaceSegments

[checkAddSpaceSegments p505]
 [maxindex p??]
 [charPosition p??]
 [strconc p??]
 [\$charBlank p??]

— defun checkAddSpaceSegments —

```

(defun |checkAddSpaceSegments| (u k)
  (let (m i j n)
    (declare (special |$charBlank|))
    (setq m (maxindex u))
    (setq i (|charPosition| |$charBlank| u k))
    (cond
      ((> i m) u)
      (t
       (setq j i)
       (loop while (and (incf j) (char= (elt u j) #\space)))
       (setq n (- j i)) ; number of blanks
       (if (> n 1)
         (strconc (substring u 0 i) "\\space{" (stringimage n) "}"
                   (|checkAddSpaceSegments| (substring u (+ i n) nil) 0))
         (|checkAddSpaceSegments| u j)))))

```

9.1.49 defun checkTrim

```
[charPosition p??]
[nequal p??]
[systemError p??]
[checkDocError p478]
[$charBlank p??]
[$x p??]
[$charPlus p??]
```

— defun checkTrim —

```
(defun |checkTrim| (|$x| lines)
  (declare (special |$x|))
  (labels (
    (trim (s)
      (let (k)
        (declare (special |$charBlank|))
        (setq k (wherePP s))
        (substring s (+ k 2) nil)))
    (wherePP (u)
      (let (k)
        (declare (special |$charPlus|))
        (setq k (|charPosition| |$charPlus| u 0))
        (if (or (eql k (|#| u))
              (nequal (|charPosition| |$charPlus| u (1+ k)) (1+ k)))
            (|systemError| " Improper comment found"
              k))))
    (let (j s)
      (setq s (list (wherePP (car lines)))))
      (loop for x in (rest lines)
        do
          (setq j (wherePP x))
          (unless (member j s)
            (|checkDocError| (list |$x| " has varying indentation levels"))
            (setq s (cons j s))))
      (loop for y in lines
        collect (trim y))))))
```

9.1.50 defun checkExtract

[firstNonBlankPosition p493]
 [substring? p??]
 [charPosition p??]
 [length p??]

— defun checkExtract —

```
(defun |checkExtract| (header lines)
  (let (line u margin firstLines m k j i acc)
    (loop while lines
      do
        (setq line (car lines))
        (setq k (|firstNonBlankPosition| line)) ; gives margin of description
        (if (|substring?| header line k)
          (return nil)
          (setq lines (cdr lines))))
    (cond
      ((null lines) nil)
      (t
       (setq u (car lines))
       (setq j (|charPosition| #\: u k))
       (setq margin k)
       (setq firstLines
        (if (nequal (setq k (|firstNonBlankPosition| u (1+ j))) -1)
          (cons (substring u (1+ j) nil) (cdr lines))
          (cdr lines)))
        ; now look for another header; if found skip all rest of these lines
        (setq acc nil)
        (loop for line in firstLines
          do
            (setq m (|#| line))
            (cond
              ((eq1 (setq k (|firstNonBlankPosition| line)) -1) '|skip|)
              ((> k margin) '|skip|)
              ((null (upper-case-p (elt line k))) '|skip|)
              ((equal (setq j (|charPosition| #\: line k)) m) '|skip|)
              ((> j (setq i (|charPosition| #\space line (1+ k)))) '|skip|)
              (t (return nil)))
            (setq acc (cons line acc)))
        (nreverse acc))))))
```

9.1.51 defun checkFixCommonProblem

```
[member p??]
[ifcar p??]
[ifcdr p??]
[nequal p??]
[checkDocError p478]
[$charLbrace p??]
[$HTspadmacros p??]
```

— **defun checkFixCommonProblem** —

```
(defun |checkFixCommonProblem| (u)
  (let (x next acc)
    (declare (special |$charLbrace| |$HTspadmacros|))
    (loop while u
      do
        (setq x (car u))
        (cond
          ((and (equal x |$charLbrace|)
                (|member| (setq next (ifcar (cdr u))) |$HTspadmacros|)
                (nequal (ifcar (ifcdr (cdr u))) |$charLbrace|))
            (|checkDocError| (list "Reversing " next " and left brace")))
          (setq acc (cons |$charLbrace| (cons next |acc|)))
          (setq u (cddr u)))
        (t
          (setq acc (cons x acc))
          (setq u (cdr u))))))
  (nreverse acc)))
```

—————

9.1.52 defun checkDecorate

```
[checkDocError p478]
[member p??]
[checkAddBackSlashes p511]
[hasNoVowels p511]
[$checkingXmptex? p??]
[$charExclusions p??]
[$argl p??]
[$charBack p??]
[$charRbrace p??]
[$charLbrace p??]
```

— **defun checkDecorate** —

```

(defun |checkDecorate| (u)
  (let (x count mathSymbolsOk spadflag verbatim v xcount acc)
    (declare (special |$charLbrace| |$charRbrace| |$charBack| |$arg1|
                     |$charExclusions| |$checkingXmptex?|))
    (setq count 0)
    (loop while u
      do
        (setq x (car u))
        (cond
          ((null verbatim)
            (cond
              ((string= x "\\em")
                (cond
                  (> count 0)
                  (setq mathSymbolsOk (1- count))
                  (setq spadflag (1- count)))
                (t
                 (|checkDocError| (list "\\em must be enclosed in braces")))))
            (when (|member| x '("\\spadpaste" "\\spad" "\\spadop"))
              (setq mathSymbolsOk count))
            (cond
              ((|member| x '("\\s" "\\spadtype" "\\spadsys" "\\example" "\\andexample"
                           "\\spadop" "\\spad" "\\spadignore" "\\spadpaste"
                           "\\spadcommand" "\\footnote"))
                (setq spadflag count))
              ((equal x |$charLbrace|)
                (setq count (1+ count)))
              ((equal x |$charRbrace|)
                (setq count (1- count))
                (when (eql mathSymbolsOk count) (setq mathSymbolsOk nil))
                (when (eql spadflag count) (setq spadflag nil)))
              ((and (null |mathSymbolsOk|)
                    (|member| x '("+ " * " = " == " ->"))
                    (when |$checkingXmptex?|
                      (|checkDocError|
                       (list '|Symbol| | x " appearing outside \\spad{}"))))))
            (setq acc
              (cond
                ((string= x "\\end{verbatim}")
                  (setq verbatim nil)
                  (cons x acc))
                (verbatim (cons x acc))
                ((string= x "\\begin{verbatim}")
                  (setq verbatim t)
                  (cons x acc))
                ((and (string= x "\\begin")
                      (equal (car (setq v (ifcdr u))) |$charLbrace|)
                      (string= (car (setq v (ifcdr v))) "detail")
                      (equal (car (setq v (ifcdr v))) |$charRbrace|))
                  (setq u v)

```

```

(cons "\\blankline " acc))
((and (string= x "\\end")
      (equal (car (setq v (ifcdr u))) |$charLbrace|)
      (string= (car (setq v (ifcdr v))) "detail")
      (equal (car (setq v (ifcdr v))) |$charRbrace|))
 (setq u v)
 acc)
((or (char= x #\$) (string= x "$"))
 (cons "\\$" acc))
((or (char= x #\%) (string= x "%"))
 (cons "\\%" acc))
((or (char= x #\,) (string= x ","))
 (cons ",{" acc))
((string= x "\\spad")
 (cons "\\spad" acc))
((and (stringp x) (digitp (elt x 0)))
 (cons x acc))
((and (null spadflag)
      (or (and (charp x)
                (alpha-char-p x)
                (null (member x |$charExclusions|)))
          (|member| x |$argl|))))
 (cons |$charRbrace| (cons x (cons |$charLbrace| (cons "\\spad" acc)))))
((and (null spadflag)
      (or (and (stringp x)
                (null (equal (elt x 0) |$charBack|))
                (digitp (elt x (maxindex x))))
          (|member| x '("true" "false"))))
 (cons |$charRbrace| (cons x (cons |$charLbrace| (cons "\\spad" acc)))))
(t
 (setq xcount (|#| x))
 (cond
  ((and (eql xcount 3)
        (char= (elt x 1) #\t)
        (char= (elt x 2) #\h))
   (cons "th" (cons |$charRbrace|
                    (cons (elt x 0) (cons |$charLbrace| (cons "\\spad" acc))))))
  ((and (eql xcount 4)
        (char= (elt x 1) #\-)
        (char= (elt x 2) #\t)
        (char= (elt x 3) #\h))
   (cons "-th" (cons |$charRbrace|
                    (cons (elt x 0) (cons |$charLbrace| (cons "\\spad" acc))))))
  ((or (and (eql xcount 2)
            (char= (elt x 1) #\i))
       (and (null spadflag)
            (> xcount 0)
            (> 4 xcount)
            (null (|member| x '("th" "rd" "st")))
            (|hasNoVowels| x)))
   )

```

```

      (cons |$charRbrace|
        (cons x (cons |$charLbrace| (cons "\\spad" acc))))))
    (t
      (cons (|checkAddBackSlashes| x) acc))))))
  (setq u (cdr u)))
(nreverse acc)))

```

9.1.53 defun hasNoVowels

[maxindex p??]

— defun hasNoVowels —

```

(defun |hasNoVowels| (x)
  (labels (
    (isVowel (c)
      (or (eq c #\a) (eq c #\e) (eq c #\i) (eq c #\o) (eq c #\u)
          (eq c #\A) (eq c #\E) (eq c #\I) (eq c #\O) (eq c #\U))))
    (let (max)
      (setq max (maxindex x))
      (cond
        ((char= (elt x max) #\y) nil)
        (t
         (let ((result t))
           (loop for i from 0 to max
                 do (setq result (and result (null (isVowel (elt x i))))))
           result))))))

```

9.1.54 defun checkAddBackSlashes

[streconc p??]

[maxindex p??]

[checkAddBackSlashes p511]

[\$charBack p??]

[\$charEscapeList p??]

— defun checkAddBackSlashes —

```

(defun |checkAddBackSlashes| (s)
  (let (c m char insertIndex k)
    (declare (special |$charBack| |$charEscapeList|)))

```

```

(cond
  ((or (and (charp s) (setq c s))
        (and (eq1 (|#| s) 1) (setq c (elt s 0))))
    (if (member s |$charEscapeList|)
        (strconc |$charBack| c)
        s))
  (t
   (setq k 0)
   (setq m (maxindex s))
   (setq insertIndex nil)
   (loop while (< k m)
     do
       (setq char (elt s k))
       (cond
        ((char= char |$charBack|) (setq k (+ k 2)))
        ((member char |$charEscapeList|) (return (setq insertIndex k))))
       (setq k (1+ k)))
   (cond
    (insertIndex
     (|checkAddBackSlashes|
      (strconc (substring s 0 insertIndex) |$charBack| (elt s k)
                (substring s (1+ insertIndex) nil))))
    (T s))))))

```

—

9.1.55 defun checkAddSpaces

```

[$charBlank p??]
[$charFauxNewline p??]

```

— defun checkAddSpaces —

```

(defun |checkAddSpaces| (u)
  (let (u2 space)
    (declare (special |$charBlank| |$charFauxNewline|))
    (cond
     ((null u) nil)
     ((null (cdr u)) u)
     (t
      (setq space |$charBlank|)
      (setq i 0)
      (loop for f in u
        do
          (incf i)
          (when (string= f "\\begin{verbatim}")
            (setq space |$charFauxNewline|)

```

```
(unless u2 (setq u2 (list space)))  
(if (> i 1)  
    (setq u2 (append u2 (list space f)))  
    (setq u2 (append u2 (list f))))  
(when (string= f "\\end{verbatim}")  
    (setq u2 (append u2 (list space)))  
    (setq space |$charBlank|))  
u2)))
```

Chapter 10

Utility Functions

10.0.56 defun translabel

[translabel1 p515]

— defun translabel —

```
(defun translabel (x al)
  (translabel1 x al) x)
```

—————

10.0.57 defun translabel1

[refvecp p??]
[maxindex p??]
[translabel1 p515]
[lassoc p??]

— defun translabel1 —

```
(defun translabel1 (x al)
  "Transforms X according to AL = ((<label> . Sexpr) ..)."
  (cond
    ((refvecp x)
     (do ((i 0 (1+ i)) (k (maxindex x)))
         ((> i k)
          (if (let ((y (lassoc (elt x i) al))) (setelt x i y))
              (translabel1 (elt x i) al))))
     ((atom x) nil)
    ((let ((y (lassoc (first x) al)))
```

```
(if y (setf (first x) y) (translabel1 (cdr x) al)))
((translabel1 (first x) al) (translabel1 (cdr x) al)))
```

10.0.58 defun displayPreCompilationErrors

```
[length p??]
[remdup p??]
[sayBrightly p??]
[nequal p??]
[sayMath p??]
[$postStack p??]
[$topOp p??]
[$InteractiveMode p??]
```

— defun displayPreCompilationErrors —

```
(defun |displayPreCompilationErrors| ()
  (let (n errors heading)
    (declare (special |$postStack| |$topOp| |$InteractiveMode|))
    (setq n (|#| (setq |$postStack| (remdup (nreverse |$postStack|)))))
    (unless (eql n 0)
      (setq errors (cond ((> n 1) "errors") (t "error")))
      (cond
        (|$InteractiveMode|
         (|sayBrightly| (list " Semantic " errors " detected: ")))
        (t
         (setq heading
           (if (nequal |$topOp| '|$topOp|)
               (list " " |$topOp| " has")
               (list " You have")))
         (|sayBrightly|
          (append heading (list n "precompilation " errors ":" )))))
      (cond
        ((> n 1)
         (let ((i 1))
           (dolist (x |$postStack|)
             (|sayMath| (cons " " (cons i (cons " " x))))))
         (t (|sayMath| (cons " " (car |$postStack|))))
        (terpri))))
```

10.0.59 defun bumperrorcount

[\$InteractiveMode p??]
[\$spad-errors p??]

— defun bumperrorcount —

```
(defun bumperrorcount (kind)
  (declare (special |$InteractiveMode| $spad_errors))
  (unless |$InteractiveMode|
    (let ((index (case kind
                    (|syntax| 0)
                    (|precompilation| 1)
                    (|semantic| 2)
                    (t (error (break "BUMPERERRORCOUNT: kind=~s~%" kind))))))
      (setelt $spad_errors index (1+ (elt $spad_errors index))))))
```

10.0.60 defun parseTranCheckForRecord

[qcar p??]
[qcdr p??]
[postError p364]
[parseTran p93]

— defun parseTranCheckForRecord —

```
(defun |parseTranCheckForRecord| (x op)
  (declare (ignore op))
  (let (tmp3)
    (setq x (|parseTran| x))
    (cond
      ((and (consp x) (eq (qfirst x) '|Record|))
        (cond
          ((do ((z nil tmp3) (tmp4 (qrest x) (cdr tmp4)) (y nil))
                ((or z (atom tmp4)) tmp3)
                (setq y (car tmp4))
                (cond
                  ((null (and (consp y) (eq (qfirst y) '|:|) (consp (qrest y))
                               (consp (qcddr y)) (eq (qcdddr y) nil)))
                    (setq tmp3 (or tmp3 y))))
              (|postError| (list " Constructor" x "has missing label" )))
            (t x))))
      (t x))))
```

10.0.61 defun new2OldLisp

[new2OldTran p??]
 [postTransform p359]

— defun new2OldLisp —

```
(defun |new2OldLisp| (x)
  (|new2OldTran| (|postTransform| x)))
```

—————

10.0.62 defun makeSimplePredicateOrNil

[isSimple p??]
 [isAlmostSimple p??]
 [wrapSEQExit p??]

— defun makeSimplePredicateOrNil —

```
(defun |makeSimplePredicateOrNil| (p)
  (let (u g)
    (cond
      ((|isSimple| p) nil)
      ((setq u (|isAlmostSimple| p)) u)
      (t (|wrapSEQExit| (list (list 'let (setq g (gensym)) p) g))))))
```

—————

10.0.63 defun parse-spadstring

[match-current-token p453]
 [token-symbol p??]
 [push-reduction p462]
 [advance-token p456]

— defun parse-spadstring —

```
(defun parse-spadstring ()
  (let* ((tok (match-current-token 'spadstring))
        (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'spadstring-token (copy-tree symbol))
      (advance-token))))
```

t)))

10.0.64 defun parse-string

[match-current-token p453]
[token-symbol p??]
[push-reduction p462]
[advance-token p456]

— defun parse-string —

```
(defun parse-string ()
  (let* ((tok (match-current-token 'string))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'string-token (copy-tree symbol))
      (advance-token)
      t)))
```

10.0.65 defun parse-identifier

[match-current-token p453]
[token-symbol p??]
[push-reduction p462]
[advance-token p456]

— defun parse-identifier —

```
(defun parse-identifier ()
  (let* ((tok (match-current-token 'identifier))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'identifier-token (copy-tree symbol))
      (advance-token)
      t)))
```

10.0.66 defun parse-number

[match-current-token p453]
 [token-symbol p??]
 [push-reduction p462]
 [advance-token p456]

— **defun parse-number** —

```
(defun parse-number ()
  (let* ((tok (match-current-token 'number))
        (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'number-token (copy-tree symbol))
      (advance-token)
      t)))
```

10.0.67 defun parse-keyword

[match-current-token p453]
 [token-symbol p??]
 [push-reduction p462]
 [advance-token p456]

— **defun parse-keyword** —

```
(defun parse-keyword ()
  (let* ((tok (match-current-token 'keyword))
        (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'keyword-token (copy-tree symbol))
      (advance-token)
      t)))
```

10.0.68 defun parse-argument-designator

[push-reduction p462]
 [match-current-token p453]
 [token-symbol p??]
 [advance-token p456]

— defun parse-argument-designator —

```
(defun parse-argument-designator ()
  (let* ((tok (match-current-token 'argument-designator))
         (symbol (if tok (token-symbol tok))))
    (when tok
      (push-reduction 'argument-designator-token (copy-tree symbol))
      (advance-token)
      t)))
```

10.0.69 defun print-package

[out-stream p??]

— defun print-package —

```
(defun print-package (package)
  (declare (special out-stream))
  (format out-stream "&~%(IN-PACKAGE ~S )~%" package))
```

10.0.70 defun checkWarning

[postError p364]
[concat p??]

— defun checkWarning —

```
(defun |checkWarning| (msg)
  (|postError| (|concat| "Parsing error: " msg)))
```

10.0.71 defun tuple2List

[tuple2List p521]
[postTranSegment p378]
[postTran p360]

```
[$boot p??]
[$InteractiveMode p??]
```

— **defun tuple2List** —

```
(defun |tuple2List| (arg)
  (let (u p q)
    (declare (special |$InteractiveMode| $boot))
    (when (consp arg)
      (setq u (|tuple2List| (qrest arg)))
      (cond
        ((and (consp (qfirst arg)) (eq (qcaar arg) 'segment)
              (consp (qcddar arg))
              (consp (qcddar arg))
              (eq (qcdddar arg) nil))
         (setq p (qcadar arg))
         (setq q (qcaddar arg))
         (cond
           ((null u) (list '|construct| (|postTranSegment| p q)))
           ((and |$InteractiveMode| (null $boot))
            (cons '|append|
                  (cons (list '|construct| (|postTranSegment| p q))
                        (list (|tuple2List| (qrest arg))))))
           (t
            (cons '|nconc|
                  (cons (list '|construct| (|postTranSegment| p q))
                        (list (|tuple2List| (qrest arg)))))))
         ((null u) (list '|construct| (|postTran| (qfirst arg))))
         (t (list '|cons| (|postTran| (qfirst arg)) (|tuple2List| (qrest arg)))))))
```

10.0.72 defmacro pop-stack-1

```
[reduction-value p??]
[Pop-Reduction p524]
```

— **defmacro pop-stack-1** —

```
(defmacro pop-stack-1 () '(reduction-value (Pop-Reduction)))
```

10.0.73 defmacro pop-stack-2

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p524]

— defmacro pop-stack-2 —

```
(defmacro pop-stack-2 ()
  '(let* ((top (Pop-Reduction)) (next (Pop-Reduction)))
    (stack-push top Reduce-Stack)
    (reduction-value next)))
```

—————

10.0.74 defmacro pop-stack-3

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p524]

— defmacro pop-stack-3 —

```
(defmacro pop-stack-3 ()
  '(let* ((top (Pop-Reduction)) (next (Pop-Reduction)) (nnext (Pop-Reduction)))
    (stack-push next Reduce-Stack)
    (stack-push top Reduce-Stack)
    (reduction-value nnext)))
```

—————

10.0.75 defmacro pop-stack-4

[stack-push p89]
 [reduction-value p??]
 [Pop-Reduction p524]

— defmacro pop-stack-4 —

```
(defmacro pop-stack-4 ()
  '(let* ((top (Pop-Reduction))
    (next (Pop-Reduction))
    (nnext (Pop-Reduction))
    (nnnext (Pop-Reduction)))
```

```

(stack-push nnext Reduce-Stack)
(stack-push next Reduce-Stack)
(stack-push top Reduce-Stack)
(reduction-value nnext)))

```

10.0.76 defmacro nth-stack

```

[stack-store p??]
[reduction-value p??]

```

— defmacro nth-stack —

```

(defmacro nth-stack (x)
  '(reduction-value (nth (1- ,x) (stack-store Reduce-Stack))))

```

10.0.77 defun Pop-Reduction

```

[stack-pop p90]

```

— defun Pop-Reduction —

```

(defun Pop-Reduction () (stack-pop Reduce-Stack))

```

10.0.78 defun addclose

```

[suffix p??]

```

— defun addclose —

```

(defun addclose (line char)
  (cond
    ((char= (char line (maxindex line)) #\; )
     (setelt line (maxindex line) char)
     (if (char= char #\;) line (suffix #\; line)))
    ((suffix char line))))

```

10.0.79 defun blankp

— defun blankp —

```
(defun blankp (char)
  (or (eq char #\Space) (eq char #\tab)))
```

—————

10.0.80 defun drop

Return a pointer to the Nth cons of X, counting 0 as the first cons. [drop p525]

[take p??]

[croak p??]

— defun drop —

```
(defun drop (n x &aux m)
  (cond
    ((eq1 n 0) x)
    ((> n 0) (drop (1- n) (cdr x)))
    ((>= (setq m (+ (length x) n)) 0) (take m x))
    ((croak (list "Bad args to DROP" n x)))))
```

—————

10.0.81 defun escaped

— defun escaped —

```
(defun escaped (str n)
  (and (> n 0) (eq (char str (1- n)) #\_)))
```

—————

10.0.82 defvar \$comblocklist

— initvars —

```
(defvar $comblocklist nil "a dynamic lists of comments for this block")
```

10.0.83 defun fincomblock

- NUM is the line number of the current line
- OLDNUMS is the list of line numbers of previous lines
- OLDLOCS is the list of previous indentation locations
- NCBLOCK is the current comment block

```
[prepare-echo p88]
[$comblocklist p525]
[$EchoLineStack p??]
```

— defun fincomblock —

```
(defun fincomblock (num oldnums oldlocs ncblock linelist)
  (declare (special $EchoLineStack $comblocklist))
  (push
   (cond
    ((eql (car ncblock) 0) (cons (1- num) (reverse (cdr ncblock))))
    ;; comment for constructor itself paired with 1st line -1
    (t
     (when $EchoLineStack
      (setq num (pop $EchoLineStack))
      (prepare-echo linelist)
      (setq $EchoLineStack (list num)))
     (cons
      ;; scan backwards for line to left of current
      (do ((onums oldnums (cdr onums))
          (olocs oldlocs (cdr olocs))
          (sloc (car ncblock)))
          ((null onums) nil)
          (when (and (numberp (car olocs)) (<= (car olocs) sloc))
            (return (car onums))))
      (reverse (cdr ncblock))))
    $comblocklist))
```

10.0.84 defun indent-pos

— defun indent-pos —

```
(defun indent-pos (str)
  (do ((i 0 (1+ i)) (pos 0))
      ((>= i (length str)) nil)
    (case (char str i)
      (#\space (incf pos))
      (#\tab (setq pos (next-tab-loc pos)))
      (otherwise (return pos)))))
```

10.0.85 defun infixtok

[string2id-n p??]

— defun infixtok —

```
(defun infixtok (s)
  (member (string2id-n s 1) '(|then| |else|) :test #'eq))
```

10.0.86 defun is-console

[fp-output-stream p??]
 [*terminal-io* p??]

— defun is-console —

```
(defun is-console (stream)
  (and (streamp stream) (output-stream-p stream)
    (eq (system:fp-output-stream stream)
        (system:fp-output-stream *terminal-io*))))
```

10.0.87 defun next-tab-loc

— defun next-tab-loc —

```
(defun next-tab-loc (i)
  (* (1+ (truncate i 8)) 8))
```

10.0.88 defun nonblankloc

[blankp p525]

— **defun nonblankloc** —

```
(defun nonblankloc (str)
  (position-if-not #'blankp str))
```

—

10.0.89 defun parseprint— **defun parseprint** —

```
(defun parseprint (l)
  (when l
    (format t "~&~%          ***      PREPARSE          ***~%~%"
      (dolist (x l) (format t "~5d. ~a~%" (car x) (cdr x)))
      (format t "%"))))
```

—

10.0.90 defun skip-to-endif

[initial-substring p607]
 [preparseReadLine p85]
 [preparseReadLine1 p87]
 [skip-to-endif p528]

— **defun skip-to-endif** —

```
(defun skip-to-endif (x)
  (let (line ind tmp1)
    (setq tmp1 (preparseReadLine1))
    (setq ind (car tmp1))
    (setq line (cdr tmp1))
    (cond
     ((not (stringp line)) (cons ind line))
     ((initial-substring line ")endif") (preparseReadLine x))
     ((initial-substring line ")fin") (cons ind nil))
     (t (skip-to-endif x)))))
```

—

Chapter 11

The Compiler

11.1 Compiling EQ.spad

Given the top level command:

```
)co EQ
```

The default call chain looks like:

```
1> (|compiler| ...)
2> (|compileSpad2Cmd| ...)
   Compiling AXIOM source code from file /tmp/A.spad using old system
   compiler.
3> (|compilerDoit| ...)
4> (|/RQ,LIB|)
5> (/RF-1 ...)
6> (SPAD ...)
AXSERV abbreviates package AxiomServer
7> (S-PROCESS ...)
8> (|compTopLevel| ...)
9> (|comp0rCroak| ...)
10> (|comp0rCroak1| ...)
11> (|comp| ...)
12> (|compNoStacking| ...)
13> (|comp2| ...)
14> (|comp3| ...)
15> (|compExpression| ...)
* 16> (|compWhere| ...)
   17> (|comp| ...)
   18> (|compNoStacking| ...)
   19> (|comp2| ...)
   20> (|comp3| ...)
   21> (|compExpression| ...)
```

```

22> (|compSeq| ...)
23> (|compSeq1| ...)
24> (|compSeqItem| ...)
25> (|comp| ...)
26> (|compNoStacking| ...)
27> (|comp2| ...)
28> (|comp3| ...)
29> (|compExpression| ...)
<29 (|compExpression| ...)
<28 (|comp3| ...)
<27 (|comp2| ...)
<26 (|compNoStacking| ...)
<25 (|comp| ...)
<24 (|compSeqItem| ...)
24> (|compSeqItem| ...)
25> (|comp| ...)
26> (|compNoStacking| ...)
27> (|comp2| ...)
28> (|comp3| ...)
29> (|compExpression| ...)
30> (|compExit| ...)
31> (|comp| ...)
32> (|compNoStacking| ...)
33> (|comp2| ...)
34> (|comp3| ...)
35> (|compExpression| ...)
<35 (|compExpression| ...)
<34 (|comp3| ...)
<33 (|comp2| ...)
<32 (|compNoStacking| ...)
<31 (|comp| ...)
31> (|modifyModeStack| ...)
<31 (|modifyModeStack| ...)
<30 (|compExit| ...)
<29 (|compExpression| ...)
<28 (|comp3| ...)
<27 (|comp2| ...)
<26 (|compNoStacking| ...)
<25 (|comp| ...)
<24 (|compSeqItem| ...)
24> (|replaceExitEtc| ...)
25> (|replaceExitEtc,fn| ...)
26> (|replaceExitEtc| ...)
27> (|replaceExitEtc,fn| ...)
28> (|replaceExitEtc| ...)
29> (|replaceExitEtc,fn| ...)
<29 (|replaceExitEtc,fn| ...)
<28 (|replaceExitEtc| ...)
28> (|replaceExitEtc| ...)
29> (|replaceExitEtc,fn| ...)

```

```

    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    <27 (|replaceExitEtc,fn| ...)
    <26 (|replaceExitEtc| ...)
    26> (|replaceExitEtc| ...)
    27> (|replaceExitEtc,fn| ...)
    28> (|replaceExitEtc| ...)
    29> (|replaceExitEtc,fn| ...)
    30> (|replaceExitEtc| ...)
    31> (|replaceExitEtc,fn| ...)
    32> (|replaceExitEtc| ...)
    33> (|replaceExitEtc,fn| ...)
    <33 (|replaceExitEtc,fn| ...)
    <32 (|replaceExitEtc| ...)
    32> (|replaceExitEtc| ...)
    33> (|replaceExitEtc,fn| ...)
    <33 (|replaceExitEtc,fn| ...)
    <32 (|replaceExitEtc| ...)
    <31 (|replaceExitEtc,fn| ...)
    <30 (|replaceExitEtc| ...)
    30> (|convertOrCroak| ...)
    31> (|convert| ...)
    <31 (|convert| ...)
    <30 (|convertOrCroak| ...)
    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    28> (|replaceExitEtc| ...)
    29> (|replaceExitEtc,fn| ...)
    <29 (|replaceExitEtc,fn| ...)
    <28 (|replaceExitEtc| ...)
    <27 (|replaceExitEtc,fn| ...)
    <26 (|replaceExitEtc| ...)
    <25 (|replaceExitEtc,fn| ...)
    <24 (|replaceExitEtc| ...)
    <23 (|compSeq1| ...)
    <22 (|compSeq| ...)
    <21 (|compExpression| ...)
    <20 (|comp3| ...)
    <19 (|comp2| ...)
    <18 (|compNoStacking| ...)
    <17 (|comp| ...)
    17> (|comp| ...)
    18> (|compNoStacking| ...)
    19> (|comp2| ...)
    20> (|comp3| ...)
    21> (|compExpression| ...)
    22> (|comp| ...)
    23> (|compNoStacking| ...)
    24> (|comp2| ...)
    25> (|comp3| ...)

```

```

26> (|compColon| ...)
<26 (|compColon| ...)
<25 (|comp3| ...)
<24 (|comp2| ...)
<23 (|compNoStacking| ...)
<22 (|comp| ...)

```

In order to explain the compiler we will walk through the compilation of EQ.spad, which handles equations as mathematical objects. We start the system. Most of the structure in Axiom are circular so we have to the `*print-cycle*` to true.

```
root@spiff:/tmp# axiom -nox
```

```
(1) -> )lisp (setq *print-circle* t)
```

```
Value = T
```

We trace the function we find interesting:

```
(1) -> )lisp (trace |compiler|)
```

```
Value = (|compiler|)
```

11.1.1 The top level compiler command

We compile the spad file. We can see that the `compiler` function gets a list

```
(1) -> )co EQ
```

```
1> (|compiler| (EQ))
```

In order to find this file, the `pathname` and `pathnameType` functions are used to find the location and pathname to the file. They `pathnameType` function eventually returns the fact that this is a spad source file. Once that is known we call the `compileSpad2Cmd` function with a list containing the full pathname as a string.

```

1> (|compiler| (EQ))
2> (|pathname| (EQ))
<2 (|pathname| #p"EQ")
2> (|pathnameType| #p"EQ")
3> (|pathname| #p"EQ")
<3 (|pathname| #p"EQ")
<2 (|pathnameType| NIL)
2> (|pathnameType| "/tmp/EQ.spad")
3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|pathnameType| "/tmp/EQ.spad")

```

```

3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|pathnameType| "/tmp/EQ.spad")
3> (|pathname| "/tmp/EQ.spad")
<3 (|pathname| #p"/tmp/EQ.spad")
<2 (|pathnameType| "spad")
2> (|compileSpad2Cmd| ("/tmp/EQ.spad"))

[compiler helpSpad2Cmd (vol5)]
[compiler selectOptionLC (vol5)]
[compiler pathname (vol5)]
[compiler mergePathnames (vol5)]
[compiler pathnameType (vol5)]
[compiler namestring (vol5)]
[throwKeyedMsg p??]
[findfile p??]
[compileSpad2Cmd p534]
[compileSpadLispCmd p596]
[$newConlist p??]
[$options p??]
[/editfile p??]

— defun compiler —

(defun |compiler| (args)
  "The top level compiler command"
  (let (|$newConlist| optlist optname optargs havenew haveold aft ef af af1)
    (declare (special |$newConlist| |$options| /editfile))
    (setq |$newConlist| nil)
    (cond
      ((and (null args) (null |$options|) (null /editfile))
        (|helpSpad2Cmd| '(|compiler|)))
      (t
        (cond ((null args) (setq args (cons /editfile nil))))
        (setq optlist '(|new| |old| |translate| |constructor|))
        (setq havenew nil)
        (setq haveold nil)
        (do ((t0 |$options| (cdr t0)) (opt nil))
            ((or (atom t0)
                 (progn (setq opt (car t0)) nil)
                 (null (null (and havenew haveold)))))
          nil)
        (setq optname (car opt))
        (setq optargs (cdr opt))
        (case (|selectOptionLC| optname optlist nil)
          (|new| (setq havenew t))
          (|translate| (setq haveold t))
          (|constructor| (setq haveold t))

```

```

      (|old|          (setq haveold t))))
(cond
  ((and havenew haveold) (|throwKeyedMsg| 's2iz0081 nil))
  (t
   (setq af (|pathname| args))
   (setq aft (|pathnameType| af))
   (cond
    ((or haveold (string= aft "spad"))
     (if (null (setq af1 ($findfile af '(|spad|))))
         (|throwKeyedMsg| 's2il0003 (cons (namestring af) nil))
         (|compileSpad2Cmd| (cons af1 nil))))
    ((string= aft "nrllib")
     (if (null (setq af1 ($findfile af '(|nrllib|))))
         (|throwKeyedMsg| 'S2IL0003 (cons (namestring af) nil))
         (|compileSpadLispCmd| (cons af1 nil))))
    (t
     (setq af1 ($findfile af '(|spad|)))
     (cond
      ((and af1 (string= (|pathnameType| af1) "spad"))
       (|compileSpad2Cmd| (cons af1 nil)))
      (t
       (setq ef (|pathname| /editfile))
       (setq ef (|mergePathnames| af ef))
       (cond
        ((boot-equal ef af) (|throwKeyedMsg| 's2iz0039 nil))
        (t
         (setq af ef)
         (cond
          ((string= (|pathnameType| af) "spad")
           (|compileSpad2Cmd| args))
          (t
           (setq af1 ($findfile af '(|spad|)))
           (cond
            ((and af1 (string= (|pathnameType| af1) "spad"))
             (|compileSpad2Cmd| (cons af1 nil)))
            (t (|throwKeyedMsg| 's2iz0039 nil))))))))))))))

```

11.1.2 The Spad compiler top level function

The argument to this function, as noted above, is a list containing the string pathname to the file.

```
2> (|compileSpad2Cmd| ("/tmp/EQ.spad"))
```

There is a fair bit of redundant work to find the full filename and pathname of the file. This needs to be eliminated.

The trace of the functions in this routines is:

```

1> (|selectOptionLC| "compiler" (|abbreviations| |boot| |browse| |cd| |clear| |close| |compiler| |copy
<1 (|selectOptionLC| |compiler|)
1> (|selectOptionLC| |compiler| (|abbreviations| |boot| |browse| |cd| |clear| |close| |compiler| |copy
<1 (|selectOptionLC| |compiler|)
1> (|pathname| (EQ))
<1 (|pathname| #p"EQ")
1> (|pathnameType| #p"EQ")
  2> (|pathname| #p"EQ")
  <2 (|pathname| #p"EQ")
<1 (|pathnameType| NIL)
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|pathnameType| "/tmp/EQ.spad")
  2> (|pathname| "/tmp/EQ.spad")
  <2 (|pathname| #p"/tmp/EQ.spad")
<1 (|pathnameType| "spad")
1> (|compileSpad2Cmd| ("/tmp/EQ.spad"))
  2> (|pathname| ("/tmp/EQ.spad"))
  <2 (|pathname| #p"/tmp/EQ.spad")
  2> (|pathnameType| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
  <2 (|pathnameType| "spad")
  2> (|updateSourceFiles| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
    3> (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathname| #p"/tmp/EQ.spad")
    3> (|pathnameType| #p"/tmp/EQ.spad")
      4> (|pathname| #p"/tmp/EQ.spad")
      <4 (|pathname| #p"/tmp/EQ.spad")
    <3 (|pathnameType| "spad")
    3> (|pathname| ("EQ" "spad" "*"))
    <3 (|pathname| #p"EQ.spad")
    3> (|pathnameType| #p"EQ.spad")
      4> (|pathname| #p"EQ.spad")
      <4 (|pathname| #p"EQ.spad")
    <3 (|pathnameType| "spad")
  <2 (|updateSourceFiles| #p"EQ.spad")
  2> (|namestring| ("/tmp/EQ.spad"))
    3> (|pathname| ("/tmp/EQ.spad"))
    <3 (|pathname| #p"/tmp/EQ.spad")

```

```

<2 (|namestring| "/tmp/EQ.spad")
Compiling AXIOM source code from file /tmp/EQ.spad using old system
compiler.

```

Again we find a lot of redundant work. We finally end up calling **compilerDoit** with a constructed argument list:

```

2> (|compilerDoit| NIL (|rq| |lib|))

[compileSpad2Cmd pathname (vol5)]
[compileSpad2Cmd pathnameType (vol5)]
[compileSpad2Cmd namestring (vol5)]
[compileSpad2Cmd updateSourceFiles (vol5)]
[compileSpad2Cmd selectOptionLC (vol5)]
[compileSpad2Cmd terminateSystemCommand (vol5)]
[nequal p??]
[throwKeyedMsg p??]
[compileSpad2Cmd sayKeyedMsg (vol5)]
[error p??]
[strconc p??]
[object2String p??]
[browserAutoloadOnceTrigger p??]
[spad2AsTranslatorAutoloadOnceTrigger p??]
[compilerDoitWithScreenedLisplib p??]
[compilerDoit p538]
[extendLocalLibdb p??]
[spadPrompt p??]
[$newComp p??]
[$scanIfTrue p??]
[$compileOnlyCertainItems p??]
[$f p??]
[$m p??]
[$QuickLet p??]
[$QuickCode p??]
[$sourceFileTypes p??]
[$InteractiveMode p??]
[$options p??]
[$newConlist p??]
[/editfile p??]

```

— defun compileSpad2Cmd —

```

(defun |compileSpad2Cmd| (args)
  (let (|$newComp| |$scanIfTrue|
        |$compileOnlyCertainItems| |$f| |$m| |$QuickLet| |$QuickCode|
        |$sourceFileTypes| |$InteractiveMode| path optlist fun optname
        optargs fullopt constructor)

```

```

(declare (special |$newComp| |$scanIfTrue|
  |$compileOnlyCertainItems| |$f| |$m| |$QuickLet| |$QuickCode|
  |$sourceFileTypes| |$InteractiveMode| /editfile |$options|
  |$newConlist|))
(setq path (|pathname| args))
(cond
  ((nequal (|pathnameType| path) "spad") (|throwKeyedMsg| 's2iz0082 nil))
  ((null (probe-file path))
    (|throwKeyedMsg| 's2il0003 (cons (|namestring| args) nil)))
  (t
    (setq /editfile path)
    (|updateSourceFiles| path)
    (|sayKeyedMsg| 's2iz0038 (list (|namestring| args)))
    (setq optlist '(|break| |constructor| |functions| |library| |lisp|
      |new| |old| |nbreak| |nolibrary| |noquiet| |vartrace| |quiet|
      |translate|))
    (setq |$QuickLet| t)
    (setq |$QuickCode| t)
    (setq fun '(|rq| |lib|))
    (setq |$sourceFileTypes| '("SPAD"))
    (dolist (opt |$options|)
      (setq optname (car opt))
      (setq optargs (cdr opt))
      (setq fullopt (|selectOptionLC| optname optlist nil))
      (case fullopt
        (|old| nil)
        (|library| (setelt fun 1 '|lib|))
        (|nolibrary| (setelt fun 1 '|nolib|))
        (|quiet| (when (nequal (elt fun 0) '|c|) (setelt fun 0 '|rq|)))
        (|noquiet| (when (nequal (elt fun 0) '|c|) (setelt fun 0 '|rf|)))
        (|nbreak| (setq |$scanIfTrue| t))
        (|break| (setq |$scanIfTrue| nil))
        (|vartrace| (setq |$QuickLet| nil))
        (|lisp| (|throwKeyedMsg| 's2iz0036 (list ")lisp")))
        (|functions|
          (if (null optargs)
              (|throwKeyedMsg| 's2iz0037 (list ")functions"))
              (setq |$compileOnlyCertainItems| optargs)))
        (|constructor|
          (if (null optargs)
              (|throwKeyedMsg| 's2iz0037 (list ")constructor"))
              (progn
                (setelt fun 0 '|c|)
                (setq constructor (mapcar #'|unabbrev| optargs))))))
    (t
      (|throwKeyedMsg| 's2iz0036
        (list (strconc ") " (|object2String| optname))))))
  (setq |$InteractiveMode| nil)
  (cond
    (|$compileOnlyCertainItems|

```

```

(if (null constructor)
  (|sayKeyedMsg| 's2iz0040 nil)
  (|compilerDoitWithScreenedLisplib| constructor fun)))
(t (|compilerDoit| constructor fun)))
(|extendLocalLibdb| |$newConlist|)
(|terminateSystemCommand|)
(|spadPrompt|))))))

```

This trivial function cases on the second argument to decide which combination of operations was requested. For this case we see:

```

(1) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)

... [snip]...

<2 (|/RQ,LIB| T)
<1 (|compilerDoit| T)
(1) ->

```

11.1.3 defun compilerDoit

```

[compilerDoit /rq (vol5)]
[compilerDoit /rf (vol5)]
[compilerDoit member (vol5)]
[sayBrightly p??]
[opOf p??]
[/RQ,LIB p539]
[$byConstructors p599]
[$constructorsSeen p599]

```

— defun compilerDoit —

```

(defun |compilerDoit| (constructor fun)
  (let (|$byConstructors| |$constructorsSeen|)
    (declare (special |$byConstructors| |$constructorsSeen|))
    (cond
      ((equal fun '(|rf| |lib|)) (|/RQ,LIB|)) ; Ignore "noquiet"
      ((equal fun '(|rf| |nolib|)) (/rf))
      ((equal fun '(|rq| |lib|)) (|/RQ,LIB|))
      ((equal fun '(|rq| |nolib|)) (/rq))
      ((equal fun '(|c| |lib|))

```

```
(setq |$byConstructors| (loop for x in constructor collect (|opOf| x)))
(|/RQ,LIB|)
(dolist (x |$byConstructors|)
  (unless (|member| x |$constructorsSeen|)
    (|sayBrightly| '(">>> Warning " |%b| ,x |%d| " was not found"))))))
```

This function simply calls `/rf-1`.

```
(2) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)
3> (|/RF-1 NIL)
...[snip]...
  <3 (|/RF-1 T)
  <2 (|/RQ,LIB| T)
  <1 (|compilerDoit| T)
```

11.1.4 defun `/RQ,LIB`

```
[/rf-1 p540]
[/RQ,LIB echo-meta (vol5)]
[$lisplib p??]
```

— **defun `/RQ,LIB`** —

```
(defun |/RQ,LIB| (&rest foo &aux (echo-meta nil) ($lisplib t))
  (declare (special echo-meta $lisplib) (ignore foo))
  (/rf-1 nil))
```

Since this function is called with nil we fall directly into the call to the function **spad**:

```
(2) -> )co EQ
      Compiling AXIOM source code from file /tmp/EQ.spad using old system
      compiler.
1> (|compilerDoit| NIL (|rq| |lib|))
2> (|/RQ,LIB|)
3> (|/RF-1 NIL)
4> (SPAD "/tmp/EQ.spad")
...[snip]...
  <4 (SPAD T)
```

```

<3 (/RF-1 T)
<2 (|/RQ,LIB| T)
<1 (|compilerDoit| T)

```

11.1.5 defun /rf-1

```

[/rf-1 makeInputFilename (vol5)]
[ncINTERPFILE p595]
[/rf-1 spad (vol5)]
[/editfile p??]
[echo-meta p??]

```

— defun /rf-1 —

```

(defun /rf-1 (ignore)
  (declare (ignore ignore))
  (let* ((input-file (makeInputFilename /editfile))
        (type (pathname-type input-file)))
    (declare (special echo-meta /editfile))
    (cond
      ((string= type "lisp") (load input-file))
      ((string= type "input") (|ncINTERPFILE| input-file echo-meta))
      (t (spad input-file)))))

```

—————

Here we begin the actual compilation process.

```

1> (SPAD "/tmp/EQ.spad")
2> (|makeInitialModemapFrame|)
<2 (|makeInitialModemapFrame| ((NIL)))
2> (INIT-BOOT/SPAD-READER)
<2 (INIT-BOOT/SPAD-READER NIL)
2> (OPEN "/tmp/EQ.spad" :DIRECTION :INPUT)
<2 (OPEN #<input stream "/tmp/EQ.spad">)
2> (INITIALIZE-PREPARSE #<input stream "/tmp/EQ.spad">)
<2 (INITIALIZE-PREPARSE ")abbrev domain EQ Equation")
2> (PREPARSE #<input stream "/tmp/EQ.spad">)
EQ abbreviates domain Equation
<2 (PREPARSE (# # # # # # # ...))
2> (|PARSE-NewExpr|)
<2 (|PARSE-NewExpr| T)
2> (S-PROCESS (|where| # #))
...[snip]...
3> (OPEN "/tmp/EQ.erlib/info" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.erlib/info">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.lsp")

```

```

<3 (OPEN #<input stream "/tmp/EQ.nrllib/EQ.lsp">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.data" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.data">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.c" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.c">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.h" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.h">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.fn" :DIRECTION :OUTPUT)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.fn">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.o" :DIRECTION :OUTPUT :IF-EXISTS :APPEND)
<3 (OPEN #<output stream "/tmp/EQ.nrllib/EQ.o">)
3> (OPEN #p"/tmp/EQ.nrllib/EQ.data")
<3 (OPEN #<input stream "/tmp/EQ.nrllib/EQ.data">)
3> (OPEN #p"/tmp/EQ.nrllib/index.kaf")
<3 (OPEN #<input stream "/tmp/EQ.nrllib/index.kaf">)
<2 (S-PROCESS NIL)
<1 (SPAD T)
1> (OPEN "temp.text" :DIRECTION :OUTPUT)
<1 (OPEN #<output stream "temp.text">)
1> (OPEN "libdb.text")
<1 (OPEN #<input stream "libdb.text">)
1> (OPEN "temp.text")
<1 (OPEN #<input stream "temp.text">)
1> (OPEN "libdb.text" :DIRECTION :OUTPUT)
<1 (OPEN #<output stream "libdb.text">)

```

The major steps in this process involve the **preparse** function. (See book volume 5 for more details). The **preparse** function returns a list of pairs of the form: ((linenumber . linestring) (linenumber . linestring)) For instance, for the file `EQ.spad`, we get:

```

<2 (PREPARSE (
(19 . "Equation(S: Type): public == private where")
(20 . " (Ex ==> OutputForm;")
(21 . "   public ==> Type with")
(22 . "   (\\"=\\" : (S, S) -> $;")
...[skip]...
(202 . "      inv eq == [inv lhs eq, inv rhs eq]);")
(203 . "      if S has ExpressionSpace then")
(204 . "          subst(eq1,eq2) ==")
(205 . "              (eq3 := eq2 pretend Equation S;")
(206 . "                  [subst(lhs eq1,eq3),subst(rhs eq1,eq3)])))))

```

And the **s-process** function which returns a parsed version of the input.

```

2> (S-PROCESS
(|where|
(== (|:| (|Equation| (|:| S |Type|)) |public|) |private|)
(|:|
(|:|
(|:|

```

```

(==> |Ex| |OutputForm|)
(==> |public|
  (|Join| |Type|
    (|with|
      (CATEGORY
        (|Signature| "=" (-> (|,| S S) $))
        (|Signature| |equation| (-> (|,| S S) $))
        (|Signature| |swap| (-> $ $))
        (|Signature| |lhs| (-> $ S))
        (|Signature| |rhs| (-> $ S))
        (|Signature| |map| (-> (|,| (-> S S) $) $))
        (|if| (|has| S (|InnerEvalable| (|,| |Symbol| S)))
          (|Attribute| (|InnerEvalable| (|,| |Symbol| S)))
          NIL)
        (|if| (|has| S |SetCategory|)
          (CATEGORY
            (|Attribute| |SetCategory|)
            (|Attribute| (|CoercibleTo| |Boolean|))
            (|if| (|has| S (|Evalable| S))
              (CATEGORY
                (|Signature| |eval| (-> (|,| $ $) $))
                (|Signature| |eval| (-> (|,| $ (|List| $)) $)))
              NIL))
            NIL)
          (|if| (|has| S |AbelianSemiGroup|)
            (CATEGORY
              (|Attribute| |AbelianSemiGroup|)
              (|Signature| "+" (-> (|,| S $) $))
              (|Signature| "+" (-> (|,| $ S) $)))
              NIL)
            (|if| (|has| S |AbelianGroup|)
              (CATEGORY
                (|Attribute| |AbelianGroup|)
                (|Signature| |leftZero| (-> $ $))
                (|Signature| |rightZero| (-> $ $))
                (|Signature| "-" (-> (|,| S $) $))
                (|Signature| "-" (-> (|,| $ S) $)))
                NIL)
              (|if| (|has| S |SemiGroup|)
                (CATEGORY
                  (|Attribute| |SemiGroup|)
                  (|Signature| "*" (-> (|,| S $) $))
                  (|Signature| "*" (-> (|,| $ S) $)))
                  NIL)
                (|if| (|has| S |Monoid|)
                  (CATEGORY
                    (|Attribute| |Monoid|)
                    (|Signature| |leftOne| (-> $ (|Union| (|,| $ "failed"))))
                    (|Signature| |rightOne| (-> $ (|Union| (|,| $ "failed")))))
                    NIL)
                  (|if| (|has| S |Group|)

```

[illegible]


```

(=
  (|eval| (|,| (|,| (|eqn| |lhs|) |ls|) |lx|))
  (|eval| (|,| (|,| (|eqn| |rhs|) |ls|) |lx|))))
NIL))
(|if| (|has| S (|Evalable| S))
(|;|
(==
  (|:| (|eval| (|,| (|:| |eqn1| $) (|:| |eqn2| $))) $)
  (=
    (|eval|
      (|,| (|eqn1| |lhs|) (|pretend| |eqn2| (|Equation| S))))
    (|eval|
      (|,| (|eqn1| |rhs|) (|pretend| |eqn2| (|Equation| S))))))
  (==
    (|:|
      (|eval| (|,| (|:| |eqn1| $) (|:| |eqn2| (|List| $)))) $)
    (=
      (|eval|
        (|,|
          (|eqn1| |lhs|)
          (|pretend| |eqn2| (|List| (|Equation| S))))
        (|eval|
          (|,|
            (|eqn1| |rhs|)
            (|pretend| |eqn2| (|List| (|Equation| S)))))))
    NIL))
(|if| (|has| S |SetCategory|)
(|;|
(|;|
(==
  (= |eq1| |eq2|)
  (|and|
    (@ (= (|eq1| |lhs|) (|eq2| |lhs|)) |Boolean|)
    (@ (= (|eq1| |rhs|) (|eq2| |rhs|)) |Boolean|)))
  (==
    (|:| (|coerce| (|:| |eqn| $)) |Ex|)
    (= (|::| (|eqn| |lhs|) |Ex|) (|::| (|eqn| |rhs|) |Ex|))))
  (==
    (|:| (|coerce| (|:| |eqn| $)) |Boolean|)
    (= (|eqn| |lhs|) (|eqn| |rhs|))))
NIL))
(|if| (|has| S |AbelianSemiGroup|)
(|;|
(|;|
(==
  (+ |eq1| |eq2|)
  (=
    (+ (|eq1| |lhs|) (|eq2| |lhs|))
    (+ (|eq1| |rhs|) (|eq2| |rhs|))))
  (== (+ |s| |eq2|) (+ (|construct| (|,| |s| |s|)) |eq2|)))

```

```

      (== (+ |eq1| |s|) (+ |eq1| (|construct| (|,| |s| |s|)))))
    NIL))
  (|if| (|has| S |AbelianGroup|)
    (|;|
      (|;|
        (|;|
          (|;|
            (|;|
              (== (- |eq1|) (= (- (|lhs| |eq1|)) (- (|rhs| |eq1|))))
              (== (- |s| |eq2|) (- (|construct| (|,| |s| |s|)) |eq2|)))
              (== (- |eq1| |s|) (- |eq1| (|construct| (|,| |s| |s|)))))
              (== (|leftZero| |eq1|) (= 0 (- (|rhs| |eq1|) (|lhs| |eq1|))))
              (== (|rightZero| |eq1|) (= (- (|lhs| |eq1|) (|rhs| |eq1|)) 0)))
              (== 0 (|equation| (|,| (|elt| S 0) (|elt| S 0)))))
            (==
              (- |eq1| |eq2|)
              (=
                (- (|eq1| |lhs|) (|eq2| |lhs|))
                (- (|eq1| |rhs|) (|eq2| |rhs|))))
            NIL))
    (|if| (|has| S |SemiGroup|)
      (|;|
        (|;|
          (|;|
            (==
              (* (|:| |eq1| $) (|:| |eq2| $))
              (=
                (* (|eq1| |lhs|) (|eq2| |lhs|))
                (* (|eq1| |rhs|) (|eq2| |rhs|))))
            (==
              (* (|:| |l| S) (|:| |eqn| $))
              (= (* |l| (|eqn| |lhs|)) (* |l| (|eqn| |rhs|)))))
            (==
              (* (|:| |l| S) (|:| |eqn| $))
              (= (* |l| (|eqn| |lhs|)) (* |l| (|eqn| |rhs|)))))
            (==
              (* (|:| |eqn| $) (|:| |l| S))
              (= (* (|eqn| |lhs|) |l|) (* (|eqn| |rhs|) |l|)))
            NIL))
      (|if| (|has| S |Monoid|)
        (|;|
          (|;|
            (|;|
              (== 1 (|equation| (|,| (|elt| S 1) (|elt| S 1))))
              (==
                (|recip| |eq1|)
                (|;|
                  (|;|
                    (=> (|case| (|:=| |lh| (|recip| (|lhs| |eq1|))) "failed"))

```

```

        "failed")
      (=> (|case| (|:=| |rh| (|recip| (|rhs| |eq|))) "failed")
        "failed"))
    (|construct| (|,| (|::| |lh| S) (|::| |rh| S))))))
  (==
    (|leftOne| |eq|)
    (|;|
      (=> (|case| (|:=| |re| (|recip| (|lhs| |eq|))) "failed")
        "failed")
      (= 1 (* (|rhs| |eq|) |re|))))))
  (==
    (|rightOne| |eq|)
    (|;|
      (=> (|case| (|:=| |re| (|recip| (|rhs| |eq|))) "failed")
        "failed")
      (= (* (|lhs| |eq|) |re| 1))))))
  NIL))
(|if| (|has| S |Group|)
  (|;|
    (|;|
      (==
        (|inv| |eq|)
        (|construct| (|,| (|inv| (|lhs| |eq|)) (|inv| (|rhs| |eq|))))))
      (== (|leftOne| |eq|) (= 1 (* (|rhs| |eq|) (|inv| (|rhs| |eq|)))))
      (== (|rightOne| |eq|) (= (* (|lhs| |eq|) (|inv| (|rhs| |eq|)) 1)))
      NIL))
  (|if| (|has| S |Ring|)
    (|;|
      (==
        (|characteristic| (|@Tuple|))
        ((|elt| S |characteristic|) (|@Tuple|)))
        (== (* (|::| |i| |Integer|) (|::| |eq| $)) (* (|::| |i| S) |eq|)))
        NIL))
  (|if| (|has| S |IntegralDomain|)
    (==
      (|factorAndSplit| |eq|)
      (|;|
        (|;|
          (=>
            (|has| S (|::| |factor| (-> S (|Factored| S))))
            (|;|
              (|:=| |eq0| (|rightZero| |eq|))
              (COLLECT
                (IN |rcf| (|factors| (|factor| (|lhs| |eq0|))))
                (|construct| (|equation| (|,| (|rcf| |factor|) 0))))))
            (=>
              (|has| S (|Polynomial| |Integer|))
              (|;|
                (|;|
                  (|;|

```

```

(|:=| |eq0| (|rightZero| |eq|))
(==> MF
  (|MultivariateFactorize|
    (|,|
      (|,| (|,| |Symbol| (|IndexedExponents| |Symbol|)) |Integer|)
      (|Polynomial| |Integer|))))
(|:=|
  (|:| |p| (|Polynomial| |Integer|))
  (|pretend| (|lhs| |eq0|) (|Polynomial| |Integer|)))
(COLLECT
  (IN |rcf| (|factors| ((|elt| MF |factor|) |p|)))
  (|construct|
    (|equation| (|,| (|pretend| (|rcf| |factor|) S) 0)))))
(|construct| |eq|))
NIL))
(|if| (|has| S (|PartialDifferentialRing| |Symbol|))
  (==
    (|:| (|differentiate| (|,| (|:| |eq| $) (|:| |sym| |Symbol|))) $)
    (|construct|
      (|,|
        (|differentiate| (|,| (|lhs| |eq|) |sym|))
        (|differentiate| (|,| (|rhs| |eq|) |sym|))))
      NIL))
  (|if| (|has| S |Field|)
    (|;|
      (|;|
        (== (|dimension| (|@Tuple|)) (|::| 2 |CardinalNumber|))
        (==
          (/ (|:| |eq1| $) (|:| |eq2| $))
          (= (/ (|eq1| |lhs|) (|eq2| |lhs|)) (/ (|eq1| |rhs|) (|eq2| |rhs|))))
          (==
            (|inv| |eq|)
            (|construct| (|,| (|inv| (|lhs| |eq|)) (|inv| (|rhs| |eq|)))))
            NIL))
    (|if| (|has| S |ExpressionSpace|)
      (==
        (|subst| (|,| |eq1| |eq2|))
        (|;|
          (|:=| |eq3| (|pretend| |eq2| (|Equation| S)))
          (|construct|
            (|,|
              (|subst| (|,| (|lhs| |eq1|) |eq3|))
              (|subst| (|,| (|rhs| |eq1|) |eq3|))))
            NIL))))
      NIL))))

```

11.1.6 defun spad

```

[spad-reader p??]
[spad addBinding (vol5)]
[spad makeInitialModemapFrame (vol5)]
[spad init-boot/spad-reader (vol5)]
[initialize-prepare p73]
[prepare p76]
[PARSE-NewExpr p411]
[pop-stack-1 p522]
[s-process p550]
[ioclear p??]
[spad shut (vol5)]
[$noSubsumption p??]
[$InteractiveFrame p??]
[$InitialDomainsInScope p??]
[$InteractiveMode p??]
[$spad p549]
[$boot p??]
[curoutstream p??]
[*fileactq-apply* p??]
[line p603]
[optionlist p??]
[echo-meta p??]
[/editfile p??]
[*comp370-apply* p??]
[*eof* p??]
[file-closed p??]
[boot-line-stack p??]
[spad-reader p??]

```

— defun spad —

```

(defun spad (&optional (*spad-input-file* nil) (*spad-output-file* nil)
  &aux (*comp370-apply* #'print-defun)
        (*fileactq-apply* #'print-defun)
        ($spad t) ($boot nil) (optionlist nil) (*eof* nil)
        (file-closed nil) (/editfile *spad-input-file*)
        (|$noSubsumption| |$noSubsumption|) in-stream out-stream)
  (declare (special echo-meta /editfile *comp370-apply* *eof* curoutstream
    file-closed |$noSubsumption| |$InteractiveFrame|
    |$InteractiveMode| optionlist
    boot-line-stack *fileactq-apply* $spad $boot))
  ;; only rebind |$InteractiveFrame| if compiling
  (progv (if (not |$InteractiveMode|) '(|$InteractiveFrame|))
    (if (not |$InteractiveMode|)
      (list (|addBinding| '|$DomainsInScope|

```

```

        '((fluid . |true|))
        (|addBinding| '$Information| nil
          (|makeInitialModemapFrame|))))
(init-boot/spad-reader)
(unwind-protect
  (progn
    (setq in-stream (if *spad-input-file*
                        (open *spad-input-file* :direction :input
                          *standard-input*)
                        (initialize-prepare in-stream))
    (setq out-stream (if *spad-output-file*
                        (open *spad-output-file* :direction :output
                          *standard-output*)
                        (when *spad-output-file*
                          (format out-stream "~&::; -*- Mode:Lisp; Package:Boot  -*-~%~%")
                          (print-package "BOOT"))
                        (setq curoutstream out-stream)
                        (loop
                          (if (or *eof* file-closed) (return nil))
                          (catch 'spad_reader
                            (if (setq boot-line-stack (prepare in-stream))
                                (let ((line (cdar boot-line-stack)))
                                  (declare (special line))
                                  (|PARSE-NewExpr|)
                                  (let ((parseout (pop-stack-1)) )
                                    (when parseout
                                      (let ((*standard-output* out-stream))
                                        (s-process parseout))
                                      (format out-stream "~&"))
                                    )))
                                (ioclear in-stream out-stream)))
                            (if *spad-input-file* (shut in-stream))
                            (if *spad-output-file* (shut out-stream)))
                          t)))
  )

```

11.1.7 defun Interpreter interface to the compiler

```

[curstrm p??]
[def-rename p553]
[new2OldLisp p518]
[parseTransform p93]
[postTransform p359]
[displayPreCompilationErrors p516]
[prettyprint p??]
[s-process processInteractive (vol5)]

```

```

[compTopLevel p554]
[def-process p??]
[displaySemanticErrors p??]
[terpri p??]
[get-internal-run-time p??]
[Index p??]
[$macroassoc p??]
[$newspad p??]
[$PolyMode p??]
[$EmptyMode p131]
[$compUniquelyIfTrue p??]
[$currentFunction p??]
[$postStack p??]
[$topOp p??]
[$semanticErrorStack p??]
[$warningStack p??]
[$exitMode p??]
[$exitModeStack p??]
[$returnMode p??]
[$leaveMode p??]
[$leaveLevelStack p??]
[$top-level p??]
[$insideFunctorIfTrue p??]
[$insideExpressionIfTrue p??]
[$insideCoerceInteractiveHardIfTrue p??]
[$insideWhereIfTrue p??]
[$insideCategoryIfTrue p??]
[$insideCapsuleFunctionIfTrue p??]
[$form p??]
[$DomainFrame p??]
[$e p??]
[$EmptyEnvironment p??]
[$genFVar p??]
[$genSDVar p??]
[$VariableCount p??]
[$previousTime p??]
[$LocalFrame p??]
[$Translation p??]
[$TranslateOnly p??]
[$PrintOnly p??]
[$currentLine p??]
[$InteractiveFrame p??]
[curoutstream p??]

```

— **defun s-process** —

```

(defun s-process (x)
  (prog ((|Index| 0)
        ($macroassoc ())
        ($newspad t)
        (|$PolyMode| |$EmptyMode|)
        (|$compUniquelyIfTrue| nil)
        |$currentFunction|
        (|$postStack| nil)
        |$topOp|
        (|$semanticErrorStack| ())
        (|$warningStack| ())
        (|$exitMode| |$EmptyMode|)
        (|$exitModeStack| ())
        (|$returnMode| |$EmptyMode|)
        (|$leaveMode| |$EmptyMode|)
        (|$leaveLevelStack| ())
        $top_level |$insideFunctorIfTrue| |$insideExpressionIfTrue|
        |$insideCoerceInteractiveHardIfTrue| |$insideWhereIfTrue|
        |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue| |$form|
        (|$DomainFrame| '((NIL)))
        (|$e| |$EmptyEnvironment|)
        (|$genFVar| 0)
        (|$genSDVar| 0)
        (|$VariableCount| 0)
        (|$previousTime| (get-internal-run-time))
        (|$LocalFrame| '((NIL)))
        (curstrm curoutstream) |$s| |$x| |$m| u)
    (declare (special |$Index| $macroassoc $newspad |$PolyMode| |$EmptyMode|
                     |$compUniquelyIfTrue| |$currentFunction| |$postStack| |$topOp|
                     |$semanticErrorStack| |$warningStack| |$exitMode| |$exitModeStack|
                     |$returnMode| |$leaveMode| |$leaveLevelStack| $top_level
                     |$insideFunctorIfTrue| |$insideExpressionIfTrue| | | | | | |
                     |$insideCoerceInteractiveHardIfTrue| |$insideWhereIfTrue|
                     |$insideCategoryIfTrue| |$insideCapsuleFunctionIfTrue| |$form|
                     |$DomainFrame| |$e| |$EmptyEnvironment| |$genFVar| |$genSDVar|
                     |$VariableCount| |$previousTime| |$LocalFrame|
                     curstrm |$s| |$x| |$m| curoutstream $traceflag |$Translation|
                     |$TranslateOnly| |$PrintOnly| |$currentLine| |$InteractiveFrame|))
      (setq $traceflag t)
      (if (not x) (return nil))
      (if $boot
        (setq x (def-rename (|new2OldLisp| x)))
        (setq x (|parseTransform| (|postTransform| x))))
      (when |$TranslateOnly| (return (setq |$Translation| x)))
      (when |$postStack| (|displayPreCompilationErrors|) (return nil))
      (when |$PrintOnly|
        (format t "~S =====>~%" |$currentLine|)
        (return (prettyprint x)))
      (if (not $boot)
        (if |$InteractiveMode|

```

```

(|processInteractive| x nil)
  (when (setq u (|compTopLevel| x |$EmptyMode| |$InteractiveFrame|))
    (setq |$InteractiveFrame| (third u)))
  (def-process x))
(when |$semanticErrorStack| (|displaySemanticErrors|))
(terpri))

```

11.1.8 defun print-defun

```

[is-console p527]
[print-full p??]
[vmisp::optionlist p??]
[$PrettyPrint p??]

```

— defun print-defun —

```

(defun print-defun (name body)
  (let* ((sp (assoc 'vmisp::compiler-output-stream vmisp::optionlist))
        (st (if sp (cdr sp) *standard-output*)))
    (declare (special vmisp::optionlist |$PrettyPrint|))
    (when (and (is-console st) (symbolp name) (fboundp name)
              (not (compiled-function-p (symbol-function name)))))
      (compile name))
    (when (or |$PrettyPrint| (not (is-console st)))
      (print-full body st) (force-output st))))

```

11.1.9 defun def-rename

```

[def-rename1 p554]

```

— defun def-rename —

```

(defun def-rename (x)
  (def-rename1 x))

```

11.1.10 defun def-rename1

[def-rename1 p554]

— defun def-rename1 —

```
(defun def-rename1 (x)
  (cond
    ((symbolp x)
     (let ((y (get x 'rename))) (if y (first y) x)))
    ((and (listp x) x)
     (if (eqcar x 'quote)
         x
         (cons (def-rename1 (first x)) (def-rename1 (cdr x)))))
    (x)))
```

—————

11.1.11 defun compTopLevel

[newComp p??]
 [compOrCroak p556]
 [\$NRTderivedTargetIfTrue p??]
 [\$killOptimizeIfTrue p??]
 [\$forceAdd p??]
 [\$compTimeSum p??]
 [\$resolveTimeSum p??]
 [\$packagesUsed p??]
 [\$envHashTable p??]

— defun compTopLevel —

```
(defun |compTopLevel| (form mode env)
  (let (|$NRTderivedTargetIfTrue| |$killOptimizeIfTrue| |$forceAdd|
        |$compTimeSum| |$resolveTimeSum| |$packagesUsed| |$envHashTable|
        t1 t2 t3 val newmode)
    (declare (special |$NRTderivedTargetIfTrue| |$killOptimizeIfTrue|
                      |$forceAdd| |$compTimeSum| |$resolveTimeSum|
                      |$packagesUsed| |$envHashTable| ))
    (setq |$NRTderivedTargetIfTrue| nil)
    (setq |$killOptimizeIfTrue| nil)
    (setq |$forceAdd| nil)
    (setq |$compTimeSum| 0)
    (setq |$resolveTimeSum| 0)
    (setq |$packagesUsed| NIL)
    (setq |$envHashTable| (make-hashtable 'equal)))
```

```

(dolist (u (car (car env)))
  (dolist (v (cdr u))
    (hput |$envHashTable| (cons (car u) (cons (car v) nil)) t)))
(cond
  ((or (and (consp form) (eq (qfirst form) 'def))
       (and (consp form) (eq (qfirst form) '|where|)
            (progn
              (setq t1 (qrest form))
              (and (consp t1)
                    (progn
                     (setq t2 (qfirst t1))
                     (and (consp t2) (eq (qfirst t2) 'def)))))))
    (setq t3 (|compOrCroak| form mode env))
    (setq val (car t3))
    (setq newmode (second t3))
    (cons val (cons newmode (cons env nil))))
  (t (|compOrCroak| form mode env))))

```

Given:

CohenCategory(): Category == SetCategory with

```

kind:(CExpr)->Boolean
operand:(CExpr,Integer)->CExpr
numberOfOperand:(CExpr)->Integer
construct:(CExpr,CExpr)->CExpr

```

the resulting call looks like:

```

(|compOrCroak|
  (DEF (|CohenCategory|)
    ((|Category|)
     (NIL)
     (|Join|
      (|SetCategory|)
      (CATEGORY |package|
        (SIGNATURE |kind| ((|Boolean|) |CExpr|))
        (SIGNATURE |operand| (|CExpr| |CExpr| (|Integer|)))
        (SIGNATURE |numberOfOperand| ((|Integer|) |CExpr|))
        (SIGNATURE |construct| (|CExpr| |CExpr| |CExpr|))))
    |$EmptyMode|
    (((
      (|$DomainsInScope|
       (FLUID . |true|)
       (special |$EmptyMode| |$NoValueMode|))))))

```

This compiler call expects the first argument `x` to be a DEF form to compile, The second argument, `m`, is the mode. The third argument, `e`, is the environment.

11.1.12 defun compOrCroak

[compOrCroak1 p556]

— defun compOrCroak —

```
(defun |compOrCroak| (form mode env)
  (|compOrCroak1| form mode env nil nil))
```

—————

This results in a call to the inner function with

```
(|compOrCroak1|
  (DEF (|CohenCategory|)
    ((|Category|))
    (NIL)
    (|Join|
      (|SetCategory|)
      (CATEGORY |package|
        (SIGNATURE |kind| ((|Boolean|) |CEpr|))
        (SIGNATURE |operand| (|CEpr| |CEpr| (|Integer|)))
        (SIGNATURE |numberOfOperand| ((|Integer|) |CEpr|))
        (SIGNATURE |construct| (|CEpr| |CEpr| |CEpr|))))
    |$EmptyMode|
    (((
      |$DomainsInScope|
      (FLUID . |true|)
      (special |$EmptyMode| |$NoValueMode|))))
    NIL
    NIL
    |comp|)
```

The inner function augments the environment with information from the compiler stack `$compStack` and `$compErrorMessageStack`. Note that these variables are passed in the argument list so they get preserved on the call stack. The calling function gets called for every inner form so we use this implicit stacking to retain the information.

11.1.13 defun compOrCroak1

[comp p557]

[compOrCroak1,compactify p595]

[stackSemanticError p??]

```

[mkErrorExpr p??]
[displaySemanticErrors p??]
[say p??]
[displayComp p??]
[userError p??]
[$compStack p??]
[$compErrorMessageStack p??]
[$level p??]
[$s p??]
[$scanIfTrue p??]
[$exitModeStack p??]
[compOrCroak p556]

```

— defun compOrCroak1 —

```

(defun |compOrCroak1| (form mode env |$compStack| |$compErrorMessageStack|)
  (declare (special |$compStack| |$compErrorMessageStack|))
  (let (td errorMessage)
    (declare (special |$level| |$s| |$scanIfTrue| |$exitModeStack|))
    (cond
      ((setq td (catch '|compOrCroak| (|comp| form mode env))) td)
      (t
       (setq |$compStack|
              (cons (list form mode env |$exitModeStack|) |$compStack|))
       (setq |$s| (|compOrCroak1,compactify| |$compStack|))
       (setq |$level| (|#| |$s|))
       (setq errorMessage
              (if |$compErrorMessageStack|
                  (car |$compErrorMessageStack|)
                  '|unspecified error|))
       (cond
         (|$scanIfTrue|
          (|stackSemanticError| errorMessage (|mkErrorExpr| |$level|))
          (list '|failedCompilation| mode env ))
         (t
          (|displaySemanticErrors|)
          (say "***** comp fails at level " |$level| " with expression: *****")
          (|displayComp| |$level|)
          (|userError| errorMessage))))))

```

11.1.14 defun comp

```

[compNoStacking p558]
[$compStack p??]

```

[\$exitModeStack p??]

— **defun comp** —

```
(defun |comp| (form mode env)
  (let (td)
    (declare (special |$compStack| |$exitModeStack|))
    (if (setq td (|compNoStacking| form mode env))
        (setq |$compStack| nil)
        (push (list form mode env |$exitModeStack|) |$compStack|))
    td))
```

—————

11.1.15 **defun compNoStacking**

\$Representation is bound in compDefineFunctor, set by doIt. This hack says that when something is undeclared, \$ is preferred to the underlying representation – RDJ 9/12/83

[comp2 p559]
 [compNoStacking1 p558]
 [\$compStack p??]
 [\$Representation p??]
 [\$EmptyMode p131]

— **defun compNoStacking** —

```
(defun |compNoStacking| (form mode env)
  (let (td)
    (declare (special |$compStack| |$Representation| |$EmptyMode|))
    (if (setq td (|comp2| form mode env))
        (if (and (equal mode |$EmptyMode|) (equal (second td) |$Representation|))
            (list (car td) '$ (third td))
            td)
        (|compNoStacking1| form mode env |$compStack|))))
```

—————

11.1.16 **defun compNoStacking1**

[get p??]
 [comp2 p559]
 [\$compStack p??]

— **defun compNoStacking1** —

```
(defun |compNoStacking1| (form mode env |$compStack|)
  (declare (special |$compStack|))
  (let (u td)
    (if (setq u (|get| (if (eq mode '$) '|Rep| mode) '|value| env))
      (if (setq td (|comp2| form (car u) env))
          (list (car td) mode (third td))
          nil)
      nil)))
```

11.1.17 defun comp2

```
[comp3 p560]
[isDomainForm p338]
[isFunctor p233]
[insert p??]
[opOf p??]
[nequal p??]
[addDomain p232]
[$bootStrapMode p??]
[$packagesUsed p??]
[$lisplib p??]
```

— defun comp2 —

```
(defun |comp2| (form mode env)
  (let (tmp1)
    (declare (special |$bootStrapMode| |$packagesUsed| $lisplib))
    (when (setq tmp1 (|comp3| form mode env))
      (destructuring-bind (y mprime env) tmp1
        (when (and $lisplib (|isDomainForm| form env) (|isFunctor| form))
          (setq |$packagesUsed| (|insert| (list (|opOf| form)) |$packagesUsed|)))
        ; isDomainForm test needed to prevent error while compiling Ring
        ; $bootStrapMode-test necessary for compiling Ring in $bootStrapMode
        (if (and (nequal mode mprime)
                  (or |$bootStrapMode| (|isDomainForm| mprime env)))
            (list y mprime (|addDomain| mprime env))
            (list y mprime env))))))
```

11.1.18 defun comp3

```
[addDomain p232]
[compWithMappingMode p586]
[compAtom p565]
[getmode p??]
[applyMapping p562]
[compApply p563]
[compColon p275]
[compCoerce p352]
[stringPrefix? p??]
[comp3 pname (vol5)]
[compTypeOf p564]
[compExpression p571]
[comp3 member (vol5)]
[getDomainsInScope p234]
[$e p??]
[$insideCompTypeOf p??]
```

— defun comp3 —

```
(defun |comp3| (form mode |$e|)
  (declare (special |$e|))
  (let (env a op ml u sig varlist tmp3 body tt xprime tmp1 mprime tmp2 eprime)
    (declare (special |$insideCompTypeOf|))
    (setq |$e| (|addDomain| mode |$e|))
    (setq env |$e|)
    (cond
      ((and (consp mode) (eq (qfirst mode) '|Mapping|))
        (|compWithMappingMode| form mode env))
      ((and (consp mode) (eq (qfirst mode) 'quote)
        (progn
          (setq tmp1 (qrest mode))
          (and (consp tmp1) (eq (qrest tmp1) nil)
            (progn (setq a (qfirst tmp1)) t))))
        (when (equal form a) (list form mode |$e|)))
      ((stringp mode)
        (when (and (atom form)
          (or (equal mode form) (equal mode (princ-to-string form))))
          (list mode mode env )))
      ((or (null form) (atom form)) (|compAtom| form mode env))
      (t
        (setq op (car form))
        (cond
          ((and (progn
            (setq tmp1 (|getmode| op env))
            (and (consp tmp1)
              (eq (qfirst tmp1) '|Mapping|)
```

```

        (progn (setq ml (qrest tmp1)) t)))
      (setq u (|applyMapping| form mode env ml)))
    u)
  ((and (consp op) (eq (qfirst op) 'kappa)
    (progn
      (setq tmp1 (qrest op))
      (and (consp tmp1)
        (progn
          (setq sig (qfirst tmp1))
          (setq tmp2 (qrest tmp1))
          (and (consp tmp2)
            (progn
              (setq varlist (qfirst tmp2))
              (setq tmp3 (qrest tmp2))
              (and (consp tmp3)
                (eq (qrest tmp3) nil)
                (progn
                  (setq body (qfirst tmp3))
                  t))))))))
      (|compApply| sig varlist body (cdr form) mode env))
    ((eq op '|:|) (|compColon| form mode env))
    ((eq op '|::|) (|compCoerce| form mode env))
    ((and (null (eq |$insideCompTypeOf| t))
      (|stringPrefix?| "TypeOf" (pname op)))
      (|compTypeOf| form mode env))
    (t
      (setq tt (|compExpression| form mode env))
      (cond
        ((and (consp tt)
          (progn
            (setq xprime (qfirst tt))
            (setq tmp1 (qrest tt))
            (and (consp tmp1)
              (progn
                (setq mprime (qfirst tmp1))
                (setq tmp2 (qrest tmp1))
                (and (consp tmp2)
                  (eq (qrest tmp2) nil)
                  (progn
                    (setq eprime (qfirst tmp2))
                    t))))
              (null (|member| mprime (|getDomainsInScope| eprime))))
            (list xprime mprime (|addDomain| mprime eprime)))
          (t tt)))))))))

```

11.1.19 defun applyMapping

```
[nequal p??]
[isCategoryForm p??]
[sublis p??]
[comp p557]
[convert p568]
[member p??]
[get p??]
[getAbbreviation p285]
[encodeItem p150]
[$FormalMapVariableList p250]
[$form p??]
[$op p??]
[$prefix p??]
[$formalArgList p??]
```

— defun applyMapping —

```
(defun |applyMapping| (t0 m e ml)
  (prog (op argl mlp temp1 arglp nprefix opp form pairlis)
    (declare (special |$FormalMapVariableList| |$form| |$op| |$prefix|
                      |$formalArgList|))

    (return
     (progn
      (setq op (car t0))
      (setq argl (cdr t0))
      (cond
       ((nequal (|#| argl) (1- (|#| ml))) nil)
       ((|isCategoryForm| (car ml) e)
        (setq pairlis
         (loop for a in argl for v in |$FormalMapVariableList|
              collect (cons v a)))
        (setq mlp (sublis pairlis ml))
        (setq arglp
         (loop for x in argl for mp in (rest mlp)
              collect (car
                       (progn
                        (setq temp1 (or (|comp| x mp e) (return '|failed|)))
                        (setq e (caddr temp1))
                        temp1))))
        (when (eq arglp '|failed|) (return nil))
        (setq form (cons op arglp))
        (|convert| (list form (car mlp) e) m))
      (t
       (setq arglp
        (loop for x in argl for mp in (rest ml)
             collect (car
```

```

      (progn
        (setq temp1 (or (|comp| x mp e) (return '|failed|)))
        (setq e (caddr temp1))
        temp1))))
(when (eq arglp '|failed|) (return nil))
(setq form
  (cond
    ((and (null (|member| op |$formalArgList|))
      (atom op)
      (null (|get| op '|value| e))))
    (setq nprefix
      (or |$prefix| (|getAbbreviation| |$op| (|#| (cdr |$form|)))))
    (setq opp
      (intern (strconc
        (|encodeItem| nprefix) '|;| (|encodeItem| op))))
    (cons opp (append arglp (list '$))))
  (t
    (cons '|call| (cons (list '|applyFun| op) arglp)))))
(setq pairlis
  (loop for a in arglp for v in |$FormalMapVariableList|
    collect (cons v a)))
(|convert| (list form (sublis pairlis (car ml) e) m))))))

```

11.1.20 defun compApply

```

[comp p557]
[Pair p??]
[removeEnv p??]
[resolve p356]
[AddContour p??]
[$EmptyMode p131]

```

— defun compApply —

```

(defun |compApply| (sig varl body argl m e)
  (let (temp1 argtl contour code mq bodyq)
    (declare (special |$EmptyMode|))
    (setq argtl
      (loop for x in argl
        collect (progn
          (setq temp1 (|comp| x |$EmptyMode| e))
          (setq e (caddr temp1))
          temp1)))
    (setq contour
      (loop for x in varl

```

```

      for mq in (cdr sig)
      for a in argl
    collect
      (|Pair| x
        (list
          (list 'mode| mq)
          (list 'value| (|removeEnv| (|comp| a mq e))))))
    (setq code
      (cons (list 'lambda varl bodyq)
        (loop for tt in argtl
          collect (car tt))))
    (setq mq (|resolve| m (car sig)))
    (setq bodyq (car (|comp| body mq (|addContour| contour e))))
    (list code mq e))

```

11.1.21 defun compTypeOf

```

[eqsubstlist p??]
[get p??]
[put p??]
[comp3 p560]
[$insideCompTypeOf p??]
[$FormalMapVariableList p250]

```

— defun compTypeOf —

```

(defun |compTypeOf| (form mode env)
  (let (|$insideCompTypeOf| op argl newModemap)
    (declare (special |$insideCompTypeOf| |$FormalMapVariableList|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq |$insideCompTypeOf| t)
    (setq newModemap
      (eqsubstlist argl |$FormalMapVariableList| (|get| op 'modemap| env)))
    (setq env (|put| op 'modemap| newModemap| env))
    (|comp3| form mode env)))

```

11.1.22 defun compColonInside

```

[addDomain p232]
[comp p557]

```

```
[coerce p346]
[stackWarning p??]
[opOf p??]
[stackSemanticError p??]
[$newCompilerUnionFlag p??]
[$EmptyMode p131]
```

— **defun compColonInside** —

```
(defun |compColonInside| (form mode env mprime)
  (let (mpp warningMessage td tprime)
    (declare (special |$newCompilerUnionFlag| |$EmptyMode|))
    (setq env (|addDomain| mprime env))
    (when (setq td (|comp| form |$EmptyMode| env))
      (cond
        ((equal (setq mpp (second td)) mprime)
         (setq warningMessage
          (list '|:| mprime '| -- should replace by @|))))
        (setq td (list (car td) mprime (third td)))
        (when (setq tprime (|coerce| td mode))
          (cond
            (warningMessage (|stackWarning| warningMessage))
            ((and |$newCompilerUnionFlag| (eq (|opOf| mpp) '|Union|))
             (setq tprime
              (|stackSemanticError|
               (list '|cannot pretend | form '| of mode | mpp '| to mode | mprime |
                nil))))
            (t
             (|stackWarning|
              (list '|:| mprime '| -- should replace by pretend|))))
          tprime))))
```

11.1.23 defun compAtom

```
[compAtomWithModemap p567]
[get p??]
[modeIsAggregateOf p??]
[compList p570]
[compVector p344]
[convert p568]
[isSymbol p??]
[compSymbol p569]
[primitiveType p568]
[primitiveType p568]
```

[\$Expression p??]

— defun compAtom —

```
(defun |compAtom| (form mode env)
  (prog (tmp1 tmp2 r td tt)
    (declare (special |$Expression|))
    (return
      (cond
        ((setq td
          (|compAtomWithModemap| form mode env (|get| form '|modemap| env))) td)
        ((eq form '|nil|)
          (setq td
            (cond
              ((progn
                (setq tmp1 (|modeIsAggregateOf| '|List| mode env))
                (and (consp tmp1)
                  (progn
                    (setq tmp2 (qrest tmp1))
                    (and (consp tmp2)
                      (eq (qrest tmp2) nil)
                      (progn
                        (setq r (qfirst tmp2)) t))))))
              (|compList| form (list '|List| r) env))
            ((progn
              (setq tmp1 (|modeIsAggregateOf| '|Vector| mode env))
              (and (consp tmp1)
                (progn
                  (setq tmp2 (qrest tmp1))
                  (and (consp tmp2) (eq (qrest tmp2) nil)
                    (progn
                      (setq r (qfirst tmp2)) t))))))
              (|compVector| form (list '|Vector| r) env))))
          (when td (|convert| td mode)))
        (t
          (setq tt
            (cond
              ((|isSymbol| form) (or (|compSymbol| form mode env) (return nil)))
              ((and (equal mode |$Expression|)
                (|primitiveType| form)) (list form mode env ))
              ((stringp form) (list form form env ))
              (t (list form (or (|primitiveType| form) (return nil)) env ))))
            (|convert| tt mode))))))
```

—

11.1.24 defun compAtomWithModemap

[transImplementation p567]
 [modeEqual p357]
 [convert p568]
 [\$NoValueMode p131]

— **defun compAtomWithModemap** —

```
(defun |compAtomWithModemap| (x m env v)
  (let (tt transimp y)
    (declare (special |$NoValueMode|))
    (cond
      ((setq transimp
        (loop for map in v
          when ; map is [[.,target],[.,fn]]
            (and (consp map) (consp (qcar map)) (consp (qcdar map))
              (eq (qcddar map) nil)
              (consp (qcdr map)) (eq (qcddr map) nil)
              (consp (qcadr map)) (consp (qcdadr map))
              (eq (qcddadr map) nil))
          collect
            (list (|transImplementation| x map (qcadadr map)) (qcadar map) env)))
        (cond
          ((setq tt
            (let (result)
              (loop for item in transimp
                when (|modeEqual| m (cadr item))
                do (setq result (or result item)))
              result))
            tt)
          ((eq1 1 (|#| (setq transimp
            (loop for ta in transimp
              when (setq y (|convert| ta m))
              collect y))))
            (car transimp))
          ((and (< 0 (|#| transimp)) (equal m |$NoValueMode|))
            (car transimp))
          (t nil))))))
```

—————

11.1.25 defun transImplementation

[genDeltaEntry p??]

— **defun transImplementation** —

```
(defun |transImplementation| (op map fn)
  (setq fn (|genDeltaEntry| (cons op map)))
  (if (and (consp fn) (eq (qcar fn) 'xlam))
      (cons fn nil)
      (cons '|call| (cons fn nil)))))
```

11.1.26 defun convert

[resolve p356]
[coerce p346]

— defun convert —

```
(defun |convert| (td mode)
  (let (res)
    (when (setq res (|resolve| (second td) mode))
      (|coerce| td res)))))
```

11.1.27 defun primitiveType

[\$DoubleFloat p??]
[\$NegativeInteger p??]
[\$PositiveInteger p??]
[\$NonNegativeInteger p??]
[\$String p339]
[\$EmptyMode p131]

— defun primitiveType —

```
(defun |primitiveType| (form)
  (declare (special |$DoubleFloat| |$NegativeInteger| |$PositiveInteger|
                    |$NonNegativeInteger| |$String| |$EmptyMode|))
  (cond
    ((null form) |$EmptyMode|)
    ((stringp form) |$String|)
    ((integerp form)
     (cond
       ((eql form 0) |$NonNegativeInteger|)
       ((> form 0) |$PositiveInteger|)
       (t |$NegativeInteger|)))
    ((floatp form) |$DoubleFloat|))
```

```
(t nil)))
```

11.1.28 defun compSymbol

```
[getmode p??]
[get p??]
[NRTgetLocalIndex p192]
[compSymbol member (vol5)]
[isFunction p??]
[errorRef p??]
[stackMessage p??]
[$Symbol p??]
[$Expression p??]
[$FormalMapVariableList p250]
[$compForModeIfTrue p??]
[$formalArgList p??]
[$NoValueMode p131]
[$functorLocalParameters p??]
[$Boolean p??]
[$NoValue p??]
```

— defun compSymbol —

```
(defun |compSymbol| (form mode env)
  (let (v mprime newmode)
    (declare (special |$Symbol| |$Expression| |$FormalMapVariableList|
                      |$compForModeIfTrue| |$formalArgList| |$NoValueMode|
                      |$functorLocalParameters| |$Boolean| |$NoValue|))
    (cond
      ((eq form '|$NoValue|) (list '|$NoValue| |$NoValueMode| env ))
      ((|isFluid| form)
       (setq newmode (|getmode| form env))
       (when newmode (list form (|getmode| form env) env)))
      ((eq form '|true|) (list '(quote t) |$Boolean| env ))
      ((eq form '|false|) (list nil |$Boolean| env ))
      ((or (equal form mode)
            (|get| form '|isLiteral| env)) (list (list 'quote form) form env))
      ((setq v (|get| form '|value| env))
       (cond
         ((member form |$functorLocalParameters|)
          ; s will be replaced by an ELT form in beforeCompile
          (|NRTgetLocalIndex| form)
          (list form (second v) env))
         (t
```

```

      ; form has been SETQd
      (list form (second v) env)))
((setq mprime (|getmode| form env))
 (cond
  ((and (null (|member| form |$formalArgList|))
        (null (member form |$FormalMapVariableList|))
        (null (|isFunction| form env))
        (null (eq |$compForModeIfTrue| t)))
   (|errorRef| form)))
 (list form mprime env ))
(member form |$FormalMapVariableList|)
(|stackMessage| (list '|no mode found for| form )))
((or (equal mode |$Expression|) (equal mode |$Symbol|))
 (list (list 'quote form) mode env ))
((null (|isFunction| form env)) (|errorRef| form))))

```

11.1.29 defun compList

[comp p557]

— defun compList —

```

(defun |compList| (form mode env)
  (let (tmp1 tmp2 t0 failed (newmode (second mode)))
    (if (null form)
      (list nil mode env)
      (progn
       (setq t0
        (do ((t3 form (cdr t3)) (x nil))
            ((or (atom t3) failed) (unless failed (nreverse0 tmp2)))
          (setq x (car t3))
          (if (setq tmp1 (|comp| x newmode env))
            (progn
             (setq newmode (second tmp1))
             (setq env (third tmp1))
             (push tmp1 tmp2))
            (setq failed t))))
       (unless failed
        (cons
         (cons 'list (loop for texpr in t0 collect (car texpr)))
         (list (list '|List| newmode) env)))))))

```

11.1.30 defun compExpression

```
[get1 p??]
[compForm p571]
[$insideExpressionIfTrue p??]
```

— **defun compExpression** —

```
(defun |compExpression| (form mode env)
  (let (|$insideExpressionIfTrue| fn)
    (declare (special |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| t)
    (if (and (atom (car form)) (setq fn (get1 (car form) 'special)))
        (funcall fn form mode env)
        (|compForm| form mode env))))
```

11.1.31 defun compForm

```
[compForm1 p571]
[compArgumentsAndTryAgain p585]
[stackMessageIfNone p??]
```

— **defun compForm** —

```
(defun |compForm| (form mode env)
  (cond
    ((|compForm1| form mode env))
    ((|compArgumentsAndTryAgain| form mode env))
    (t (|stackMessageIfNone| (list 'cannot compile '|%b| form '|%d| )))))
```

11.1.32 defun compForm1

```
[length p??]
[outputComp p337]
[compOrCroak p556]
[compExpressionList p578]
[coerceable p350]
[comp p557]
[coerce p346]
[compForm2 p579]
```

```
[augModemapsFromDomain1 p237]
[getFormModemaps p575]
[nreverse0 p??]
[addDomain p232]
[compToApply p573]
[$NumberOfArgsIfInteger p??]
[$Expression p??]
[$EmptyMode p131]
```

— **defun compForm1** —

```
(defun |compForm1| (form mode env)
  (let (|NumberOfArgsIfInteger| op arg1 domain tmp1 opprime ans mmList td
        tmp2 tmp3 tmp4 tmp5 tmp6 tmp7)
    (declare (special |NumberOfArgsIfInteger| |$Expression| |$EmptyMode|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (setq |NumberOfArgsIfInteger| (|#| arg1))
    (cond
      ((eq op '|error|)
       (list
        (cons op
              (dolist (x arg1 (nreverse0 tmp4))
                (setq tmp2 (|outputComp| x env))
                (setq env (third tmp2))
                (push (car tmp2) tmp4)))
              mode env))
        ((and (consp op) (eq (qfirst op) '|elt|)
         (progn
          (setq tmp3 (qrest op))
          (and (consp tmp3)
               (progn
                (setq domain (qfirst tmp3))
                (setq tmp1 (qrest tmp3))
                (and (consp tmp1)
                     (eq (qrest tmp1) nil)
                     (progn
                      (setq opprime (qfirst tmp1))
                      t))))))
          (cond
            ((eq domain '|Lisp|)
             (list
              (cons opprime
                    (dolist (x arg1 (nreverse tmp7))
                      (setq tmp2 (|compOrCroak| x |$EmptyMode| env))
                      (setq env (third tmp2))
                      (push (car tmp2) tmp7)))
                    mode env))
              ((and (equal domain |$Expression|) (eq opprime '|construct|))
```

```

(|compExpressionList| arg1 mode env))
((and (eq opprime 'collect) (|coerceable| domain mode env))
 (when (setq td (|comp| (cons opprime arg1) domain env))
  (|coerce| td mode)))
((and (consp domain) (eq (qfirst domain) '|Mapping|)
 (setq ans
  (|compForm2| (cons opprime arg1) mode
   (setq env (|augModemapsFromDomain1| domain domain env))
   (dolist (x (|getFormModemaps| (cons opprime arg1) env)
    (nreverse0 tmp6))
    (when
     (and (consp x)
      (and (consp (qfirst x)) (equal (qcaar x) domain)))
     (push x tmp6))))))
 ans)
((setq ans
 (|compForm2| (cons opprime arg1) mode
  (setq env (|addDomain| domain env))
  (dolist (x (|getFormModemaps| (cons opprime arg1) env)
   (nreverse0 tmp5))
   (when
    (and (consp x)
     (and (consp (qfirst x)) (equal (qcaar x) domain)))
    (push x tmp5))))))
 ans)
((and (eq opprime '|construct|) (|coerceable| domain mode env))
 (when (setq td (|comp| (cons opprime arg1) domain env))
  (|coerce| td mode)))
 (t nil)))
(t
 (setq env (|addDomain| mode env))
 (cond
  ((and (setq mmList (|getFormModemaps| form env))
   (setq td (|compForm2| form mode env mmList)))
   td)
  (t
   (|compToApply| op arg1 mode env))))))

```

11.1.33 defun compToApply

[compNoStacking p558]
 [compApplication p574]
 [\$EmptyMode p131]

— defun compToApply —

```
(defun |compToApply| (op argl m e)
  (let (tt m1)
    (declare (special |$EmptyMode|))
    (setq tt (|compNoStacking| op |$EmptyMode| e))
    (when tt
      (setq m1 (cadr tt))
      (cond
        ((and (consp (car tt)) (eq (qcar (car tt)) 'quote)
              (consp (qcdr (car tt))) (eq (qcddr (car tt)) nil)
              (equal (qcadr (car tt)) m1))
         nil)
        (t
         (|compApplication| op argl m (caddr tt) tt))))))
```

11.1.34 defun compApplication

```
[eltForm p??]
[resolve p356]
[coerce p346]
[strconc p??]
[encodeItem p150]
[getAbbreviation p285]
[length p??]
[member p??]
[comp p557]
[nequal p??]
[isCategoryForm p??]
[$Category p??]
[$formatArgList p??]
[$op p??]
[$form p??]
[$prefix p??]
```

— defun compApplication —

```
(defun |compApplication| (op argl m env tt)
  (let (argml retm temp1 argT1 nprefix opp form eltForm)
    (declare (special |$form| |$op| |$prefix| |$formalArgList| |$Category|))
    (cond
      ((and (consp (cadr tt)) (eq (qcar (cadr tt)) '|Mapping|)
            (consp (qcdr (cadr tt))))
       (setq retm (qcadr (cadr tt)))
       (setq argml (qcddr (cadr tt)))
       (cond
```

```

((nequal (|#| argl) (|#| argml)) nil)
(t
 (setq retm (|resolve| m retm))
 (cond
  ((or (equal retm |$Category|) (|isCategoryForm| retm env))
   nil)
  (t
   (setq argTl
    (loop for x in argl for m in argml
      collect (progn
        (setq temp1 (or (|comp| x m env) (return '|failed|)))
        (setq env (caddr temp1))
        temp1)))
   (cond
    ((eq argTl '|failed|) nil)
    (t
     (setq form
      (cond
       ((and
        (null
         (or (|member| op |$formalArgList|)
              (|member| (car tt) |$formalArgList|)))
         (atom (car tt)))
        (setq nprefix
         (or |$prefix| (|getAbbreviation| |$op| (|#| (cdr |$form|)))))
        (setq opp
         (intern
          (strconc (|encodeItem| nprefix) '|;' (|encodeItem| (car tt)))))
        (cons opp
         (append
          (loop for item in argTl collect (car item))
          (list '$))))
       (t
        (cons '|call|
         (cons (list '|applyFun| (car tt))
                  (loop for item in argTl collect (car item))))))
        (|coerce| (list form retm env) (|resolve| retm m))))))))
  ((eq op '|elt|) nil)
  (t
   (setq eltForm (cons '|elt| (cons op argl)))
   (|comp| eltForm m env))))

```

11.1.35 defun getFormModemaps

```

[qcar p??]
[qcdr p??]

```

```

[getFormModemaps p575]
[nreverse0 p??]
[get p??]
[nequal p??]
[eltModemapFilter p577]
[last p??]
[length p??]
[stackMessage p??]
[$insideCategoryPackageIfTrue p??]

```

— defun getFormModemaps —

```

(defun |getFormModemaps| (form env)
  (let (op arg1 domain op1 modemapList nargs finalModemapList)
    (declare (special |$insideCategoryPackageIfTrue|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (cond
      ((and (consp op) (eq (qfirst op) '|elt|) (CONSP (qrest op))
        (consp (qcddr op)) (eq (qcddr op) nil))
       (setq op1 (third op))
       (setq domain (second op))
       (loop for x in (|getFormModemaps| (cons op1 arg1) env)
         when (and (consp x) (consp (qfirst x)) (equal (qcaar x) domain))
           collect x))
      ((null (atom op)) nil)
      (t
       (setq modemapList (|get| op '|modemap| env))
       (when |$insideCategoryPackageIfTrue|
        (setq modemapList
          (loop for x in modemapList
            when (and (consp x) (consp (qfirst x)) (nequal (qcaar x) '$))
              collect x))))))
    (cond
      ((eq op '|elt|)
       (setq modemapList (|eltModemapFilter| (|last| arg1) modemapList env)))
      ((eq op '|setelt|)
       (setq modemapList (|seteltModemapFilter| (CADR arg1) modemapList env))))
    (setq nargs (|#| arg1))
    (setq finalModemapList
      (loop for mm in modemapList
        when (equal (|#| (cddar mm)) nargs)
          collect mm))
    (when (and modemapList (null finalModemapList))
      (|stackMessage|
       (list '|no modemap for| '|%b| op '|%d| '|with | nargs '| arguments|)))
    finalModemapList))

```

11.1.36 defun eltModemapFilter

[qcar p??]
 [qcdr p??]
 [isConstantId p??]
 [stackMessage p??]

— defun eltModemapFilter —

```
(defun |eltModemapFilter| (name mmList env)
  (let (z)
    (if (|isConstantId| name env)
      (cond
        ((setq z
          (loop for mm in mmList
            when (and (consp mm) (consp (qfirst mm)) (consp (qcdr mm))
              (consp (qcddar mm))
              (consp (qcdddar mm))
              (equal (fourth (first mm)) name))
          collect mm))
        (t
          (|stackMessage|
            (list '|selector variable: | name '| is undeclared and unbound|))
            nil))
      mmList)))
```

11.1.37 defun seteltModemapFilter

[isConstantId p??]
 [stackMessage p??]

— defun seteltModemapFilter —

```
(defun |seteltModemapFilter| (name mmList env)
  (let (z)
    (if (|isConstantId| name env)
      (cond
        ((setq z
          (loop for mm in mmList
            when (equal (car (cdddar mm)) name)
            collect mm))
        (t
          (|stackMessage|
            (list '|selector variable: | name '| is undeclared and unbound|))
            nil))
      mmList)))
```

```

      z)
    (t
      (|stackMessage|
        (list '|selector variable: | name '| is undeclared and unbound|))
      nil))
    mmList)))

```

11.1.38 defun compExpressionList

```

[nreverse0 p??]
[comp p557]
[convert p568]
[$Expression p??]

```

— defun compExpressionList —

```

(defun |compExpressionList| (arg1 m env)
  (let (tmp1 tlst)
    (declare (special |$Expression|))
    (setq tlst
      (prog (result)
        (return
          (do ((tmp2 arg1 (cdr tmp2)) (x nil))
              ((or (atom tmp2)) (nreverse0 result))
            (setq x (car tmp2))
            (setq result
              (cons
                (progn
                  (setq tmp1 (or (|comp| x |$Expression| env) (return '|failed|)))
                  (setq env (third tmp1))
                  tmp1)
                result))))))
    (unless (eq tlst '|failed|)
      (|convert|
        (list (cons 'list
          (prog (result)
            (return
              (do ((tmp3 tlst (cdr tmp3)) (y nil))
                  ((or (atom tmp3)) (nreverse0 result))
                (setq y (car tmp3))
                (setq result (cons (car y) result))))))
          |$Expression| env)
        m))))

```

11.1.39 defun compForm2

```

[take p??]
[length p??]
[nreverse0 p??]
[sublis p??]
[assoc p??]
[PredImplies p??]
[isSimple p??]
[compUniquely p584]
[compFormPartiallyBottomUp p584]
[compForm3 p580]
[$EmptyMode p131]
[$TriangleVariableList p??]

```

— defun compForm2 —

```

(defun |compForm2| (form mode env modemapList)
  (let (op arg1 sarg1 aList dc cond nsig v ncond deleteList newList td t1
        partialModeList tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7)
    (declare (special |$EmptyMode| |$TriangleVariableList|))
    (setq op (car form))
    (setq arg1 (cdr form))
    (setq sarg1 (take (|#| arg1) |$TriangleVariableList|))
    (setq aList (mapcar #'(lambda (x y) (cons x y)) sarg1 arg1))
    (setq modemaplist (sublis aList modemapList))
    ; now delete any modemaps that are subsumed by something else, provided
    ; the conditions are right (i.e. subsumer true whenever subsumee true)
    (dolist (u modemapList)
      (cond
        ((and (consp u)
              (progn
                (setq tmp6 (qfirst u))
                (and (consp tmp6) (progn (setq dc (qfirst tmp6)) t))))
          (progn
            (setq tmp7 (qrest u))
            (and (consp tmp7) (eq (qrest tmp7) nil)
              (progn
                (setq tmp1 (qfirst tmp7))
                (and (consp tmp1)
                  (progn
                    (setq cond (qfirst tmp1))
                    (setq tmp2 (qrest tmp1))
                    (and (consp tmp2) (eq (qrest tmp2) nil)
                      (progn
                        (setq tmp3 (qfirst tmp2))
                        (and (consp tmp3) (eq (qfirst tmp3) '|Subsumed|)
                          (progn
                            (setq tmp4 (qrest tmp3))

```

```

                                (and (consp tmp4)
                                    (progn
                                        (setq tmp5 (qrest tmp4))
                                        (and (consp tmp5)
                                            (eq (qrest tmp5) nil)
                                            (progn
                                                (setq nsig (qfirst tmp5))
                                                t)))))))))
                                (setq v (|assoc| (cons dc nsig) modemapList))
                                (consp v)
                                (progn
                                    (setq tmp6 (qrest v))
                                    (and (consp tmp6) (eq (qrest tmp6) nil)
                                        (progn
                                            (setq tmp7 (qfirst tmp6))
                                            (and (consp tmp7)
                                                (progn
                                                    (setq ncond (qfirst tmp7))
                                                    t))))))
                                (setq deleteList (cons u deleteList))
                                (unless (|PredImplies| ncond cond)
                                    (setq newList (push '(. (car u) (.cond (elt ,dc nil))) newList))))))
                                (when deleteList
                                    (setq modemapList
                                        (remove-if #'(lambda (x) (member x deleteList)) modemapList)))
                                ; it is important that subsumed ops (newList) be considered last
                                (when newList (setq modemapList (append modemapList newList)))
                                (setq t1
                                    (loop for x in arg1
                                        while (and (|isSimple| x)
                                                    (setq td (|compUniquely| x |$EmptyMode| env)))
                                        collect td
                                        do (setq env (third td))))
                                (cond
                                    ((some #'identity t1)
                                        (setq partialModeList (loop for x in t1 collect (when x (second x))))
                                        (or
                                            (|compFormPartiallyBottomUp| form mode env modemapList partialModeList)
                                            (|compForm3| form mode env modemapList)))
                                    (t (|compForm3| form mode env modemapList))))

```

11.1.40 defun compForm3

```

[compFormWithModemap p??]
[compUniquely p584]
[$compUniquelyIfTrue p??]

```

— defun compForm3 —

```

(defun |compForm3| (form mode env modemapList)
  (let (op argl mml tt)
    (declare (special |$compUniquelyIfTrue|))
    (setq op (car form))
    (setq argl (cdr form))
    (setq tt
      (let (result)
        (maplist #'(lambda (mlst)
          (setq result (or result
            (|compFormWithModemap| form mode env (car (setq mml mlst))))))
          modemapList)
        result))
    (when |$compUniquelyIfTrue|
      (if (let (result)
          (mapcar #'(lambda (mm)
            (setq result (or result (|compFormWithModemap| form mode env mm))))
            (rest mml))
          result)
        (throw '|compUniquely| nil)
        tt))
    tt))

```

11.1.41 defun compFocompFormWithModemap

```

[isCategoryForm p??]
[isFunctor p233]
[substituteIntoFunctorModemap p583]
[listOfSharpVars p??]
[coerceable p350]
[compApplyModemap p239]
[isCategoryForm p??]
[identp p??]
[get p??]
[last p??]
[convert p568]
[$Category p??]
[$FormalMapVariableList p250]

```

— defun compFormWithModemap —

```

(defun |compFormWithModemap| (form m env modemap)

```

```

(prog (op argl sv target cexpr targetp map temp1 f transimp sl mp formp z c
      xp ep tt)
(declare (special |$Category| |$FormalMapVariableList|))
(return
  (progn
    (setq op (car form))
    (setq argl (cdr form))
    (setq map (car modemap))
    (setq target (cadar modemap))
    (when (and (|isCategoryForm| target env) (|isFunction| op))
      (setq temp1 (or (|substituteIntoFunctorModemap| argl modemap env)
                      (return nil)))
      (setq modemap (car temp1))
      (setq env (cadr temp1))
      (setq map (car modemap))
      (setq target (cadar modemap))
      (setq cexpr (cdr modemap))
      modemap)
    (setq sv (|listOfSharpVars| map))
    (when sv
      (loop for x in argl for ss in |$FormalMapVariableList|
        do (when (|member| ss sv)
              (setq modemap (msubst x ss modemap))
              (setq map (car modemap))
              (setq target (cadar modemap))
              (setq cexpr (cdr modemap))
              modemap)))
    (cond
      ((null (setq targetp (|coerceable| target m env))) nil)
      (t
       (setq map (cons targetp (cdr map)))
       (setq temp1 (or (|compApplyModemap| form modemap env nil)
                       (return nil)))
       (setq f (car temp1))
       (setq transimp (cadr temp1))
       (setq sl (caddr temp1))
       (setq mp (sublis sl (elt map 1)))
       (setq xp
        (progn
          (setq formp (cons f (loop for tt in transimp collect (car tt))))
          (cond
            ((or (equal mp |$Category|) (|isCategoryForm| mp env)) formp)
            ((and (eq op '|elt|) (consp f) (eq (qcar f) '|xlam|)
                  (identp (car argl))
                  (setq c (|get| (car argl) '|condition| env))
                  (consp c) (eq (qcdr c) nil)
                  (consp (qcar c)) (eq (qcaar c) '|case|)
                  (consp (qcdr c)) (equal (qcadar c) z)
                  (consp (qcddar c)) (eq (qcdr (qcddar c)) nil)
                  (or (and (consp (qcaddar c))
                           (|isCategoryForm| qcaddar env)
                           (|isCategoryForm| qcaddar env))
                       (return nil)))))))

```

```

      (eq (qcar (qcaddar c)) '|:|)
      (consp (qcdr (qcaddar c)))
      (equal (qcadr (qcaddar c)) (cadr argl))
      (consp (qcddr (qcaddar c)))
      (eq (qcddr (qcaddar c)) nil)
      (equal (qcaddr (qcaddar c)) m))
    (eq (qcaddar c) (cadr argl))))
  (list 'cdr (car argl)))
  (t (cons '|call| formp))))))
(setq ep
  (if transimp
    (caddr (|last| transimp))
    env))
(setq tt (list xp mp ep))
(|convert| tt m))))))

```

11.1.42 defun substituteIntoFunctorModemap

```

[nequal p??]
[keyedSystemError p??]
[eqsubstlist p??]
[compOrCroak p556]
[sublis p??]

```

— defun substituteIntoFunctorModemap —

```

(defun |substituteIntoFunctorModemap| (argl modemap env)
  (let (dc sig tmp1 tl substitutionList)
    (setq dc (caar modemap))
    (setq sig (cdar modemap))
    (cond
      ((nequal (|#| dc) (|#| sig))
        (|keyedSystemError| 'S2GE0016
          (list "substituteIntoFunctorModemap" "Incompatible maps"))))
      ((equal (|#| argl) (|#| (cdr sig)))
        (setq sig (eqsubstlist argl (cdr dc) sig))
        (setq tl
          (loop for a in argl for m in (rest sig)
            collect (progn
              (setq tmp1 (|compOrCroak| a m env))
              (setq env (caddr tmp1))
              tmp1))))
        (setq substitutionList
          (loop for x in (rest dc) for tt in tl
            collect (cons x (car tt)))))

```

```
(list (sublis substitutionList modemap) env))
(t nil))))
```

11.1.43 defun compFormPartiallyBottomUp

```
[compForm3 p580]
[compFormMatch p584]
```

— defun compFormPartiallyBottomUp —

```
(defun |compFormPartiallyBottomUp| (form mode env modemapList partialModeList)
  (let (mmList)
    (when (setq mmList (loop for mm in modemapList
                            when (|compFormMatch| mm partialModeList)
                            collect mm))
      (|compForm3| form mode env mmList))))
```

11.1.44 defun compFormMatch

— defun compFormMatch —

```
(defun |compFormMatch| (mm partialModeList)
  (labels (
    (ismatch (a b)
      (cond
        ((null b) t)
        ((null (car b)) (|compFormMatch,match| (cdr a) (cdr b)))
        ((and (equal (car a) (car b)) (ismatch (cdr a) (cdr b))))))
    (and (consp mm) (consp (qfirst mm)) (consp (qcddar mm))
      (ismatch (qcddar mm) partialModeList))))
```

11.1.45 defun compUniquely

```
[compUniquely p584]
[comp p557]
[$compUniquelyIfTrue p??]
```

— defun compUniquely —

```
(defun |compUniquely| (x m env)
  (let (|$compUniquelyIfTrue|)
    (declare (special |$compUniquelyIfTrue|))
    (setq |$compUniquelyIfTrue| t)
    (catch '|compUniquely| (|comp| x m env))))
```

11.1.46 defun compArgumentsAndTryAgain

```
[comp p557]
[compForm1 p571]
[$EmptyMode p131]
```

— defun compArgumentsAndTryAgain —

```
(defun |compArgumentsAndTryAgain| (form mode env)
  (let (arg1 tmp1 a tmp2 tmp3 u)
    (declare (special |$EmptyMode|))
    (setq arg1 (cdr form))
    (cond
      ((and (consp form) (eq (qfirst form) '|elt|))
        (progn
          (setq tmp1 (qrest form))
          (and (consp tmp1)
            (progn
              (setq a (qfirst tmp1))
              (setq tmp2 (qrest tmp1))
              (and (consp tmp2) (eq (qrest tmp2) nil)))))))
      (when (setq tmp3 (|comp| a |$EmptyMode| env))
        (setq env (third tmp3))
        (|compForm1| form mode env)))
    (t
      (setq u
        (dolist (x arg1)
          (setq tmp3 (or (|comp| x |$EmptyMode| env) (return '|failed|))))
        (setq env (third tmp3))
        tmp3))
      (unless (eq u '|failed|)
        (|compForm1| form mode env))))))
```

11.1.47 defun compWithMappingMode

```
[compWithMappingMode1 p586]
[$formalArgList p??]
```

— defun compWithMappingMode —

```
(defun |compWithMappingMode| (form mode oldE)
  (declare (special |$formalArgList|))
  (|compWithMappingMode1| form mode oldE |$formalArgList|))
```

—————

11.1.48 defun compWithMappingMode1

```
[isFunction p233]
[get p??]
[qcar p??]
[qcdr p??]
[extendsCategoryForm p??]
[compLambda p315]
[stackAndThrow p??]
[take p??]
[compMakeDeclaration p593]
[hasFormalMapVariable p592]
[comp p557]
[extractCodeAndConstructTriple p591]
[optimizeFunctionDef p208]
[comp-tran p??]
[freelist p594]
[$formalArgList p??]
[$killOptimizeIfTrue p??]
[$funname p??]
[$funnameTail p??]
[$QuickCode p??]
[$EmptyMode p131]
[$FormalMapVariableList p250]
[$CategoryFrame p??]
[$formatArgList p??]
```

— defun compWithMappingMode1 —

```
(defun |compWithMappingMode1| (form mode oldE |$formalArgList|)
  (declare (special |$formalArgList|))
  (prog (|$killOptimizeIfTrue| $funname $funnameTail mprime s1 tmp1 tmp2
```

```

tmp3 tmp4 tmp5 tmp6 target argModeList nx oldstyle ress v11 e t t
  u frees i scode locals body vec expandedFunction fname uu)
(declare (special |$killOptimizeIfTrue| $funname $funnameTail
              |$QuickCode| |$EmptyMode| |$FormalMapVariableList|
              |$CategoryFrame| |$formatArgList|))

(return
  (seq
    (progn
      (setq mprime (second mode))
      (setq sl (cddr mode))
      (setq |$killOptimizeIfTrue| t)
      (setq e oldE)
      (cond
        ((|isFunctor| form)
          (cond
            ((and (progn
                    (setq tmp1 (|get| form ' |modemap| |$CategoryFrame|))
                    (and (consp tmp1)
                        (progn
                          (setq tmp2 (qfirst tmp1))
                          (and (consp tmp2)
                              (progn
                                (setq tmp3 (qfirst tmp2))
                                (and (consp tmp3)
                                    (progn
                                      (setq tmp4 (qrest tmp3))
                                      (and (consp tmp4)
                                          (progn
                                            (setq target (qfirst tmp4))
                                            (setq argModeList (qrest tmp4))
                                            t))))))
                                (progn
                                  (setq tmp5 (qrest tmp2))
                                  (and (consp tmp5) (eq (qrest tmp5) nil))))))))
              (progn
                (setq t1 t)
                (return
                  (do ((t2 nil (null t1))
                      (t3 argModeList (cdr t3))
                      (newmode nil)
                      (t4 sl (cdr t4))
                      (s nil))
                    ((or t2 (atom t3)
                        (progn (setq newmode (car t3)) nil)
                        (atom t4)
                        (progn (setq s (car t4)) nil))
                     t1)
                  (seq (exit
                        (setq t1
                          (and t1 (|extendsCategoryForm| '$ s newmode)))))))

```

```

(|extendsCategoryForm| '$ target mprime))
  (return (list form mode e )))
(t nil)))
(t
  (when (stringp form) (setq form (intern form)))
  (setq ress nil)
  (setq oldstyle t)
  (cond
    ((and (consp form)
      (eq (qfirst form) '+->)
      (progn
        (setq tmp1 (qrest form))
        (and (consp tmp1)
          (progn
            (setq v1 (qfirst tmp1))
            (setq tmp2 (qrest tmp1))
            (and (consp tmp2)
              (eq (qrest tmp2) nil)
              (progn (setq nx (qfirst tmp2)) t))))))
      (setq oldstyle nil)
      (cond
        ((and (consp v1) (eq (qfirst v1) '|:|))
          (setq ress (|compLambda| form mode oldE)
            ress)
          (t
            (setq v1
              (cond
                ((and (consp v1)
                  (eq (qfirst v1) '|@Tuple|)
                  (progn (setq v11 (qrest v1)) t))
                  v11)
                (t v1))))
            (setq v1
              (cond
                ((symbolp v1) (cons v1 nil))
                ((and
                  (listp v1)
                  (prog (t5)
                    (setq t5 t)
                    (return
                      (do ((t7 nil (null t5))
                        (t6 v1 (cdr t6))
                        (v nil))
                        ((or t7 (atom t6) (progn (setq v (car t6)) nil)) t5)
                      (seq
                        (exit
                          (setq t5 (and t5 (symbolp v))))))))))
                  v1)
                (t
                  (|stackAndThrow| (cons '|bad +-> arguments:| (list v1 ))))))))

```

```

      (setq |$formatArgList| (append vl |$formalArgList|))
      (setq form nx))))
  (t
   (setq vl (take (|#| sl) |$FormalMapVariableList|))))
  (cond
   (ress ress)
   (t
    (do ((t8 sl (cdr t8)) (m nil) (t9 vl (cdr t9)) (v nil))
        ((or (atom t8)
              (progn (setq m (car t8)) nil)
                     (atom t9)
                     (progn (setq v (car t9)) nil))
             nil)
         (seq (exit (progn
                     (setq tmp6
                          (|compMakeDeclaration| (list '|:| v m ) |$EmptyMode| e))
                     (setq e (third tmp6))
                     tmp6))))))
    (cond
     ((and oldstyle
            (null (null vl))
            (null (|hasFormalMapVariable| form vl))))
     (return
      (progn
       (setq tmp6 (or (|comp| (cons form vl) mprime e) (return nil)))
       (setq u (car tmp6))
       (|extractCodeAndConstructTriple| u mode oldE))))
     ((and (null vl) (setq tt (|comp| (cons form nil) mprime e)))
      (return
       (progn
        (setq u (car tt))
        (|extractCodeAndConstructTriple| u mode oldE))))
     (t
      (setq tmp6 (or (|comp| form mprime e) (return nil)))
      (setq u (car tmp6))
      (setq uu (|optimizeFunctionDef| '(nil (lambda ,vl ,u))))
      ; -- At this point, we have a function that we would like to pass.
      ; -- Unfortunately, it makes various free variable references outside
      ; -- itself. So we build a mini-vector that contains them all, and
      ; -- pass this as the environment to our inner function.
      (setq $funname nil)
      (setq $funnameTail (list nil))
      (setq expandedFunction (comp-tran (second uu)))
      (setq frees (freelist expandedFunction vl nil e))
      (setq expandedFunction
       (cond
        ((eql (|#| frees) 0)
         (cons 'lambda (cons (append vl (list '$$))
                               (cddr expandedFunction))))
        ((eql (|#| frees) 1)

```



```

        nil))))
      (t
        (setq body
          (cons
            (cons 'prog
              (cons locals
                (append scode
                  (cons
                    (cons 'return
                      (cons
                        (cons 'progn body)
                        nil))
                    nil))))
            nil))))
        nil))))))
      (setq vec (cons 'vector (nreverse vec)))
      (cons 'lambda (cons (append vl (list '$$) body))))))
(setq fname (list 'closedfn expandedFunction))
(setq uu
  (cond
    (frees (list 'cons fname vec))
    (t (list 'list fname))))
(list uu mode oldE)))))))))

```

11.1.49 defun extractCodeAndConstructTriple

— defun extractCodeAndConstructTriple —

```

(defun |extractCodeAndConstructTriple| (form mode oldE)
  (let (tmp1 a fn op env)
    (cond
      ((and (consp form) (eq (qfirst form) '|call|))
        (progn
          (setq tmp1 (qrest form))
          (and (consp tmp1)
            (progn (setq fn (qfirst tmp1)) t))))
      (cond
        ((and (consp fn) (eq (qfirst fn) '|applyFun|))
          (progn
            (setq tmp1 (qrest fn))
            (and (consp tmp1) (eq (qrest tmp1) nil)
              (progn (setq a (qfirst tmp1)) t))))
        (setq fn a)))
      (list fn mode oldE))
    (t
      (setq op (car form))

```

```
(setq env (car (reverse (cdr form))))
(list (list 'cons (list '|function| op) env) mode oldE))))
```

11.1.50 defun hasFormalMapVariable

```
[hasFormalMapVariable ScanOrPairVec (vol5)]
[$formalMapVariables p??]
```

— defun hasFormalMapVariable —

```
(defun |hasFormalMapVariable| (x vl)
  (let (|$formalMapVariables|)
    (declare (special |$formalMapVariables|))
    (when (setq |$formalMapVariables| vl)
      (|ScanOrPairVec| #'(lambda (y) (member y |$formalMapVariables|) x))))
```

11.1.51 defun argsToSig

— defun argsToSig —

```
(defun |argsToSig| (args)
  (let (tmp1 v tmp2 tt sig1 arg1 bad)
    (cond
      ((and (consp args) (eq (qfirst args) '|:|)
        (progn
          (setq tmp1 (qrest args))
          (and (consp tmp1)
            (progn
              (setq v (qfirst tmp1))
              (setq tmp2 (qrest tmp1))
              (and (consp tmp2)
                (eq (qrest tmp2) nil)
                (progn
                  (setq tt (qfirst tmp2))
                  t))))))
        (list (list v) (list tt)))
      (t
        (setq sig1 nil)
        (setq arg1 nil)
        (setq bad nil)
```

```

(dolist (arg args)
  (cond
    ((and (consp arg) (eq (qfirst arg) '|:|))
      (progn
        (setq tmp1 (qrest arg))
        (and (consp tmp1)
          (progn
            (setq v (qfirst tmp1))
            (setq tmp2 (qrest tmp1))
            (and (consp tmp2) (eq (qrest tmp2) nil)
              (progn
                (setq tt (qfirst tmp2))
                t))))))
      (setq sig1 (cons tt sig1))
      (setq arg1 (cons v arg1)))
    (t (setq bad t))))
  (cond
    (bad (list nil nil ))
    (t (list (reverse arg1) (reverse sig1))))))

```

11.1.52 defun compMakeDeclaration

[compColon p275]
 [\$insideExpressionIfTrue p??]

— defun compMakeDeclaration —

```

(defun |compMakeDeclaration| (form mode env)
  (let (|$insideExpressionIfTrue|)
    (declare (special |$insideExpressionIfTrue|))
    (setq |$insideExpressionIfTrue| nil)
    (|compColon| form mode env)))

```

11.1.53 defun modifyModeStack

[say p??]
 [copy p??]
 [setelt p??]
 [resolve p356]
 [\$reportExitModeStack p??]
 [\$exitModeStack p??]

— defun modifyModeStack —

```

(defun |modifyModeStack| (m index)
  (declare (special |$exitModeStack| |$reportExitModeStack|))
  (if |$reportExitModeStack|
    (say "exitModeStack: " (copy |$exitModeStack|)
      " ==> "
      (progn
        (setelt |$exitModeStack| index
          (|resolve| m (elt |$exitModeStack| index)))
        |$exitModeStack|))
    (setelt |$exitModeStack| index
      (|resolve| m (elt |$exitModeStack| index)))))

```

11.1.54 defun Create a list of unbound symbols

We walk argument *u* looking for symbols that are unbound. If we find a symbol we add it to the free list. If it occurs in a prog then it is bound and we remove it from the free list. Multiple instances of a single symbol in the free list are represented by the alist (symbol . count) [freelist p594]

```

[freelist assq (vol5)]
[freelist identp (vol5)]
[getmode p??]
[unionq p??]

```

— defun freelist —

```

(defun freelist (u bound free e)
  (let (v op)
    (if (atom u)
      (cond
        ((null (identp u)) free)
        ((member u bound) free)
        ; more than 1 free becomes alist (name . number)
        ((setq v (assq u free)) (rplacd v (+ 1 (cdr v))) free)
        ((null (|getmode| u e)) free)
        (t (cons (cons u 1) free)))
      (progn
        (setq op (car u))
        (cond
          ((member op '(quote go |function|)) free)
          ((eq op 'lambda) ; lambdas bind symbols
            (setq bound (unionq bound (second u)))
            (dolist (v (cddr u))

```

```

      (setq free (freelist v bound free e))))
    ((eq op 'prog) ; progs bind symbols
      (setq bound (unionq bound (second u)))
      (dolist (v (cddr u))
        (unless (atom v)
          (setq free (freelist v bound free e)))))
    ((eq op 'seq)
      (dolist (v (cdr u))
        (unless (atom v)
          (setq free (freelist v bound free e)))))
    ((eq op 'cond)
      (dolist (v (cdr u))
        (dolist (vv v)
          (setq free (freelist vv bound free e)))))
    (t
      (when (atom op) (setq u (cdr u))) ; atomic functions aren't descended
      (dolist (v u)
        (setq free (freelist v bound free e)))))
    free))))

```

11.1.55 defun compOrCroak1,compactify

[compOrCroak1,compactify p595]
 [lassoc p??]

— defun compOrCroak1,compactify —

```

(defun |compOrCroak1,compactify| (al)
  (cond
    ((null al) nil)
    ((lassoc (caar al) (cdr al)) (|compOrCroak1,compactify| (cdr al)))
    (t (cons (car al) (|compOrCroak1,compactify| (cdr al))))))

```

11.1.56 defun Compiler/Interpreter interface

[ncINTERPFILE SpadInterpretStream (vol5)]
 [\$EchoLines p??]
 [\$ReadingFile p??]

— defun ncINTERPFILE —

```
(defun |ncINTERPFILE| (file echo)
  (let ((|$EchoLines| echo) (|$ReadingFile| t))
    (declare (special |$EchoLines| |$ReadingFile|))
    (|SpadInterpretStream| 1 file nil)))
```

11.1.57 defun compileSpadLispCmd

```
[compileSpadLispCmd pathname (vol5)]
[compileSpadLispCmd pathnameType (vol5)]
[compileSpadLispCmd selectOptionLC (vol5)]
[compileSpadLispCmd namestring (vol5)]
[compileSpadLispCmd terminateSystemCommand (vol5)]
[compileSpadLispCmd fnameMake (vol5)]
[compileSpadLispCmd pathnameDirectory (vol5)]
[compileSpadLispCmd pathnameName (vol5)]
[compileSpadLispCmd fnameReadable? (vol5)]
[compileSpadLispCmd localdatabase (vol5)]
[throwKeyedMsg p??]
[object2String p??]
[compileSpadLispCmd sayKeyedMsg (vol5)]
[recompile-lib-file-if-necessary p597]
[spadPrompt p??]
[$options p??]
```

— defun compileSpadLispCmd —

```
(defun |compileSpadLispCmd| (args)
  (let (path optlist optname optargs beQuiet dolibrary lsp)
    (declare (special |$options|))
    (setq path (|pathname| (|fnameMake| (car args) "code" "lsp")))
    (cond
      ((null (probe-file path))
        (|throwKeyedMsg| 's2il0003 (cons (|namestring| args) nil)))
      (t
        (setq optlist '(|quiet| |noquiet| |library| |nolibrary|))
        (setq beQuiet nil)
        (setq dolibrary t)
        (dolist (opt |$options|)
          (setq optname (car opt))
          (setq optargs (cdr opt))
          (case (|selectOptionLC| optname optlist nil)
            (|quiet| (setq beQuiet t))
            (|noquiet| (setq beQuiet nil))
            (|library| (setq dolibrary t))
```

```

(|nolibrary| (setq dolibrary nil))
(t
  (|throwKeyedMsg| 's2iz0036
    (list (strconc ") " (|object2String| optname))))))
(setq lsp
  (|fnameMake|
    (|pathnameDirectory| path)
    (|pathnameName| path)
    (|pathnameType| path)))
(cond
  ((|fnameReadable?| lsp)
    (unless beQuiet (|sayKeyedMsg| 's2iz0089 (list (|namestring| lsp))))
    (recompile-lib-file-if-necessary lsp))
  (t
    (|sayKeyedMsg| 's2il0003 (list (|namestring| lsp))))))
(cond
  (dolibrary
    (unless beQuiet (|sayKeyedMsg| 's2iz0090 (list (|pathnameName| path))))
    (localdatabase (list (|pathnameName| (car args))) nil))
  ((null beQuiet) (|sayKeyedMsg| 's2iz0084 nil))
  (t nil))
(|terminateSystemCommand|)
(|spadPrompt|))))

```

11.1.58 defun recompile-lib-file-if-necessary

[compile-lib-file p598]
 [*lisp-bin-filetype* p??]

— defun recompile-lib-file-if-necessary —

```

(defun recompile-lib-file-if-necessary (lfile)
  (let* ((bfile (make-pathname :type *lisp-bin-filetype* :defaults lfile))
        (bdate (and (probe-file bfile) (file-write-date bfile)))
        (ldate (and (probe-file lfile) (file-write-date lfile))))
    (declare (special *lisp-bin-filetype*))
    (unless (and ldate bdate (> bdate ldate))
      (compile-lib-file lfile)
      (list bfile))))

```

11.1.59 defun spad-fixed-arg

— defun spad-fixed-arg —

```
(defun spad-fixed-arg (fname )
  (and (equal (symbol-package fname) (find-package "BOOT"))
        (not (get fname 'compiler::spad-var-arg))
        (search ";" (symbol-name fname))
        (or (get fname 'compiler::fixed-args)
              (setf (get fname 'compiler::fixed-args) t)))
  nil)
```

11.1.60 defun compile-lib-file

— defun compile-lib-file —

```
(defun compile-lib-file (fn &rest opts)
  (unwind-protect
    (progn
      (trace (compiler::fast-link-proclaimed-type-p
               :exitcond nil
               :entrycond (spad-fixed-arg (car system::arglist))))
      (trace (compiler::t1defun
               :exitcond nil
               :entrycond (spad-fixed-arg (caar system::arglist))))
      (apply #'compile-file fn opts))
    (untrace compiler::fast-link-proclaimed-type-p compiler::t1defun)))
```

11.1.61 defun compileFileQuietly

if `$InteractiveMode` then use a null outputstream [`$InteractiveMode p??`]
`[*standard-output* p??]`

— defun compileFileQuietly —

```
(defun |compileFileQuietly| (fn)
  (let (
        (*standard-output*
         (if |$InteractiveMode| (make-broadcast-stream)
```

```

      *standard-output*))
(declare (special *standard-output* |$InteractiveMode|))
(compile-file fn))

```

11.1.62 defvar \$byConstructors

— initvars —

```
(defvar |$byConstructors| () "list of constructors to be compiled")
```

11.1.63 defvar \$constructorsSeen

— initvars —

```
(defvar |$constructorsSeen| () "list of constructors found")
```

Chapter 12

Level 1

12.0.64 `defvar $current-fragment`

A string containing remaining chars from `readline`; needed because Symbolics `read-line` returns embedded newlines in a c-m-Y.

— **initvars** —

```
(defvar current-fragment nil)
```

—————

12.0.65 `defun read-a-line`

```
[subseq p??]  
[Line-New-Line p??]  
[read-a-line p601]  
[*eof* p??]  
[File-Closed p??]
```

— **defun read-a-line** —

```
(defun read-a-line (&optional (stream t))  
  (let (cp)  
    (declare (special *eof* File-Closed))  
    (if (and Current-Fragment (> (length Current-Fragment) 0))  
        (let ((line (with-input-from-string  
                      (s Current-Fragment :index cp :start 0)  
                      (read-line s nil nil))))  
          (setq Current-Fragment (subseq Current-Fragment cp))  
          line)  
        (read-line stream))
```

```
(prog nil
  (when (stream-eof in-stream)
    (setq File-Closed t)
    (setq *eof* t)
    (Line-New-Line (make-string 0) Current-Line)
    (return nil))
  (when (setq Current-Fragment (read-line stream))
    (return (read-a-line stream))))))
```

Chapter 13

Level 0

13.1 Line Handling

13.1.1 Line Buffer

The philosophy of lines is that

- NEXT LINE will always return a non-blank line or fail.
- Every line is terminated by a blank character.

Hence there is always a current character, because there is never a non-blank line, and there is always a separator character between tokens on separate lines. Also, when a line is read, the character pointer is always positioned ON the first character.

13.1.2 defstruct \$line

— initvars —

```
(defstruct line "Line of input file to parse."  
  (buffer (make-string 0) :type string)  
  (current-char #\Return :type character)  
  (current-index 1 :type fixnum)  
  (last-index 0 :type fixnum)  
  (number 0 :type fixnum))
```

—————

13.1.3 defvar \$current-line

The current input line.

— **initvars** —

```
(defvar current-line (make-line))
```

13.1.4 defmacro line-clear

[*\$line* p603]

— **defmacro line-clear** —

```
(defmacro line-clear (line)
  '(let ((l ,line))
    (setf (line-buffer l) (make-string 0))
    (setf (line-current-char l) #\return)
    (setf (line-current-index l) 1)
    (setf (line-last-index l) 0)
    (setf (line-number l) 0)))
```

13.1.5 defun line-print

[*\$line* p603]

[*\$out-stream* p??]

— **defun line-print** —

```
(defun line-print (line)
  (declare (special out-stream))
  (format out-stream "~&~5D> ~A~%" (Line-Number line) (Line-Buffer line))
  (format out-stream "~v@T~%" (+ 7 (Line-Current-Index line))))
```

13.1.6 defun line-at-end-p

[*\$line* p603]

— **defun line-at-end-p** —

```
(defun line-at-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (>= (line-current-index line) (line-last-index line)))
```

13.1.7 defun line-past-end-p

[*\$line* p603]

— **defun line-past-end-p** —

```
(defun line-past-end-p (line)
  "Tests if line is empty or positioned past the last character."
  (> (line-current-index line) (line-last-index line)))
```

13.1.8 defun line-next-char

[*\$line* p603]

— **defun line-next-char** —

```
(defun line-next-char (line)
  (elt (line-buffer line) (1+ (line-current-index line))))
```

13.1.9 defun line-advance-char

[*\$line* p603]

— **defun line-advance-char** —

```
(defun line-advance-char (line)
  (setf (line-current-char line)
        (elt (line-buffer line) (incf (line-current-index line)))))
```

13.1.10 defun line-current-segment

[\$line p603]

— defun line-current-segment —

```
(defun line-current-segment (line)
  "Buffer from current index to last index."
  (if (line-at-end-p line)
      (make-string 0)
      (subseq (line-buffer line)
               (line-current-index line)
               (line-last-index line))))
```

—————

13.1.11 defun line-new-line

[\$line p603]

— defun line-new-line —

```
(defun line-new-line (string line &optional (linenum nil))
  "Sets string to be the next line stored in line."
  (setf (line-last-index line) (1- (length string)))
  (setf (line-current-index line) 0)
  (setf (line-current-char line)
        (or (and (> (length string) 0) (elt string 0)) #\Return))
  (setf (line-buffer line) string)
  (setf (line-number line) (or linenum (1+ (line-number line)))))
```

—————

13.1.12 defun next-line

[\$in-stream p??]

[\$line-handler p??]

— defun next-line —

```
(defun next-line (&optional (in-stream t))
  (declare (special in-stream line-handler))
  (funcall Line-Handler in-stream))
```

—————

13.1.13 defun Advance-Char

[Line-At-End-P p??]
 [Line-Advance-Char p??]
 [next-line p606]
 [current-char p457]
 [\$in-stream p??]
 [\$line p603]

— defun Advance-Char —

```
(defun Advance-Char ()
  "Advances IN-STREAM, invoking Next Line if necessary."
  (declare (special in-stream))
  (loop
    (cond
      ((not (Line-At-End-P Current-Line))
       (return (Line-Advance-Char Current-Line)))
      ((next-line in-stream)
       (return (current-char)))
      ((return nil)))))
```

—————

13.1.14 defun storeblanks

— defun storeblanks —

```
(defun storeblanks (line n)
  (do ((i 0 (1+ i)))
      ((= i n) line)
    (setf (char line i) #\ )))
```

—————

13.1.15 defun initial-substring

[mismatch p??]

— defun initial-substring —

```
(defun initial-substring (pattern line)
  (let ((ind (mismatch pattern line)))
```

```
(or (null ind) (eq1 ind (size pattern))))))
```

13.1.16 defun get-a-line

```
[is-console p527]
[get-a-line mkprompt (vol5)]
[read-a-line p601]
```

— defun get-a-line —

```
(defun get-a-line (stream)
  (when (is-console stream) (princ (mkprompt)))
  (let ((l1 (read-a-line stream)))
    (if (and (stringp l1) (adjustable-array-p l1))
        (make-array (array-dimensions l1) :element-type 'string-char
                     :adjustable t :initial-contents l1)
        l1)))
```

Chapter 14

The Chunks

— Compiler —

```
(in-package "BOOT")

\getchunk{initvars}

\getchunk{LEDNUDTables}
\getchunk{GLIPHTable}
\getchunk{RENAME TOKTable}
\getchunk{GENERICTable}

\getchunk{defmacro bang}
\getchunk{defmacro line-clear}
\getchunk{defmacro must}
\getchunk{defmacro nth-stack}
\getchunk{defmacro pop-stack-1}
\getchunk{defmacro pop-stack-2}
\getchunk{defmacro pop-stack-3}
\getchunk{defmacro pop-stack-4}
\getchunk{defmacro reduce-stack-clear}
\getchunk{defmacro stack-/empty}
\getchunk{defmacro star}

\getchunk{defun action}
\getchunk{defun addArgumentConditions}
\getchunk{defun addclose}
\getchunk{defun addConstructorModemaps}
\getchunk{defun addDomain}
\getchunk{defun addEltModemap}
\getchunk{defun addEmptyCapsuleIfNecessary}
\getchunk{defun addModemapKnown}
```

```

\getchunk{defun addModemap}
\getchunk{defun addModemap0}
\getchunk{defun addModemap1}
\getchunk{defun addNewDomain}
\getchunk{defun add-parens-and-semis-to-line}
\getchunk{defun addSuffix}
\getchunk{defun Advance-Char}
\getchunk{defun advance-token}
\getchunk{defun alistSize}
\getchunk{defun allLASSOCs}
\getchunk{defun aplTran}
\getchunk{defun aplTran1}
\getchunk{defun aplTranList}
\getchunk{defun applyMapping}
\getchunk{defun argsToSig}
\getchunk{defun assignError}
\getchunk{defun AssocBarGensym}
\getchunk{defun augLisplibModemapsFromCategory}
\getchunk{defun augmentLisplibModemapsFromFunctor}
\getchunk{defun augModemapsFromCategory}
\getchunk{defun augModemapsFromCategoryRep}
\getchunk{defun augModemapsFromDomain}
\getchunk{defun augModemapsFromDomain1}
\getchunk{defun autoCoerceByModemap}

\getchunk{defun blankp}
\getchunk{defun bootStrapError}
\getchunk{defun bumperrorcount}

\getchunk{defun canReturn}
\getchunk{defun char-eq}
\getchunk{defun char-ne}
\getchunk{defun checkAddBackSlashes}
\getchunk{defun checkAddMacros}
\getchunk{defun checkAddPeriod}
\getchunk{defun checkAddSpaceSegments}
\getchunk{defun checkAlphabetic}
\getchunk{defun checkAndDeclare}
\getchunk{defun checkArguments}
\getchunk{defun checkBalance}
\getchunk{defun checkBeginEnd}
\getchunk{defun checkComments}
\getchunk{defun checkDecorate}
\getchunk{defun checkDecorateForHt}
\getchunk{defun checkDocError}
\getchunk{defun checkDocError1}
\getchunk{defun checkDocMessage}
\getchunk{defun checkExtract}
\getchunk{defun checkFixCommonProblem}
\getchunk{defun checkGetArgs}

```

```

\getchunk{defun checkGetMargin}
\getchunk{defun checkGetParse}
\getchunk{defun checkHTargs}
\getchunk{defun checkIeEg}
\getchunk{defun checkIeEgfun}
\getchunk{defun checkIndentedLines}
\getchunk{defun checkLookForLeftBrace}
\getchunk{defun checkLookForRightBrace}
\getchunk{defun checkNumOfArgs}
\getchunk{defun checkTexht}
\getchunk{defun checkRecordHash}
\getchunk{defun checkRemoveComments}
\getchunk{defun checkRewrite}
\getchunk{defun checkSayBracket}
\getchunk{defun checkSkipBlanks}
\getchunk{defun checkSkipIdentifierToken}
\getchunk{defun checkSkipOpToken}
\getchunk{defun checkSkipToken}
\getchunk{defun checkAddSpaces}
\getchunk{defun checkSplitBackslash}
\getchunk{defun checkSplitBrace}
\getchunk{defun checkSplitOn}
\getchunk{defun checkSplitPunctuation}
\getchunk{defun checkSplit2Words}
\getchunk{defun checkTransformFirsts}
\getchunk{defun checkTrim}
\getchunk{defun checkTrimCommented}
\getchunk{defun checkWarning}
\getchunk{defun coerce}
\getchunk{defun coerceable}
\getchunk{defun coerceByModemap}
\getchunk{defun coerceEasy}
\getchunk{defun coerceExit}
\getchunk{defun coerceExtraHard}
\getchunk{defun coerceHard}
\getchunk{defun coerceSubset}
\getchunk{defun collectAndDeleteAssoc}
\getchunk{defun collectComBlock}
\getchunk{defun comma2Tuple}
\getchunk{defun comp}
\getchunk{defun comp2}
\getchunk{defun comp3}
\getchunk{defun compAdd}
\getchunk{defun compApplication}
\getchunk{defun compApply}
\getchunk{defun compApplyModemap}
\getchunk{defun compArgumentConditions}
\getchunk{defun compArgumentsAndTryAgain}
\getchunk{defun compAtom}
\getchunk{defun compAtomWithModemap}

```

```

\getchunk{defun compAtSign}
\getchunk{defun compBoolean}
\getchunk{defun compCapsule}
\getchunk{defun compCapsuleInner}
\getchunk{defun compCapsuleItems}
\getchunk{defun compCase}
\getchunk{defun compCase1}
\getchunk{defun compCat}
\getchunk{defun compCategory}
\getchunk{defun compCategoryItem}
\getchunk{defun compCoerce}
\getchunk{defun compCoerce1}
\getchunk{defun compColon}
\getchunk{defun compColonInside}
\getchunk{defun compCons}
\getchunk{defun compCons1}
\getchunk{defun compConstruct}
\getchunk{defun compConstructorCategory}
\getchunk{defun compDefine}
\getchunk{defun compDefine1}
\getchunk{defun compDefineAddSignature}
\getchunk{defun compDefineCapsuleFunction}
\getchunk{defun compDefineCategory}
\getchunk{defun compDefineCategory1}
\getchunk{defun compDefineCategory2}
\getchunk{defun compDefineFunctor}
\getchunk{defun compDefineFunctor1}
\getchunk{defun compDefineLisplib}
\getchunk{defun compDefWhereClause}
\getchunk{defun compElt}
\getchunk{defun compExit}
\getchunk{defun compExpression}
\getchunk{defun compExpressionList}
\getchunk{defun compForm}
\getchunk{defun compForm1}
\getchunk{defun compForm2}
\getchunk{defun compForm3}
\getchunk{defun compFormMatch}
\getchunk{defun compFormMode}
\getchunk{defun compFormPartiallyBottomUp}
\getchunk{defun compFormWithModemap}
\getchunk{defun compFromIf}
\getchunk{defun compFunctorBody}
\getchunk{defun compHas}
\getchunk{defun compHasFormat}
\getchunk{defun compIf}
\getchunk{defun compile}
\getchunk{defun compileCases}
\getchunk{defun compileConstructor}
\getchunk{defun compileConstructor1}

```

```

\getchunk{defun compileDocumentation}
\getchunk{defun compileFileQuietly}
\getchunk{defun compile-lib-file}
\getchunk{defun compiler}
\getchunk{defun compilerDoit}
\getchunk{defun compilerDoitWithScreenedLisplib}
\getchunk{defun compileSpad2Cmd}
\getchunk{defun compileSpadLispCmd}
\getchunk{defun compileTimeBindingOf}
\getchunk{defun compImport}
\getchunk{defun compInternalFunction}
\getchunk{defun compIs}
\getchunk{defun compJoin}
\getchunk{defun compLambda}
\getchunk{defun compLeave}
\getchunk{defun compList}
\getchunk{defun compMacro}
\getchunk{defun compMakeCategoryObject}
\getchunk{defun compMakeDeclaration}
\getchunk{defun compMapCond}
\getchunk{defun compMapCond'}
\getchunk{defun compMapCond''}
\getchunk{defun compMapCondFun}
\getchunk{defun compNoStacking}
\getchunk{defun compNoStacking1}
\getchunk{defun compOrCroak}
\getchunk{defun compOrCroak1}
\getchunk{defun compOrCroak1,compactify}
\getchunk{defun compPretend}
\getchunk{defun compQuote}
\getchunk{defun compRepeatOrCollect}
\getchunk{defun compReduce}
\getchunk{defun compReduce1}
\getchunk{defun compReturn}
\getchunk{defun compSeq}
\getchunk{defun compSeqItem}
\getchunk{defun compSeq1}
\getchunk{defun compSetq}
\getchunk{defun compSetq1}
\getchunk{defun compSingleCapsuleItem}
\getchunk{defun compString}
\getchunk{defun compSubDomain}
\getchunk{defun compSubDomain1}
\getchunk{defun compSymbol}
\getchunk{defun compSubsetCategory}
\getchunk{defun compSuchthat}
\getchunk{defun compToApply}
\getchunk{defun compTopLevel}
\getchunk{defun compTuple2Record}
\getchunk{defun compTypeOf}

```

```

\getchunk{defun compUniquely}
\getchunk{defun compVector}
\getchunk{defun compWhere}
\getchunk{defun compWithMappingMode}
\getchunk{defun compWithMappingModel}
\getchunk{defun constructMacro}
\getchunk{defun containsBang}
\getchunk{defun convert}
\getchunk{defun convertOpAlist2compilerInfo}
\getchunk{defun convertOrCroak}
\getchunk{defun current-char}
\getchunk{defun current-symbol}
\getchunk{defun current-token}

\getchunk{defun decodeScripts}
\getchunk{defun deepestExpression}
\getchunk{defun def-rename}
\getchunk{defun def-rename1}
\getchunk{defun disallowNilAttribute}
\getchunk{defun displayMissingFunctions}
\getchunk{defun displayPreCompilationErrors}
\getchunk{defun doIt}
\getchunk{defun doItIf}
\getchunk{defun dollarTran}
\getchunk{defun domainMember}
\getchunk{defun drop}

\getchunk{defun eltModemapFilter}
\getchunk{defun encodeItem}
\getchunk{defun encodeFunctionName}
\getchunk{defun EqualBarGensym}
\getchunk{defun errhuh}
\getchunk{defun escape-keywords}
\getchunk{defun escaped}
\getchunk{defun evalAndRwriteLispForm}
\getchunk{defun evalAndSub}
\getchunk{defun extractCodeAndConstructTriple}

\getchunk{defun flattenSignatureList}
\getchunk{defun finalizeDocumentation}
\getchunk{defun finalizeLisplib}
\getchunk{defun fincomblock}
\getchunk{defun firstNonBlankPosition}
\getchunk{defun fixUpPredicate}
\getchunk{defun floatexpid}
\getchunk{defun formal2Pattern}
\getchunk{defun freelist}

\getchunk{defun get-a-line}
\getchunk{defun getAbbreviation}

```

```

\getchunk{defun getArgumentMode}
\getchunk{defun getArgumentModeOrMoan}
\getchunk{defun getCaps}
\getchunk{defun getCategoryOpsAndAtts}
\getchunk{defun getConstructorOpsAndAtts}
\getchunk{defun getDomainsInScope}
\getchunk{defun getFormModemaps}
\getchunk{defun getFunctorOpsAndAtts}
\getchunk{defun getInverseEnvironment}
\getchunk{defun getMatchingRightPren}
\getchunk{defun getModemap}
\getchunk{defun getModemapList}
\getchunk{defun getModemapListFromDomain}
\getchunk{defun getOperationAlist}
\getchunk{defun getScriptName}
\getchunk{defun getSignature}
\getchunk{defun getSignatureFromMode}
\getchunk{defun getSlotFromCategoryForm}
\getchunk{defun getSlotFromFunctor}
\getchunk{defun getSpecialCaseAssoc}
\getchunk{defun getSuccessEnvironment}
\getchunk{defun getTargetFromRhs}
\getchunk{defun get-token}
\getchunk{defun getToken}
\getchunk{defun getUnionMode}
\getchunk{defun getUniqueModemap}
\getchunk{defun getUniqueSignature}
\getchunk{defun genDomainOps}
\getchunk{defun genDomainViewList0}
\getchunk{defun genDomainViewList}
\getchunk{defun genDomainView}
\getchunk{defun giveFormalParametersValues}

\getchunk{defun hackforis}
\getchunk{defun hackforis1}
\getchunk{defun hasAplExtension}
\getchunk{defun hasFormalMapVariable}
\getchunk{defun hasFullSignature}
\getchunk{defun hasNoVowels}
\getchunk{defun hasSigInTargetCategory}
\getchunk{defun hasType}
\getchunk{defun htcharPosition}

\getchunk{defun indent-pos}
\getchunk{defun infixtok}
\getchunk{defun initialize-preparse}
\getchunk{defun initial-substring}
\getchunk{defun initial-substring-p}
\getchunk{defun initializeLisplib}
\getchunk{defun insertModemap}

```

```

\getchunk{defun interactiveModemapForm}
\getchunk{defun isCategoryPackageName}
\getchunk{defun is-console}
\getchunk{defun isDomainConstructorForm}
\getchunk{defun isDomainForm}
\getchunk{defun isDomainSubst}
\getchunk{defun isFunctor}
\getchunk{defun isListConstructor}
\getchunk{defun isMacro}
\getchunk{defun isSuperDomain}
\getchunk{defun isTokenDelimiter}
\getchunk{defun isUnionMode}

\getchunk{defun killColons}

\getchunk{defun line-advance-char}
\getchunk{defun line-at-end-p}
\getchunk{defun line-current-segment}
\getchunk{defun line-next-char}
\getchunk{defun line-past-end-p}
\getchunk{defun line-print}
\getchunk{defun line-new-line}
\getchunk{defun lispize}
\getchunk{defun lisplibDoRename}
\getchunk{defun lisplibWrite}
\getchunk{defun loadIfNecessary}
\getchunk{defun loadLibIfNecessary}

\getchunk{defun macroExpand}
\getchunk{defun macroExpandInPlace}
\getchunk{defun macroExpandList}
\getchunk{defun makeCategoryForm}
\getchunk{defun makeCategoryPredicates}
\getchunk{defun makeFunctorArgumentParameters}
\getchunk{defun makeSimplePredicateOrNil}
\getchunk{defun make-symbol-of}
\getchunk{defun match-advance-string}
\getchunk{defun match-current-token}
\getchunk{defun match-next-token}
\getchunk{defun match-string}
\getchunk{defun match-token}
\getchunk{defun maxSuperType}
\getchunk{defun mergeModemap}
\getchunk{defun mergeSignatureAndLocalVarAlists}
\getchunk{defun meta-syntax-error}
\getchunk{defun mkAbbrev}
\getchunk{defun mkAlistOfExplicitCategoryOps}
\getchunk{defun mkCategoryPackage}
\getchunk{defun mkConstructor}
\getchunk{defun mkDatabasePred}

```

```

\getchunk{defun mkEvalableCategoryForm}
\getchunk{defun mkExplicitCategoryFunction}
\getchunk{defun mkList}
\getchunk{defun mkNewModemapList}
\getchunk{defun mkOpVec}
\getchunk{defun mkRepetitionAssoc}
\getchunk{defun mkUnion}
\getchunk{defun modifyModeStack}
\getchunk{defun modeEqual}
\getchunk{defun modeEqualSubst}
\getchunk{defun modemapPattern}
\getchunk{defun moveORsOutside}
\getchunk{defun mustInstantiate}

```

```

\getchunk{defun ncINTERPFILE}
\getchunk{defun newWordFrom}
\getchunk{defun next-char}
\getchunk{defun next-line}
\getchunk{defun next-tab-loc}
\getchunk{defun next-token}
\getchunk{defun newString2Words}
\getchunk{defun new2OldLisp}
\getchunk{defun nonblankloc}
\getchunk{defun NRTassocIndex}
\getchunk{defun NRTgetLocalIndex}
\getchunk{defun NRTgetLookupFunction}
\getchunk{defun NRTputInHead}
\getchunk{defun NRTputInTail}

```

```

\getchunk{defun optCall}
\getchunk{defun optCallEval}
\getchunk{defun optCallSpecially}
\getchunk{defun optCatch}
\getchunk{defun optCond}
\getchunk{defun optCONDtail}
\getchunk{defun optEQ}
\getchunk{defun optIF2COND}
\getchunk{defun optimize}
\getchunk{defun optimizeFunctionDef}
\getchunk{defun optional}
\getchunk{defun optLESSP}
\getchunk{defun optMINUS}
\getchunk{defun optMkRecord}
\getchunk{defun optPackageCall}
\getchunk{defun optPredicateIfTrue}
\getchunk{defun optQSMINUS}
\getchunk{defun optRECORDCOPY}
\getchunk{defun optRECORDELT}
\getchunk{defun optSETRECORDELT}
\getchunk{defun optSEQ}

```

```

\getchunk{defun optSPADCALL}
\getchunk{defun optSpecialCall}
\getchunk{defun optSuchthat}
\getchunk{defun optXLAMCond}
\getchunk{defun opt-}
\getchunk{defun orderByDependency}
\getchunk{defun orderPredicateItems}
\getchunk{defun orderPredTran}
\getchunk{defun outputComp}

\getchunk{defun PARSE-AnyId}
\getchunk{defun PARSE-Application}
\getchunk{defun parse-argument-designator}
\getchunk{defun parse-identifier}
\getchunk{defun parse-keyword}
\getchunk{defun parse-number}
\getchunk{defun parse-spadstring}
\getchunk{defun parse-string}
\getchunk{defun PARSE-Category}
\getchunk{defun PARSE-Command}
\getchunk{defun PARSE-CommandTail}
\getchunk{defun PARSE-Conditional}
\getchunk{defun PARSE-Data}
\getchunk{defun PARSE-ElseClause}
\getchunk{defun PARSE-Enclosure}
\getchunk{defun PARSE-Exit}
\getchunk{defun PARSE-Expr}
\getchunk{defun PARSE-Expression}
\getchunk{defun PARSE-Float}
\getchunk{defun PARSE-FloatBase}
\getchunk{defun PARSE-FloatBasePart}
\getchunk{defun PARSE-FloatExponent}
\getchunk{defun PARSE-FloatTok}
\getchunk{defun PARSE-Form}
\getchunk{defun PARSE-FormalParameter}
\getchunk{defun PARSE-FormalParameterTok}
\getchunk{defun PARSE-getSemanticForm}
\getchunk{defun PARSE-GlyphTok}
\getchunk{defun PARSE-Import}
\getchunk{defun PARSE-Infix}
\getchunk{defun PARSE-InfixWith}
\getchunk{defun PARSE-IntegerTok}
\getchunk{defun PARSE-Iterator}
\getchunk{defun PARSE-IteratorTail}
\getchunk{defun PARSE-Label}
\getchunk{defun PARSE-LabelExpr}
\getchunk{defun PARSE-Leave}
\getchunk{defun PARSE-LedPart}
\getchunk{defun PARSE-leftBindingPowerOf}
\getchunk{defun PARSE-Loop}

```

```

\getchunk{defun PARSE-Name}
\getchunk{defun PARSE-NBGlyphTok}
\getchunk{defun PARSE-NewExpr}
\getchunk{defun PARSE-NudPart}
\getchunk{defun PARSE-OpenBrace}
\getchunk{defun PARSE-OpenBracket}
\getchunk{defun PARSE-Operation}
\getchunk{defun PARSE-Option}
\getchunk{defun PARSE-Prefix}
\getchunk{defun PARSE-Primary}
\getchunk{defun PARSE-Primary1}
\getchunk{defun PARSE-PrimaryNoFloat}
\getchunk{defun PARSE-PrimaryOrQM}
\getchunk{defun PARSE-Qualification}
\getchunk{defun PARSE-Quad}
\getchunk{defun PARSE-Reduction}
\getchunk{defun PARSE-ReductionOp}
\getchunk{defun PARSE-Return}
\getchunk{defun PARSE-rightBindingPowerOf}
\getchunk{defun PARSE-ScriptItem}
\getchunk{defun PARSE-Scripts}
\getchunk{defun PARSE-Seg}
\getchunk{defun PARSE-Selector}
\getchunk{defun PARSE-SemiColon}
\getchunk{defun PARSE-Sequence}
\getchunk{defun PARSE-Sequence1}
\getchunk{defun PARSE-Sexpr}
\getchunk{defun PARSE-Sexpr1}
\getchunk{defun PARSE-SpecialCommand}
\getchunk{defun PARSE-SpecialKeyword}
\getchunk{defun PARSE-Statement}
\getchunk{defun PARSE-String}
\getchunk{defun PARSE-Suffix}
\getchunk{defun PARSE-TokenCommandTail}
\getchunk{defun PARSE-TokenList}
\getchunk{defun PARSE-TokenOption}
\getchunk{defun PARSE-TokTail}
\getchunk{defun PARSE-VarForm}
\getchunk{defun PARSE-With}
\getchunk{defun parsepiles}
\getchunk{defun parseAnd}
\getchunk{defun parseAtom}
\getchunk{defun parseAtSign}
\getchunk{defun parseCategory}
\getchunk{defun parseCoerce}
\getchunk{defun parseColon}
\getchunk{defun parseConstruct}
\getchunk{defun parseDEF}
\getchunk{defun parseDollarGreaterEqual}
\getchunk{defun parseDollarGreaterThan}

```

```

\getchunk{defun parseDollarLessEqual}
\getchunk{defun parseDollarNotEqual}
\getchunk{defun parseDropAssertions}
\getchunk{defun parseEquivalence}
\getchunk{defun parseExit}
\getchunk{defun postFlatten}
\getchunk{defun postFlattenLeft}
\getchunk{defun postForm}
\getchunk{defun parseGreaterEqual}
\getchunk{defun parseGreaterThan}
\getchunk{defun parseHas}
\getchunk{defun parseHasRhs}
\getchunk{defun parseIf}
\getchunk{defun parseIf,ifTran}
\getchunk{defun parseImplies}
\getchunk{defun parseIn}
\getchunk{defun parseInBy}
\getchunk{defun parseIs}
\getchunk{defun parseIsnt}
\getchunk{defun parseJoin}
\getchunk{defun parseLeave}
\getchunk{defun parseLessEqual}
\getchunk{defun parseLET}
\getchunk{defun parseLETD}
\getchunk{defun parseLhs}
\getchunk{defun parseMDEF}
\getchunk{defun parseNot}
\getchunk{defun parseNotEqual}
\getchunk{defun parseOr}
\getchunk{defun parsePretend}
\getchunk{defun parseprint}
\getchunk{defun parseReturn}
\getchunk{defun parseSegment}
\getchunk{defun parseSeq}
\getchunk{defun parseTran}
\getchunk{defun parseTranCheckForRecord}
\getchunk{defun parseTranList}
\getchunk{defun parseTransform}
\getchunk{defun parseType}
\getchunk{defun parseVCONS}
\getchunk{defun parseWhere}
\getchunk{defun Pop-Reduction}
\getchunk{defun postAdd}
\getchunk{defun postAtom}
\getchunk{defun postAtSign}
\getchunk{defun postBigFloat}
\getchunk{defun postBlock}
\getchunk{defun postBlockItem}
\getchunk{defun postBlockItemList}
\getchunk{defun postCapsule}

```

```

\getchunk{defun postCategory}
\getchunk{defun postcheck}
\getchunk{defun postCollect}
\getchunk{defun postCollect,finish}
\getchunk{defun postColon}
\getchunk{defun postColonColon}
\getchunk{defun postComma}
\getchunk{defun postConstruct}
\getchunk{defun postDef}
\getchunk{defun postDefArgs}
\getchunk{defun postError}
\getchunk{defun postExit}
\getchunk{defun postIf}
\getchunk{defun postIn}
\getchunk{defun postIn}
\getchunk{defun postInSeq}
\getchunk{defun postIteratorList}
\getchunk{defun postJoin}
\getchunk{defun postMakeCons}
\getchunk{defun postMapping}
\getchunk{defun postMDef}
\getchunk{defun postOp}
\getchunk{defun postPretend}
\getchunk{defun postQUOTE}
\getchunk{defun postReduce}
\getchunk{defun postRepeat}
\getchunk{defun postScripts}
\getchunk{defun postScriptsForm}
\getchunk{defun postSemiColon}
\getchunk{defun postSignature}
\getchunk{defun postSlash}
\getchunk{defun postTran}
\getchunk{defun postTranList}
\getchunk{defun postTranScripts}
\getchunk{defun postTranSegment}
\getchunk{defun postTransform}
\getchunk{defun postTransformCheck}
\getchunk{defun postTuple}
\getchunk{defun postTupleCollect}
\getchunk{defun postType}
\getchunk{defun postWhere}
\getchunk{defun postWith}
\getchunk{defun print-package}
\getchunk{defun preparse}
\getchunk{defun preparse1}
\getchunk{defun preparse-echo}
\getchunk{defun preparseReadLine}
\getchunk{defun preparseReadLine1}
\getchunk{defun primitiveType}
\getchunk{defun print-defun}

```

```

\getchunk{defun processFunctor}
\getchunk{defun push-reduction}
\getchunk{defun putDomainsInScope}
\getchunk{defun putInLocalDomainReferences}

\getchunk{defun quote-if-string}

\getchunk{defun read-a-line}
\getchunk{defun recompile-lib-file-if-necessary}
\getchunk{defun recordAttributeDocumentation}
\getchunk{defun recordDocumentation}
\getchunk{defun recordHeaderDocumentation}
\getchunk{defun recordSignatureDocumentation}
\getchunk{defun replaceExitEtc}
\getchunk{defun removeBackslashes}
\getchunk{defun removeSuperfluousMapping}
\getchunk{defun replaceVars}
\getchunk{defun resolve}
\getchunk{defun reportOnFunctorCompilation}
\getchunk{defun /rf-1}
\getchunk{defun /RQ,LIB}
\getchunk{defun rwriteLispForm}

\getchunk{defun setDefOp}
\getchunk{defun seteltModemapFilter}
\getchunk{defun setqMultiple}
\getchunk{defun setqMultipleExplicit}
\getchunk{defun setqSetelt}
\getchunk{defun setqSingle}
\getchunk{defun signatureTran}
\getchunk{defun skip-blanks}
\getchunk{defun skip-ifblock}
\getchunk{defun skip-to-endif}
\getchunk{defun spad}
\getchunk{defun spadCompileOrSetq}
\getchunk{defun spad-fixed-arg}
\getchunk{defun splitEncodedFunctionName}
\getchunk{defun stack-clear}
\getchunk{defun stack-load}
\getchunk{defun stack-pop}
\getchunk{defun stack-push}
\getchunk{defun storeblanks}
\getchunk{defun stripOffArgumentConditions}
\getchunk{defun stripOffSubdomainConditions}
\getchunk{defun subrname}
\getchunk{defun substituteCategoryArguments}
\getchunk{defun substituteIntoFunctorModemap}
\getchunk{defun substNames}
\getchunk{defun substVars}
\getchunk{defun s-process}

```

```

\getchunk{defun token-install}
\getchunk{defun token-lookahead-type}
\getchunk{defun token-print}
\getchunk{defun transDoc}
\getchunk{defun transDocList}
\getchunk{defun transformAndRecheckComments}
\getchunk{defun transformOperationAlist}
\getchunk{defun transImplementation}
\getchunk{defun transIs}
\getchunk{defun transIs1}
\getchunk{defun translablel}
\getchunk{defun translablel1}
\getchunk{defun TruthP}
\getchunk{defun try-get-token}
\getchunk{defun tuple2List}

\getchunk{defun uncons}
\getchunk{defun underscore}
\getchunk{defun unget-tokens}
\getchunk{defun unknownTypeError}
\getchunk{defun unloadOneConstructor}
\getchunk{defun unTuple}
\getchunk{defun updateCategoryFrameForCategory}
\getchunk{defun updateCategoryFrameForConstructor}

\getchunk{defun whoOwns}
\getchunk{defun wrapDomainSub}
\getchunk{defun writeLib1}

\getchunk{postvars}

```

Bibliography

- [1] Jenks, R.J. and Sutor, R.S. “Axiom – The Scientific Computation System” Springer-Verlag New York (1992) ISBN 0-387-97855-0
- [2] Knuth, Donald E., “Literate Programming” Center for the Study of Language and Information ISBN 0-937073-81-4 Stanford CA (1992)
- [3] Daly, Timothy, “The Axiom Wiki Website”
<http://axiom.axiom-developer.org>
- [4] Watt, Stephen, “Aldor”,
<http://www.aldor.org>
- [5] Lamport, Leslie, “Latex – A Document Preparation System”, Addison-Wesley, New York ISBN 0-201-52983-1
- [6] Ramsey, Norman “Noweb – A Simple, Extensible Tool for Literate Programming”
<http://www.eecs.harvard.edu/~nr/noweb>
- [7] Daly, Timothy, ”The Axiom Literate Documentation”
<http://axiom.axiom-developer.org/axiom-website/documentation.html>
- [8] Pratt, Vaughn “Top down operator precedence” POPL ’73 Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages
hall.org.ua/halls/wizzard/pdf/Vaughan.Pratt.TDOP.pdf
- [9] Floyd, R. W. “Semantic Analysis and Operator Precedence” JACM 10, 3, 316-333 (1963)

Chapter 15

Index

Index

- `+- >`, 315
 - `defplist`, 315
- `- >`, 384
 - `defplist`, 384
- `<=`, 121
 - `defplist`, 121
- `==>`, 385
 - `defplist`, 385
- `=>`, 381
 - `defplist`, 381
- `>`, 108
 - `defplist`, 108
- `>=`, 106, 107
 - `defplist`, 106, 107
- `*comp370-apply*`
 - usedby `spad`, 549
- `*eof*`
 - usedby `read-a-line`, 601
 - usedby `spad`, 549
- `*fileactq-apply*`
 - usedby `spad`, 549
- `*lisp-bin-filetype*`
 - usedby `recompile-lib-file-if-necessary`, 597
- `*standard-output*`
 - usedby `compileFileQuietly`, 598
- `*terminal-io*`
 - usedby `is-console`, 527
- `., 376`
 - `defplist`, 376
- `-`, 222
 - `defplist`, 222
- `/`, 391
 - `defplist`, 391
- `/RQ,LIB`, 539
 - calledby `compilerDoit`, 538
 - calls `/rf-1`, 539
 - calls `echo-meta`[5], 539
 - uses `$lisplib`, 539
 - defun, 539
- `/editfile`
 - usedby `/rf-1`, 540
 - usedby `compAdd`, 256
 - usedby `compFunctorBody`, 196
 - usedby `compileSpad2Cmd`, 536
 - usedby `compiler`, 533
 - usedby `initializeLisplib`, 175
 - usedby `spad`, 549
- `/major-version`
 - usedby `initializeLisplib`, 175
- `/rf-1`, 540
 - calledby `/RQ,LIB`, 539
 - calls `makeInputFilename`[5], 540
 - calls `ncINTERPFILE`, 540
 - calls `spad`[5], 540
 - uses `/editfile`, 540
 - uses `echo-meta`, 540
 - defun, 540
- `/rf[5]`
 - called by `compilerDoit`, 538
- `/rq[5]`
 - called by `compilerDoit`, 538
- `., 100, 274, 374`
 - `defplist`, 100, 274, 374
- `::, 99, 352, 375`
 - `defplist`, 99, 352, 375
- `:BF:, 369`
 - `defplist`, 369
- `;;, 389`
 - `defplist`, 389
- `==`, 378
 - `defplist`, 378
- `*eof*`
 - local `ref preparseReadLine`, 85
- `$/editfile`

- local ref finalizeLisplib, 176
- \$BasicPredicates, 211
 - local ref optPredicateIfTrue, 211
 - defvar, 211
- \$Boolean
 - local ref compArgumentConditions, 294
 - local ref compHas, 303
 - local ref doItIf, 265
 - usedby compCase1, 268
 - usedby compIf, 305
 - usedby compIs, 312
 - usedby compReduce1, 321
 - usedby compRepeatOrCollect, 323
 - usedby compSubDomain1, 341
 - usedby compSuchthat, 343
 - usedby compSymbol, 569
- \$CapsuleDomainsInScope
 - local def compDefineCapsuleFunction, 289
 - local def putDomainsInScope, 235
 - local ref getDomainsInScope, 234
- \$CapsuleModemapFrame
 - local def addModemapKnown, 253
 - local def addModemap, 253
 - local def compDefineCapsuleFunction, 289
 - local ref addModemap, 253
- \$CategoryFrame
 - local def updateCategoryFrameForCategory, 113
 - local def updateCategoryFrameForConstructor, 112
 - local ref isFunctor, 233
 - local ref loadLibIfNecessary, 111
 - local ref mkEvaluableCategoryForm, 141
 - local ref updateCategoryFrameForCategory, 113
 - local ref updateCategoryFrameForConstructor, 112
 - usedby compDefineFunctor1, 184
 - usedby compSubDomain1, 341
 - usedby compWithMappingModel1, 586
 - usedby parseHasRhs, 110
 - usedby parseHas, 109
- \$CategoryNames
 - local ref mkEvaluableCategoryForm, 141
- \$Category
 - local ref augModemapsFromDomain, 236
 - local ref compApplication, 574
 - local ref compFocompFormWithModemap, 581
 - local ref compMakeCategoryObject, 182
 - local ref mkEvaluableCategoryForm, 141
 - usedby compConstructorCategory, 282
 - usedby compDefine1, 283
 - usedby compJoin, 313
- \$CheckVectorList
 - usedby compDefineFunctor1, 184
 - usedby displayMissingFunctions, 198
- \$ConditionalOperators
 - usedby genDomainOps, 202
 - usedby makeFunctorArgumentParameters, 199
- \$ConstructorCache
 - local ref compileConstructor1, 153
- \$ConstructorNames
 - usedby compDefine1, 283
- \$DomainFrame
 - usedby s-process, 551
- \$DomainsInScope
 - local ref compDefineCapsuleFunction, 288
- \$DoubleFloat
 - usedby primitiveType, 568
- \$DummyFunctorNames
 - local ref augModemapsFromDomain, 236
 - local ref mustInstantiate, 274
- \$EchoLineStack
 - usedby fincomblock, 526
 - usedby preparse-echo, 88
 - usedby preparseReadLine1, 87
- \$EchoLines
 - usedby ncINTERPFILE, 595
- \$EmptyEnvironment
 - local ref augLisplibModemapsFromCategory, 157
 - local ref compHasFormat, 303
 - local ref getInverseEnvironment, 310
 - local ref getSuccessEnvironment, 309
 - usedby genDomainViewList, 201
 - usedby s-process, 551
- \$EmptyMode, 131
 - local def NRTgetLocalIndex, 192
 - local ref coerceEasy, 346
 - local ref compApply, 563

- local ref compHasFormat, 303
- local ref compToApply, 573
- local ref compileDocumentation, 174
- local ref doIt, 262
- local ref getSuccessEnvironment, 309
- local ref makeCategoryForm, 278
- local ref mkEvalableCategoryForm, 141
- local ref resolve, 356
- local ref setqMultipleExplicit, 333
- local ref setqMultiple, 331
- usedby compAdd, 256
- usedby compArgumentsAndTryAgain, 585
- usedby compCase1, 268
- usedby compColonInside, 565
- usedby compCons1, 279
- usedby compDefine1, 283
- usedby compDefineAddSignature, 133
- usedby compDefineCategory1, 137
- usedby compForm1, 572
- usedby compForm2, 579
- usedby compIs, 312
- usedby compMacro, 317
- usedby compNoStacking, 558
- usedby compPretend, 319
- usedby compSetq1, 330
- usedby compSubDomain1, 341
- usedby compWhere, 345
- usedby compWithMappingModel, 586
- usedby primitiveType, 568
- usedby s-process, 551
- usedby setqSingle, 335
- defvar, 131
- \$EmptyVector
 - usedby compVector, 344
- \$Exit
 - local ref coerceEasy, 347
- \$Expression
 - local ref coerceExtraHard, 349
 - local ref compExpressionList, 578
 - local ref outputComp, 337
 - usedby compAtom, 566
 - usedby compForm1, 572
 - usedby compSymbol, 569
- \$FormalMapVariableList, 250
 - local ref applyMapping, 562
 - local ref compDefineCategory2, 142
 - local ref compFocompFormWithModemap, 581
 - local ref compHasFormat, 303
 - local ref finalizeDocumentation, 466
 - local ref finalizeLisplib, 177
 - local ref getSignatureFromMode, 287
 - local ref getSlotFromCategoryForm, 179
 - local ref interactiveModemapForm, 160
 - local ref isDomainConstructorForm, 339
 - local ref substVars, 168
 - usedby compColon, 275
 - usedby compDefineFunctor1, 184
 - usedby compSymbol, 569
 - usedby compTypeOf, 564
 - usedby compWithMappingModel, 586
 - usedby makeCategoryPredicates, 138
 - usedby mkCategoryPackage, 139
 - usedby mkOpVec, 203
 - usedby substNames, 251
 - defvar, 250
- \$GensymAssoc
 - local def EqualBarGensym, 228
 - local ref EqualBarGensym, 228
- \$HTlinks
 - local ref checkRecordHash, 473
- \$HTlisplinks
 - local ref checkRecordHash, 473
- \$HTmacs
 - local ref checkAddMacros, 501
- \$HTspadmacros
 - local ref checkFixCommonProblem, 508
- \$Index
 - usedby s-process, 551
- \$Information
 - local ref compMapCond", 241
- \$InitialDomainsInScope
 - usedby spad, 549
- \$InteractiveFrame
 - usedby s-process, 551
 - usedby spad, 549
- \$InteractiveMode
 - local def addConstructorModemaps, 238
 - local ref addModemap, 253
 - local ref coerce, 346
 - local ref displayPreCompilationErrors, 516
 - local ref isFunctor, 234

- local ref loadLibIfNecessary, 111
- local ref mkNewModemapList, 246
- local ref optCatch, 225
- local ref optSPADCALL, 223
- usedby bumperrorcount, 517
- usedby compileFileQuietly, 598
- usedby compileSpad2Cmd, 536
- usedby dollarTran, 459
- usedby parseAnd, 97
- usedby parseAtSign, 98
- usedby parseCoerce, 100
- usedby parseColon, 100
- usedby parseHas, 109
- usedby parseIf,ifTran, 114
- usedby parseNot, 124
- usedby postBigFloat, 370
- usedby postDef, 379
- usedby postError, 364
- usedby postMDef, 385
- usedby postReduce, 387
- usedby spad, 549
- usedby tuple2List, 522
- \$LocalDomainAlist
 - local def doIt, 262
 - local ref doIt, 262
 - usedby compDefineFunctor1, 184
- \$LocalFrame
 - usedby s-process, 551
- \$NRTaddForm
 - local ref NRTassocIndex, 336
 - local ref NRTgetLocalIndex, 192
 - usedby compAdd, 256
 - usedby compDefineFunctor1, 184
 - usedby compFunctorBody, 196
 - usedby compSubDomain, 340
- \$NRTaddList
 - usedby compDefineFunctor1, 184
- \$NRTattributeAlist
 - usedby compDefineFunctor1, 184
- \$NRTbase
 - local def NRTgetLocalIndex, 192
 - local ref NRTassocIndex, 336
 - usedby compDefineFunctor1, 184
- \$NRTdeltaLength
 - local ref NRTassocIndex, 336
 - local ref NRTgetLocalIndex, 192
 - usedby compDefineFunctor1, 184
- \$NRTdeltaListComp
 - local ref NRTgetLocalIndex, 192
 - usedby compDefineFunctor1, 184
- \$NRTdeltaList
 - local ref NRTassocIndex, 336
 - local ref NRTgetLocalIndex, 192
 - usedby compDefineFunctor1, 184
- \$NRTderivedTargetIfTrue
 - usedby compTopLevel, 554
- \$NRTdomainFormList
 - usedby compDefineFunctor1, 184
- \$NRTloadTimeAlist
 - usedby compDefineFunctor1, 184
- \$NRTopt
 - local ref doIt, 262
- \$NRTslot1Info
 - usedby compDefineFunctor1, 184
- \$NRTslot1PredicateList
 - local def finalizeLisplib, 177
 - usedby compDefineFunctor1, 184
- \$NegativeInteger
 - usedby primitiveType, 568
- \$NoValueMode, 131
 - local ref coerceEasy, 347
 - local ref compAtomWithModemap, 567
 - local ref resolve, 356
 - local ref setqMultipleExplicit, 333
 - local ref setqMultiple, 331
 - usedby compDefine1, 283
 - usedby compImport, 312
 - usedby compMacro, 317
 - usedby compRepeatOrCollect, 323
 - usedby compSeq1, 326
 - usedby compSymbol, 569
 - usedby setqSingle, 335
 - defvar, 131
- \$NoValue
 - usedby compSymbol, 569
 - usedby parseAtom, 94
- \$NonMentionableDomainNames
 - local ref doIt, 262
- \$NonNegativeInteger
 - usedby primitiveType, 568
- \$NumberOfArgsIfInteger
 - usedby compForm1, 572

- \$One
 - usedby compElt, 300
- \$PatternVariableList
 - local ref augLisplibModemapsFromCategory, 157
 - local ref augmentLisplibModemapsFromFunctor, 193
 - local ref formal2Pattern, 195
 - local ref interactiveModemapForm, 160
 - local ref modemapPattern, 169
- \$PolyMode
 - usedby s-process, 551
- \$PositiveInteger
 - usedby primitiveType, 568
- \$PrettyPrint
 - usedby print-defun, 553
- \$PrintOnly
 - usedby s-process, 551
- \$QuickCode
 - local ref doIt, 262
 - local ref optCall, 214
 - local ref optSpecialCall, 216
 - local ref putInLocalDomainReferences, 154
 - usedby compDefineFunctor1, 185
 - usedby compWithMappingMode1, 586
 - usedby compileSpad2Cmd, 536
- \$QuickLet
 - usedby compileSpad2Cmd, 536
 - usedby setqSingle, 335
- \$ReadingFile
 - usedby ncINTERPFILE, 595
- \$Representation
 - local def doIt, 262
 - local ref doIt, 262
 - usedby compDefineFunctor1, 184
 - usedby compNoStacking, 558
- \$Rep
 - local ref coerce, 346
 - local ref mkUnion, 356
- \$SpecialDomainNames
 - local ref isDomainForm, 338
 - usedby addEmptyCapsuleIfNecessary, 134
- \$StringCategory
 - usedby compString, 339
- \$String
 - local ref coerceHard, 348
 - local ref resolve, 356
 - usedby primitiveType, 568
- \$Symbol
 - usedby compSymbol, 569
- \$TranslateOnly
 - usedby s-process, 551
- \$Translation
 - usedby s-process, 551
- \$TriangleVariableList
 - local ref compDefineCategory2, 142
 - usedby compForm2, 579
 - usedby makeCategoryPredicates, 138
- \$Undef
 - local ref optSpecialCall, 216
- \$VariableCount
 - usedby s-process, 551
- \$Void
 - local ref coerceEasy, 347
- \$Zero
 - usedby compElt, 300
- \$abbreviationTable
 - local def getAbbreviation, 285
 - local ref getAbbreviation, 285
- \$addFormLhs
 - usedby compAdd, 256
 - usedby compSubDomain, 340
- \$addForm
 - local def compDefineCategory2, 143
 - usedby compAdd, 256
 - usedby compCapsuleInner, 259
 - usedby compDefineFunctor1, 184
 - usedby compSubDomain, 340
- \$algebraOutputStream
 - local ref compDefineLisplib, 171
- \$alternateViewList
 - usedby makeFunctorArgumentParameters, 199
- \$argl
 - local def checkComments, 481
 - local def transDoc, 469
 - local ref checkDecorate, 508
 - local ref checkRewrite, 471
- \$argumentConditionList
 - local def addArgumentConditions, 294
 - local def compArgumentConditions, 294
 - local def compDefineCapsuleFunction, 289

- local def stripOffArgumentConditions, 296
 - local def stripOffSubdomainConditions, 295
 - local ref addArgumentConditions, 294
 - local ref compArgumentConditions, 294
 - local ref stripOffArgumentConditions, 296
 - local ref stripOffSubdomainConditions, 295
- \$atList
 - local def compCategory, 271
 - local ref compCategoryItem, 271
 - local ref compCategory, 271
- \$attribute?
 - local def transDoc, 469
 - local ref checkComments, 481
 - local ref transDoc, 469
- \$attributesName
 - usedby compDefineFunctor1, 184
- \$base
 - local def augModemapsFromCategoryRep, 252
 - local def augModemapsFromCategory, 245
- \$beginEndList
 - local ref checkBeginEnd, 484
- \$bindings
 - local def compApplyModemap, 239
 - local ref compApplyModemap, 239
 - local ref compMapCond, 241
- \$body
 - local ref addArgumentConditions, 294
- \$bootStrapMode
 - local ref coerceHard, 348
 - local ref optCall, 214
 - usedby comp2, 559
 - usedby compAdd, 256
 - usedby compCapsule, 258
 - usedby compColon, 275
 - usedby compDefineCategory1, 137
 - usedby compDefineFunctor1, 184
 - usedby compFunctorBody, 196
- \$boot
 - local ref PARSE-FloatTok, 447
 - local ref PARSE-Primary1, 429
 - usedby PARSE-Quad, 433
 - usedby PARSE-Selector, 427
 - usedby PARSE-TokTail, 424
 - usedby aplTran1, 395
 - usedby aplTran, 395
 - usedby postAtom, 361
 - usedby postBigFloat, 370
 - usedby postColonColon, 375
 - usedby postDef, 379
 - usedby postForm, 364
 - usedby postIf, 381
 - usedby postMDef, 385
 - usedby quote-if-string, 450
 - usedby spad, 549
 - usedby tuple2List, 522
- \$byConstructors, 599
 - local ref prepare1, 81
 - usedby compilerDoit, 538
 - defvar, 599
- \$byteAddress
 - usedby compDefineFunctor1, 185
- \$byteVec
 - usedby compDefineFunctor1, 185
- \$categoryPredicateList
 - usedby compDefineCategory1, 137
 - usedby mkCategoryPackage, 139
- \$charBack
 - local ref checkAddBackSlashes, 511
 - local ref checkBeginEnd, 484
 - local ref checkDecorate, 508
 - local ref checkRecordHash, 473
 - local ref checkSplitBackslash, 496
 - local ref checkSplitOn, 499
 - local ref checkSplitPunctuation, 497
 - local ref checkTransformFirsts, 488
 - local ref htcharPosition, 501
 - local ref removeBackslashes, 476
- \$charBlank
 - local ref checkAddSpaceSegments, 505
 - local ref checkAddSpaces, 512
 - local ref checkLookForLeftBrace, 487
 - local ref checkSkipBlanks, 491
 - local ref checkTrim, 506
 - local ref newWordFrom, 503
- \$charComma
 - local ref checkGetArgs, 504
 - local ref checkSplitPunctuation, 497
- \$charDash
 - local ref checkSplitPunctuation, 497

- \$charDelimiters
 - local ref checkSkipOpToken, 492
- \$charEscapeList
 - local ref checkAddBackSlashes, 511
- \$charExclusions
 - local ref checkDecorate, 508
- \$charFauxNewline
 - local ref checkAddSpaces, 512
 - local ref checkIndentedLines, 502
 - local ref newWordFrom, 503
- \$charIdentifierEndings
 - local ref checkAlphabetic, 491
- \$charLbrace
 - local ref checkBeginEnd, 484
 - local ref checkDecorateForHt, 477
 - local ref checkDecorate, 508
 - local ref checkFixCommonProblem, 508
 - local ref checkLookForLeftBrace, 487
 - local ref checkLookForRightBrace, 487
 - local ref checkTexht, 476
- \$charPeriod
 - local ref checkIeEgfun, 494
 - local ref checkSplitPunctuation, 497
- \$charPlus
 - local ref checkTrim, 506
- \$charQuote
 - local ref checkSplitPunctuation, 497
- \$charRbrace
 - local ref checkBeginEnd, 484
 - local ref checkDecorateForHt, 477
 - local ref checkDecorate, 508
 - local ref checkLookForRightBrace, 487
 - local ref checkTexht, 476
- \$charSemiColon
 - local ref checkSplitPunctuation, 497
- \$charSplitList
 - local ref checkSplitOn, 499
- \$checkErrorFlag
 - local def checkComments, 481
 - local def checkDocError, 479
 - local ref checkComments, 481
 - local ref checkDocError, 479
 - local ref checkRewrite, 471
- \$checkPrenAlist
 - local ref checkBalance, 483
 - local ref checkTransformFirsts, 488
- \$checkingXmptex?
 - local def transformAndRecheckComments, 470
 - local ref checkDecorateForHt, 477
 - local ref checkDecorate, 508
 - local ref checkRewrite, 471
- \$clamList
 - local def compileConstructor1, 153
 - local ref compileConstructor1, 153
- \$comblocklist, 525
 - local def collectComBlock, 465
 - local def recordHeaderDocumentation, 464
 - local ref collectAndDeleteAssoc, 465
 - local ref finalizeDocumentation, 466
 - local ref recordHeaderDocumentation, 464
 - usedby fincomblock, 526
 - usedby preparse, 77
 - defvar, 525
- \$compErrorMessageStack
 - usedby compOrCroak1, 557
- \$compForModelIfTrue
 - local def compForMode, 315
 - usedby compSymbol, 569
- \$compStack
 - local def compOrCroak1, 557
 - local ref compNoStacking1, 558
 - local ref compNoStacking, 558
 - local ref comp, 558
- \$compTimeSum
 - usedby compTopLevel, 554
- \$compUniquelyIfTrue
 - local def compUniquely, 585
 - local ref compForm3, 581
 - usedby s-process, 551
- \$compileDocumentation
 - local ref checkDocError1, 478
 - local ref compDefineLisplib, 171
- \$compileOnlyCertainItems
 - local ref compDefineCapsuleFunction, 289
 - local ref compile, 146
 - usedby compDefineFunctor1, 185
 - usedby compileSpad2Cmd, 536
- \$condAlist
 - usedby compDefineFunctor1, 185
- \$constructorLineNumber
 - usedby preparse, 77

- \$constructorName
 - local ref checkDocError, 479
 - local ref checkDocMessage, 479
 - local ref transDocList, 469
- \$constructorsSeen, 599
 - local ref preparse1, 81
 - usedby compilerDoit, 538
 - defvar, 599
- \$currentFunction
 - usedby s-process, 551
- \$currentLine
 - usedby s-process, 551
- \$currentSysList
 - local ref checkRecordHash, 473
- \$defOp
 - usedby parseTransform, 93
 - usedby postError, 364
 - usedby postTransformCheck, 363
 - usedby setDefOp, 394
- \$definition
 - local def compDefineCategory2, 142
 - local ref compDefineCategory2, 142
- \$defstack, 401
 - defvar, 401
- \$devaluateList
 - local ref NRTputInTail, 155
- \$doNotCompileJustPrint
 - local ref compile, 146
- \$docList
 - local def recordDocumentation, 464
 - local ref finalizeDocumentation, 466
 - usedby postDef, 379
 - usedby preparse, 77
- \$domainShell
 - local def compDefineCategory2, 143
 - local ref augLisplibModemapsFromCategory, 157
 - local ref hasSigInTargetCategory, 298
 - usedby compDefineCategory, 171
 - usedby compDefineFunctor1, 185
 - usedby compDefineFunctor, 183
 - usedby getOperationAlist, 250
- \$echolinestack, 72
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - defvar, 72
- \$elt
 - local def putInLocalDomainReferences, 154
 - local ref NRTputInHead, 155
 - local ref NRTputInTail, 155
- \$endTestList
 - usedby compReduce1, 321
- \$envHashTable
 - usedby compTopLevel, 554
- \$env
 - usedby displayMissingFunctions, 198
- \$erase
 - local ref initializeLisplib, 175
- \$exitModeStack
 - usedby compExit, 302
 - usedby compLeave, 317
 - usedby compOrCroak1, 557
 - usedby compRepeatOrCollect, 323
 - usedby compReturn, 325
 - usedby compSeq1, 326
 - usedby compSeq, 326
 - usedby comp, 558
 - usedby modifyModeStack, 594
 - usedby s-process, 551
- \$exitMode
 - local ref coerceExit, 351
 - usedby s-process, 551
- \$exposeFlagHeading
 - local def checkDocError, 479
 - local def transformAndRecheckComments, 471
 - local ref checkDocError, 479
 - local ref transformAndRecheckComments, 470
- \$exposeFlag
 - local ref checkDocError, 479
 - local ref whoOwns, 480
- \$extraParms
 - local def compDefineCategory2, 142
 - local ref compDefineCategory2, 142
- \$e
 - local def NRTgetLocalIndex, 192
 - local def addEltModemap, 245
 - local def augmentLisplibModemapsFromFunctor, 193
 - local def coerceHard, 348

- local def compApplyModemap, 239
- local def compCapsuleItems, 260
- local def compHas, 302
- local def doItIf, 265
- local def doIt, 262
- local def mkEvalableCategoryForm, 141
- local ref addModemapKnown, 253
- local ref addModemap, 253
- local ref augmentLisplibModemapsFrom-Functor, 193
- local ref coerceHard, 348
- local ref compApplyModemap, 239
- local ref compCapsuleItems, 260
- local ref compHasFormat, 303
- local ref compHas, 302
- local ref compMakeCategoryObject, 182
- local ref compMapCond", 241
- local ref compSingleCapsuleItem, 261
- local ref compileDocumentation, 174
- local ref compile, 146
- local ref doItIf, 265
- local ref doIt, 262
- local ref finalizeDocumentation, 466
- local ref getSignature, 297
- local ref getSlotFromFunctor, 181
- local ref mkAlistOfExplicitCategoryOps, 158
- local ref mkDatabasePred, 195
- local ref mkEvalableCategoryForm, 141
- local ref optCallSpecially, 215
- local ref signatureTran, 163
- usedby comp3, 560
- usedby compReduce1, 321
- usedby genDomainOps, 202
- usedby genDomainView, 201
- usedby getOperationAlist, 250
- usedby mkCategoryPackage, 139
- usedby s-process, 551
- \$fcopy
 - local ref compileDocumentation, 174
- \$filep
 - local ref compDefineLisplib, 171
- \$finalEnv
 - local def compDefineCapsuleFunction, 289
 - local def replaceExitEtc, 327
 - local ref replaceExitEtc, 327
- usedby compSeq1, 326
- \$forceAdd
 - local ref mergeModemap, 248
 - local ref mkNewModemapList, 246
 - usedby compTopLevel, 554
 - usedby makeFunctorArgumentParameters, 199
- \$formalArgList
 - local def compDefineCapsuleFunction, 289
 - local def compDefineCategory2, 142
 - local ref NRTgetLocalIndex, 192
 - local ref applyMapping, 562
 - local ref compDefineCapsuleFunction, 288
 - local ref compDefineCategory2, 142
 - usedby compDefine1, 283
 - usedby compReduce, 320
 - usedby compRepeatOrCollect, 323
 - usedby compSymbol, 569
 - usedby compWithMappingModel1, 586
 - usedby compWithMappingMode, 586
 - usedby displayMissingFunctions, 198
- \$formalMapVariables
 - local def hasFormalMapVariable, 592
- \$formatArgList
 - local ref compApplication, 574
 - usedby compWithMappingModel1, 586
- \$form
 - local def compDefineCapsuleFunction, 289
 - local def compDefineCategory2, 142
 - local ref applyMapping, 562
 - local ref compApplication, 574
 - local ref compDefineCategory2, 142
 - local ref compHasFormat, 303
 - usedby compCapsuleInner, 259
 - usedby compDefine1, 283
 - usedby compDefineFunctor1, 185
 - usedby s-process, 551
 - usedby setqSingle, 335
- \$found
 - local ref NRTassocIndex, 336
- \$fromCoerceable
 - local ref autoCoerceByModemap, 354
 - local ref coerceable, 350
 - local ref coerce, 346
- \$frontier
 - local def compDefineCategory2, 142

- \$functionLocations
 - local def compDefineCapsuleFunction, 289
 - local ref compDefineCapsuleFunction, 288
 - local ref transformOperationAlist, 180
 - usedby compDefineFunctor1, 185
- \$functionName
 - local ref addArgumentConditions, 294
- \$functionStats
 - local def compDefineCapsuleFunction, 289
 - local def compDefineCategory2, 142
 - local def compile, 146
 - local ref compDefineCapsuleFunction, 289
 - local ref compile, 146
 - usedby compDefineFunctor1, 185
 - usedby reportOnFunctorCompilation, 197
- \$functorForm
 - local def compDefineCategory2, 143
 - local ref addModemap0, 254
 - local ref compile, 146
 - local ref getSpecialCaseAssoc, 293
 - usedby compAdd, 256
 - usedby compCapsule, 258
 - usedby compDefineFunctor1, 185
 - usedby compFunctorBody, 196
 - usedby getOperationAlist, 250
- \$functorLocalParameters
 - local def doItIf, 265
 - local def doIt, 262
 - local ref doItIf, 265
 - local ref doIt, 262
 - usedby compCapsuleInner, 259
 - usedby compDefineFunctor1, 185
 - usedby compSymbol, 569
- \$functorSpecialCases
 - local ref getSpecialCaseAssoc, 293
 - usedby compDefineFunctor1, 185
- \$functorStats
 - local def compDefineCategory2, 142
 - local ref compDefineCapsuleFunction, 289
 - usedby compDefineFunctor1, 185
 - usedby reportOnFunctorCompilation, 197
- \$functorTarget
 - usedby compDefineFunctor1, 185
- \$functorsUsed
 - local def doIt, 262
 - local ref doIt, 262
- usedby compDefineFunctor1, 185
 - \$funnameTail
 - usedby compWithMappingMode1, 586
 - \$funname
 - usedby compWithMappingMode1, 586
 - \$f
 - usedby compileSpad2Cmd, 536
 - \$genFVar
 - usedby compDefineFunctor1, 185
 - usedby s-process, 551
 - \$genSDVar
 - usedby compDefineFunctor1, 185
 - usedby s-process, 551
 - \$genno
 - local def aplTran, 395
 - local def doIt, 262
 - \$getDomainCode
 - local def compDefineCategory2, 142
 - local ref compileCases, 292
 - local ref doItIf, 265
 - local ref optCallSpecially, 215
 - usedby compCapsuleInner, 259
 - usedby compDefineFunctor1, 185
 - usedby genDomainOps, 202
 - usedby genDomainView, 201
 - \$glossHash
 - local def checkRecordHash, 473
 - local ref checkRecordHash, 473
 - \$goGetList
 - usedby compDefineFunctor1, 185
 - \$headerDocumentation
 - local def recordHeaderDocumentation, 464
 - local ref recordHeaderDocumentation, 464
 - usedby postDef, 379
 - usedby preparse, 77
 - \$htHash
 - local def checkRecordHash, 473
 - local ref checkRecordHash, 473
 - \$htMacroTable
 - local ref checkArguments, 486
 - local ref checkBeginEnd, 484
 - local ref checkSplitPunctuation, 497
 - \$in-stream
 - local ref Advance-Char, 607
 - local ref next-line, 606
 - local ref preparse1, 81

- \$index, 72
 - local def preparse1, 81
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$initCapsuleErrorCount
 - local def compDefineCapsuleFunction, 289
- \$initList
 - usedby compReduce1, 321
- \$insideCapsuleFunctionIfTrue
 - local def compDefineCapsuleFunction, 289
 - local ref CapsuleModemapFrame, 253
 - local ref addEltModemap, 245
 - local ref addModemap, 253
 - local ref compile, 146
 - local ref getDomainsInScope, 234
 - local ref putDomainsInScope, 235
 - local ref spadCompileOrSetq, 152
 - usedby compDefine1, 283
 - usedby s-process, 551
- \$insideCategoryIfTrue
 - local def compDefineCategory2, 142
 - usedby compCapsuleInner, 259
 - usedby compColon, 275
 - usedby compDefine1, 283
 - usedby s-process, 551
- \$insideCategoryPackageIfTrue
 - local ref getFormModemaps, 576
 - usedby compCapsuleInner, 259
 - usedby compDefineCategory1, 137
 - usedby compDefineFunctor1, 185
- \$insideCoerceInteractiveHardIfTrue
 - usedby s-process, 551
- \$insideCompTypeOf
 - usedby comp3, 560
 - usedby compTypeOf, 564
- \$insideConstructIfTrue
 - local ref parseColon, 100
 - usedby parseConstruct, 95
- \$insideExpressionIfTrue
 - local def compDefineCapsuleFunction, 289
 - usedby compCapsule, 258
 - usedby compColon, 275
 - usedby compDefine1, 283
 - usedby compExpression, 571
 - usedby compMakeDeclaration, 593
 - usedby compSeq1, 326
 - usedby compWhere, 345
 - usedby s-process, 551
- \$insideFunctorIfTrue
 - local ref compileCases, 292
 - usedby compColon, 275
 - usedby compDefine1, 283
 - usedby compDefineCategory, 171
 - usedby compDefineFunctor1, 185
 - usedby getOperationAlist, 250
 - usedby s-process, 551
- \$insidePostCategoryIfTrue
 - usedby postCategory, 371
 - usedby postWith, 394
- \$insideSetqSingleIfTrue
 - usedby setqSingle, 335
- \$insideWhereIfTrue
 - usedby compDefine1, 283
 - usedby compWhere, 345
 - usedby s-process, 551
- \$is-eqlist, 402
 - defvar, 402
- \$is-gensymlist, 402
 - defvar, 402
- \$is-spill-list, 401
 - defvar, 401
- \$is-spill, 401
 - defvar, 401
- \$isOpPackageName
 - usedby compDefineFunctor1, 185
- \$keywords
 - local ref escape-keywords, 451
- \$killOptimizeIfTrue
 - usedby compTopLevel, 554
 - usedby compWithMappingMode1, 586
- \$leaveLevelStack
 - usedby compLeave, 317
 - usedby compRepeatOrCollect, 323
 - usedby s-process, 551
- \$leaveMode
 - usedby s-process, 551
- \$level
 - usedby compOrCroak1, 557
- \$lhsOfColon

- local def evalAndSub, 249
 - usedby compColon, 275
 - usedby compSubsetCategory, 342
- \$lhs
 - usedby parseDEF, 101
 - usedby parseMDEF, 123
- \$libFile
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref compDefineCategory2, 142
 - local ref compilerDoitWithScreenedLisplib, 358
 - local ref finalizeLisplib, 176
 - local ref initializeLisplib, 175
 - local ref rwriteLispForm, 170
 - usedby compDefineFunctor1, 185
- \$line-handler
 - local ref next-line, 606
- \$linelist, 72
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - defvar, 72
- \$line
 - usedby Advance-Char, 607
 - usedby current-char, 457
 - usedby line-advance-char, 605
 - usedby line-at-end-p, 604
 - usedby line-clear, 604
 - usedby line-new-line, 606
 - usedby line-next-char, 605
 - usedby line-past-end-p, 605
 - usedby line-print, 604, 606
 - usedby match-advance-string, 449
 - usedby match-string, 448
- \$lispHash
 - local def checkRecordHash, 473
 - local ref checkRecordHash, 473
- \$lisplibAbbreviation
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeLisplib, 177
 - usedby compDefineFunctor1, 185
- \$lisplibAncestors
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeLisplib, 177
 - usedby compDefineFunctor1, 185
- \$lisplibAttributes
 - local ref finalizeLisplib, 177
- \$lisplibCategoriesExtended
 - local def compDefineLisplib, 172
 - usedby compDefineFunctor1, 185
- \$lisplibCategory
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def finalizeLisplib, 177
 - local ref compDefineCategory2, 142
 - local ref finalizeLisplib, 176
 - usedby compDefineCategory1, 137
 - usedby compDefineCategory, 171
 - usedby compDefineFunctor1, 185
- \$lisplibForm
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeDocumentation, 466
 - local ref finalizeLisplib, 176
 - usedby compDefineFunctor1, 185
- \$lisplibFunctionLocations
 - usedby compDefineFunctor1, 185
- \$lisplibItemsAlreadyThere
 - local ref compile, 146
- \$lisplibKind
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref compDefineLisplib, 171
 - local ref finalizeLisplib, 176
 - usedby compDefineFunctor1, 185
- \$lisplibMissingFunctions
 - usedby compDefineFunctor1, 185
- \$lisplibModemapAlist
 - local def augLisplibModemapsFromCategory, 157
 - local def augmentLisplibModemapsFromFunctor, 193
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175

- local ref augLisplibModemapsFromCategory, 157
 - local ref augmentLisplibModemapsFromFunctor, 193
 - local ref finalizeLisplib, 176
- \$lisplibModemap
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeLisplib, 176, 177
 - usedby compDefineFunctor1, 185
- \$lisplibOpAlist
 - local def initializeLisplib, 175
- \$lisplibOperationAlist
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref getSlotFromFunctor, 181
 - usedby compDefineFunctor1, 185
- \$lisplibParents
 - local def compDefineCategory2, 143
 - local def compDefineLisplib, 171
 - local ref finalizeLisplib, 177
 - usedby compDefineFunctor1, 185
- \$lisplibPredicates
 - local def compDefineLisplib, 172
 - local ref finalizeLisplib, 177
- \$lisplibSignatureAlist
 - local def encodeFunctionName, 148
 - local def initializeLisplib, 175
 - local ref encodeFunctionName, 148
 - local ref finalizeLisplib, 177
- \$lisplibSlot1
 - local def compDefineLisplib, 172
 - local ref finalizeLisplib, 177
 - usedby compDefineFunctor1, 185
- \$lisplibSuperDomain
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeLisplib, 177
 - usedby compSubDomain1, 341
- \$lisplibVariableAlist
 - local def compDefineLisplib, 172
 - local def initializeLisplib, 175
 - local ref finalizeLisplib, 177
- \$lisplib
 - local def compDefineLisplib, 171
- local ref compDefineCategory2, 142
- local ref compile, 146
- local ref encodeFunctionName, 148
- local ref lisplibWrite, 182
- local ref rwriteLispForm, 170
- usedby /RQ,LIB, 539
- usedby comp2, 559
- usedby compDefineCategory, 171
- usedby compDefineFunctor1, 184
- usedby compDefineFunctor, 183
- \$lookupFunction
 - usedby compDefineFunctor1, 185
- \$macroIfTrue
 - local ref compile, 146
 - usedby compDefine, 282
 - usedby compMacro, 317
- \$macroassoc
 - usedby s-process, 551
- \$maxSignatureLineNumber
 - local def recordDocumentation, 464
 - local ref recordHeaderDocumentation, 464
 - usedby postDef, 379
 - usedby prepare, 77
- \$mutableDomains
 - usedby compDefineFunctor1, 185
- \$mutableDomain
 - local ref compileConstructor1, 153
 - usedby compDefineFunctor1, 185
- \$mvl
 - usedby makeCategoryPredicates, 138
- \$myFunctorBody
 - local def compCapsuleItems, 260
 - usedby compDefineFunctor1, 185
- \$m
 - usedby compileSpad2Cmd, 536
- \$name
 - local def transformAndRecheckComments, 470
 - local ref checkRecordHash, 473
- \$newCompilerUnionFlag
 - usedby compColonInside, 565
 - usedby compPretend, 319
- \$newComp
 - usedby compileSpad2Cmd, 536
- \$newConlist
 - local def compDefineLisplib, 172

- local ref compDefineLisplib, 171
 - usedby compileSpad2Cmd, 536
 - usedby compiler, 533
- \$newspad
 - usedby s-process, 551
- \$noEnv
 - local ref setqMultiple, 331
 - usedby compColon, 275
- \$noParseCommands
 - local ref PARSE-SpecialCommand, 413
- \$noSubsumption
 - usedby spad, 549
- \$optimizableConstructorNames
 - local ref optCallSpecially, 215
- \$options
 - usedby compileSpad2Cmd, 536
 - usedby compileSpadLispCmd, 596
 - usedby compiler, 533
 - usedby mkCategoryPackage, 139
- \$op
 - local def compDefineCapsuleFunction, 289
 - local def compDefineCategory2, 142
 - local def compDefineLisplib, 171
 - local ref applyMapping, 562
 - local ref compApplication, 574
 - local ref compDefineCapsuleFunction, 288
 - local ref compDefineCategory2, 142
 - local ref finalizeDocumentation, 466
 - usedby compDefine1, 283
 - usedby compDefineFunctor1, 185
 - usedby compSubDomain1, 341
 - usedby parseDollarGreaterEqual, 104
 - usedby parseDollarGreaterThan, 104
 - usedby parseDollarLessEqual, 105
 - usedby parseDollarNotEqual, 106
 - usedby parseGreaterEqual, 107
 - usedby parseGreaterThan, 108
 - usedby parseLessEqual, 122
 - usedby parseNotEqual, 125
 - usedby parseTran, 93
 - usedby reportOnFunctorCompilation, 197
- \$origin
 - local def transformAndRecheckComments, 471
 - local ref checkRecordHash, 473
- \$out-stream
 - local ref line-print, 604
 - local ref print-package, 521
- \$outStream
 - local ref checkDocError, 479
- \$packagesUsed
 - local def doIt, 262
 - local ref doIt, 262
 - usedby comp2, 559
 - usedby compAdd, 256
 - usedby compDefine, 282
 - usedby compTopLevel, 554
- \$pairlis
 - local def finalizeLisplib, 177
 - local ref NRTgetLookupFunction, 191
 - usedby compDefineFunctor1, 185
- \$postStack
 - local ref displayPreCompilationErrors, 516
 - usedby postError, 364
 - usedby s-process, 551
- \$predAlist
 - usedby compDefWhereClause, 205
- \$predl
 - local ref doItIf, 265
 - local ref doIt, 261
- \$pred
 - local ref compCapsuleItems, 260
 - local ref compSingleCapsuleItem, 261
- \$prefix
 - local ref applyMapping, 562
 - local ref compApplication, 574
 - local ref compile, 146
 - usedby compDefine1, 283
 - usedby compDefineCategory2, 142
- \$preparse-last-line, 72
 - local def preparse1, 81
 - local ref preparse1, 81
 - usedby initialize-preparse, 73
 - usedby preparseReadLine1, 87
 - usedby preparse, 77
 - defvar, 72
- \$preparseReportIfTrue
 - usedby preparse, 77
- \$previousTime
 - usedby s-process, 551
- \$profileAlist
 - usedby compDefineFunctor, 183

- \$profileCompiler
 - local ref compDefineCapsuleFunction, 289
 - local ref finalizeLisplib, 177
 - usedby compDefineFunctor, 183
 - usedby setqSingle, 335
- \$recheckingFlag
 - local def transformAndRecheckComments, 471
 - local ref checkDocError, 479
- \$reportExitModeStack
 - usedby modifyModeStack, 593
- \$reportOptimization
 - local ref optimizeFunctionDef, 208
- \$resolveTimeSum
 - usedby compTopLevel, 554
- \$returnMode
 - local def compDefineCapsuleFunction, 289
 - local ref compDefineCapsuleFunction, 289
 - usedby compReturn, 325
 - usedby s-process, 551
- \$savableItems
 - local def compile, 146
- \$saveableItems
 - local ref compilerDoitWithScreenedLisplib, 358
 - local ref compile, 146
- \$scanIfTrue
 - usedby compOrCroak1, 557
 - usedby compileSpad2Cmd, 536
- \$semanticErrorStack
 - local ref compDefineCapsuleFunction, 288
 - usedby reportOnFunctorCompilation, 197
 - usedby s-process, 551
- \$setOptions
 - local ref checkRecordHash, 473
- \$setelt
 - usedby compDefineFunctor1, 185
- \$sideEffectsList
 - usedby compReduce1, 321
- \$sigAlist
 - usedby compDefWhereClause, 205
- \$sigList
 - local def compCategory, 271
 - local ref compCategoryItem, 271
 - local ref compCategory, 271
- \$signatureOffForm
 - local def compCapsuleItems, 260
 - local def compDefineCapsuleFunction, 289
 - local ref compDefineCapsuleFunction, 288
 - local ref compile, 146
 - local ref doIt, 262
- \$signature
 - usedby compCapsuleInner, 259
 - usedby compDefineFunctor1, 185
- \$skipme
 - local def preparse1, 81
 - usedby preparse, 77
- \$sourceFileTypes
 - usedby compileSpad2Cmd, 536
- \$spad-errors
 - usedby bumperrorcount, 517
- \$spadLibFT
 - local ref compDefineLisplib, 171
 - local ref compileDocumentation, 174
 - local ref finalizeLisplib, 177
 - local ref lisplibDoRename, 174
- \$spad
 - usedby quote-if-string, 450
 - usedby spad, 549
- \$specialCaseKeyList
 - local def compileCases, 292
 - local ref optCallSpecially, 215
- \$splitUpItemsAlreadyThere
 - local ref compile, 146
- \$stack
 - usedby reduce-stack, 462
 - usedby stack-/-empty, 89
 - usedby stack-clear, 89
 - usedby stack-load, 89
 - usedby stack-pop, 90
 - usedby stack-push, 89
- \$stringFauxNewline
 - local ref newWordFrom, 503
- \$suffix
 - local def compCapsuleItems, 260
 - local def compile, 146
 - local ref compile, 146
- \$sysHash
 - local def checkRecordHash, 473
 - local ref checkRecordHash, 473
- \$s
 - usedby compOrCroak1, 557

- \$template
 - usedby compDefineFunctor1, 185
- \$tokenCommands
 - local ref PARSE-SpecialCommand, 413
- \$token
 - usedby current-token, 91
 - usedby make-symbol-of, 454
 - usedby match-advance-string, 449
 - usedby next-token, 91
 - usedby prior-token, 90
 - usedby token-install, 92
 - usedby token-print, 92
 - usedby valid-tokens, 91
- \$top-level
 - local def compCapsuleItems, 260
 - local def compCategory, 271
 - local def compDefineCategory2, 142
 - usedby compDefineFunctor1, 184
 - usedby s-process, 551
- \$topOp
 - local ref displayPreCompilationErrors, 516
 - usedby s-process, 551
 - usedby setDefOp, 394
- \$tripleCache
 - usedby compDefine, 282
- \$tripleHits
 - usedby compDefine, 282
- \$true
 - local ref addArgumentConditions, 294
 - local ref optCONDtail, 211
 - local ref optIF2COND, 212
- \$tv1
 - usedby makeCategoryPredicates, 138
- \$uncondAlist
 - usedby compDefineFunctor1, 185
- \$until
 - usedby compReduce1, 321
 - usedby compRepeatOrCollect, 323
- \$viewNames
 - usedby compDefineFunctor1, 185
- \$v1, 402
 - defvar, 402
- \$warningStack
 - usedby reportOnFunctorCompilation, 197
 - usedby s-process, 551
- \$why
 - local def NRTgetLookupFunction, 191
 - local ref NRTgetLookupFunction, 191
- \$x
 - local def transDoc, 469
 - local def transformAndRecheckComments, 470
 - local ref checkDocMessage, 479
 - local ref checkTrim, 506
 - local ref transDoc, 469
- , 31
- abbreviation?
 - calledby checkNumOfArgs, 499
 - calledby parseHasRhs, 110
- abbreviationsSpad2Cmd
 - calledby mkCategoryPackage, 139
- action, 461
 - calledby PARSE-AnyId, 438
 - calledby PARSE-Category, 418
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-Data, 436
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-GlyphTok, 438
 - calledby PARSE-Infix, 423
 - calledby PARSE-NBGlyphTok, 437
 - calledby PARSE-NewExpr, 411
 - calledby PARSE-OpenBrace, 440
 - calledby PARSE-OpenBracket, 440
 - calledby PARSE-Operation, 421
 - calledby PARSE-Prefix, 422
 - calledby PARSE-ReductionOp, 425
 - calledby PARSE-Sexpr1, 436
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-SpecialKeyWord, 412
 - calledby PARSE-Suffix, 442
 - calledby PARSE-TokTail, 424
 - calledby PARSE-TokenCommandTail, 414
 - calledby PARSE-TokenList, 414
- defun, 461
- add, 366
 - defplist, 366
- add-parens-and-semis-to-line, 84
 - calledby parsepiles, 84
- calls addclose, 84
- calls drop, 84

- calls infixtok, 84
- calls nonblankloc, 84
- defun, 84
- addArgumentConditions, 293
 - calledby compDefineCapsuleFunction, 288
 - calls mkq, 294
 - calls qcar, 293
 - calls qcdr, 294
 - calls systemErrorHere, 294
 - local def \$argumentConditionList, 294
 - local ref \$argumentConditionList, 294
 - local ref \$body, 294
 - local ref \$functionName, 294
 - local ref \$true, 294
 - defun, 293
- addBinding
 - calledby addModemap1, 255
 - calledby compDefineCategory2, 142
 - calledby getSuccessEnvironment, 309
 - calledby setqMultiple, 331
- addBinding[5]
 - called by setqSingle, 334
 - called by spad, 549
- addclose, 524
 - calledby add-parens-and-semis-to-line, 84
 - calls suffix, 524
 - defun, 524
- addConstructorModemaps, 238
 - calledby augModemapsFromDomain1, 237
 - calls addModemap, 238
 - calls getl, 238
 - calls msubst, 238
 - calls putDomainsInScope, 238
 - calls qcar, 238
 - calls qcdr, 238
 - local def \$InteractiveMode, 238
 - defun, 238
- AddContour
 - calledby compApply, 563
- addContour
 - calledby compWhere, 345
- addDomain, 232
 - calledby comp2, 559
 - calledby comp3, 560
 - calledby compAtSign, 352
 - calledby compCapsule, 258
 - calledby compCase, 268
 - calledby compCoerce, 352
 - calledby compColonInside, 564
 - calledby compColon, 275
 - calledby compDefineCapsuleFunction, 288
 - calledby compElt, 300
 - calledby compForm1, 572
 - calledby compImport, 312
 - calledby compPretend, 318
 - calledby compSubDomain1, 341
 - calls addNewDomain, 232
 - calls constructor?, 232
 - calls domainMember, 232
 - calls getDomainsInScope, 232
 - calls getmode, 232
 - calls identp, 232
 - calls isCategoryForm, 232
 - calls isFunctor, 232
 - calls isLiteral, 232
 - calls member, 232
 - calls qslessp, 232
 - calls unknownTypeError, 232
 - defun, 232
- addEltModemap, 245
 - calledby addModemap0, 254
 - calls addModemap1, 245
 - calls makeLiteral, 245
 - calls qcar, 245
 - calls qcdr, 245
 - calls systemErrorHere, 245
 - local def \$e, 245
 - local ref \$insideCapsuleFunctionIfTrue, 245
 - defun, 245
- addEmptyCapsuleIfNecessary, 134
 - calledby compDefine1, 283
 - calls kar, 134
 - uses \$SpecialDomainNames, 134
 - defun, 134
- addInformation
 - calledby compCapsuleInner, 259
- addModemap, 253
 - calledby addConstructorModemaps, 238
 - calledby augModemapsFromCategoryRep, 252
 - calledby genDomainOps, 202

- calledby updateCategoryFrameForCategory, 113
- calledby updateCategoryFrameForConstructor, 112
- calls addModemap0, 253
- calls knownInfo, 253
- local def \$CapsuleModemapFrame, 253
- local ref \$CapsuleModemapFrame, 253
- local ref \$InteractiveMode, 253
- local ref \$e, 253
- local ref \$insideCapsuleFunctionIfTrue, 253
- defun, 253
- addModemap0, 254
 - calledby addModemapKnown, 253
 - calledby addModemap, 253
 - calls addEltModemap, 254
 - calls addModemap1, 254
 - calls qcar, 254
 - local ref \$functorForm, 254
 - defun, 254
- addModemap1, 254
 - calledby addEltModemap, 245
 - calledby addModemap0, 254
 - calls addBinding, 255
 - calls augProplist, 254
 - calls getProplist, 254
 - calls lassoc, 254
 - calls mkNewModemapList, 254
 - calls msubst, 254
 - calls unErrorRef, 254
 - defun, 254
- addModemapKnown, 253
 - calledby augModemapsFromCategory, 245
 - calls addModemap0, 253
 - local def \$CapsuleModemapFrame, 253
 - local ref \$e, 253
 - defun, 253
- addNewDomain, 236
 - calledby addDomain, 232
 - calledby augModemapsFromDomain, 236
 - calls augModemapsFromDomain, 236
 - defun, 236
- addoptions
 - calledby initializeLisplib, 175
- addStats
 - calledby compDefineCapsuleFunction, 288
 - calledby compile, 146
 - calledby reportOnFunctorCompilation, 197
- addSuffix, 286
 - calledby mkAbbrev, 286
 - defun, 286
- Advance-Char, 607
 - calls Line-Advance-Char, 607
 - calls Line-At-End-P, 607
 - calls current-char, 607
 - calls next-line, 607
 - local ref \$in-stream, 607
 - uses \$line, 607
 - defun, 607
- advance-char
 - calledby skip-blanks, 448
- advance-token, 456
 - calledby PARSE-AnyId, 438
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-GlyphTok, 438
 - calledby PARSE-Infix, 423
 - calledby PARSE-NBGlyphTok, 437
 - calledby PARSE-OpenBrace, 440
 - calledby PARSE-OpenBracket, 440
 - calledby PARSE-Prefix, 423
 - calledby PARSE-ReductionOp, 425
 - calledby PARSE-Suffix, 442
 - calledby PARSE-TokenList, 414
 - calledby parse-argument-designator, 521
 - calledby parse-identifier, 519
 - calledby parse-keyword, 520
 - calledby parse-number, 520
 - calledby parse-spadstring, 518
 - calledby parse-string, 519
 - calls copy-token, 456
 - calls current-token, 456
 - calls try-get-token, 456
 - uses current-token, 456
 - uses valid-tokens, 456
 - defun, 456
- alistSize, 286
 - calledby mkAbbrev, 286
 - defun, 286
- allLASSOCs, 194
 - calledby augmentLisplibModemapsFromFunctor, 193

- defun, 194
- and, 97
 - defplist, 97
- aplTran, 395
 - calledby postTransform, 359
 - calls aplTran1, 395
 - calls containsBang, 395
 - local def \$genno, 395
 - uses \$boot, 395
 - defun, 395
- aplTran1, 395
 - calledby aplTran1, 395
 - calledby aplTranList, 397
 - calledby aplTran, 395
 - calledby hasAplExtension, 397
 - calls aplTran1, 395
 - calls aplTranList, 395
 - calls hasAplExtension, 395
 - calls nreverse0, 395
 - calls , 395
 - uses \$boot, 395
 - defun, 395
- aplTranList, 397
 - calledby aplTran1, 395
 - calledby aplTranList, 397
 - calls aplTran1, 397
 - calls aplTranList, 397
 - defun, 397
- applyMapping, 562
 - calledby comp3, 560
 - calls comp, 562
 - calls convert, 562
 - calls encodeItem, 562
 - calls getAbbreviation, 562
 - calls get, 562
 - calls isCategoryForm, 562
 - calls member, 562
 - calls nequal, 562
 - calls sublis, 562
 - local ref \$FormalMapVariableList, 562
 - local ref \$formalArgList, 562
 - local ref \$form, 562
 - local ref \$op, 562
 - local ref \$prefix, 562
 - defun, 562
- argsToSig, 592
 - calledby compLambda, 315
 - defun, 592
- assignError, 336
 - calledby setqSingle, 334
 - calls stackMessage, 336
 - defun, 336
- assoc
 - calledby augModemapsFromCategoryRep, 252
 - calledby checkPrenAlist, 483
 - calledby compColon, 275
 - calledby compDefineAddSignature, 133
 - calledby compForm2, 579
 - calledby mkCategoryPackage, 139
 - calledby mkNewModemapList, 246
 - calledby mkOpVec, 203
 - calledby stripOffSubdomainConditions, 295
 - calledby transformOperationAlist, 180
- AssocBarGensym, 204
 - calledby mkOpVec, 203
 - calls EqualBarGensym, 204
 - defun, 204
- assocleft
 - calledby compDefWhereClause, 204
 - calledby compileCases, 292
 - calledby finalizeDocumentation, 466
 - calledby mkAlistOfExplicitCategoryOps, 158
- assocright
 - calledby compDefWhereClause, 204
 - calledby compileCases, 292
 - calledby recordHeaderDocumentation, 464
- assq
 - calledby getAbbreviation, 285
 - calledby makeFunctorArgumentParameters, 198
 - calledby mkOpVec, 203
- assq[5]
 - called by freelist, 594
- atEndOfLine
 - calledby PARSE-TokenCommandTail, 414
- augLisplibModemapsFromCategory, 157
 - calledby compDefineCategory2, 142
 - calls interactiveModemapForm, 157
 - calls isCategoryForm, 157
 - calls lassoc, 157

- calls member, 157
- calls mkAlistOfExplicitCategoryOps, 157
- calls mkpf, 157
- calls sublis, 157
- local def \$lisplibModemapAlist, 157
- local ref \$EmptyEnvironment, 157
- local ref \$PatternVariableList, 157
- local ref \$domainShell, 157
- local ref \$lisplibModemapAlist, 157
- defun, 157
- augmentLisplibModemapsFromFunctor, 193
 - calledby compDefineFunctor1, 184
 - calls allLASSOCs, 193
 - calls formal2Pattern, 193
 - calls interactiveModemapForm, 193
 - calls listOfPatternIds, 193
 - calls member, 193
 - calls mkAlistOfExplicitCategoryOps, 193
 - calls mkDatabasePred, 193
 - calls mkpf, 193
 - calls msubst, 193
 - local def \$e, 193
 - local def \$lisplibModemapAlist, 193
 - local ref \$PatternVariableList, 193
 - local ref \$e, 193
 - local ref \$lisplibModemapAlist, 193
 - defun, 193
- augModemapsFromCategory, 245
 - calledby augModemapsFromDomain1, 237
 - calledby compDefineFunctor1, 184
 - calledby genDomainView, 201
 - calls addModemapKnown, 245
 - calls compilerMessage, 245
 - calls evalAndSub, 245
 - calls putDomainsInScope, 245
 - local def \$base, 245
 - defun, 245
- augModemapsFromCategoryRep, 251
 - calledby compDefineFunctor1, 184
 - calls addModemap, 252
 - calls assoc, 252
 - calls compilerMessage, 252
 - calls evalAndSub, 251
 - calls isCategory, 252
 - calls msubst, 252
 - calls putDomainsInScope, 252
 - local def \$base, 252
 - defun, 251
- augModemapsFromDomain, 236
 - calledby addNewDomain, 236
 - calls addNewDomain, 236
 - calls augModemapsFromDomain1, 236
 - calls getDomainsInScope, 236
 - calls getdatabase, 236
 - calls kar, 236
 - calls listOrVectorElementNode, 236
 - calls member, 236
 - calls opOf, 236
 - calls stripUnionTags, 236
 - local ref \$Category, 236
 - local ref \$DummyFunctorNames, 236
 - defun, 236
- augModemapsFromDomain1, 237
 - calledby augModemapsFromDomain, 236
 - calledby compForm1, 572
 - calledby setqSingle, 334
 - calls addConstructorModemaps, 237
 - calls augModemapsFromCategory, 237
 - calls getl, 237
 - calls getmodeOrMapping, 237
 - calls getmode, 237
 - calls kar, 237
 - calls stackMessage, 237
 - calls substituteCategoryArguments, 237
 - defun, 237
- augProplist
 - calledby addModemap1, 254
- autoCoerceByModemap, 354
 - calledby coerceExtraHard, 349
 - calls getModemapList, 354
 - calls get, 354
 - calls member, 354
 - calls modeEqual, 354
 - calls qcar, 354
 - calls qcdr, 354
 - calls stackMessage, 354
 - local ref \$fromCoerceable, 354
 - defun, 354
- awk
 - calledby whoOwns, 480
- Bang, 460

- defmacro, 460
- bang
 - calledby PARSE-Category, 417
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Form, 425
 - calledby PARSE-Import, 419
 - calledby PARSE-IteratorTail, 441
 - calledby PARSE-Seg, 444
 - calledby PARSE-Sexpr1, 437
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-TokenCommandTail, 413
- bfp-
 - calledby PARSE-FloatTok, 447
- blankp, 525
 - calledby nonblankloc, 528
 - defun, 525
- Block, 370
 - defplist, 370
- boot-line-stack
 - usedby spad, 549
- bootStrapError, 196
 - calledby compCapsule, 258
 - calledby compFunctorBody, 195
 - calls mkDomainConstructor, 196
 - calls mkq, 196
 - calls namestring, 196
 - defun, 196
- bpiname
 - calledby compileTimeBindingOf, 217
 - calledby subrname, 212
- bright
 - calledby NRTgetLookupFunction, 191
 - calledby checkAndDeclare, 298
 - calledby compDefineLisplib, 171
 - calledby displayMissingFunctions, 197
 - calledby doIt, 261
 - calledby finalizeDocumentation, 466
 - calledby hasSigInTargetCategory, 298
 - calledby optimizeFunctionDef, 208
 - calledby parseInBy, 118
 - calledby postForm, 364
 - calledby spadCompileOrSetq, 152
- browserAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 536
- buildFunctor
 - calledby processFunctor, 259
- bumperrorcount, 517
 - calledby postError, 364
 - uses \$InteractiveMode, 517
 - uses \$spad-errors, 517
 - defun, 517
- call, 213
 - defplist, 213
- cannotDo
 - calledby doIt, 261
- canReturn, 306
 - calledby canReturn, 306
 - calledby compIf, 304
 - calls canReturn, 306
 - calls qcar, 306
 - calls qcdr, 306
 - calls say, 306
 - calls systemErrorHere, 306
 - defun, 306
- capsule, 258
 - defplist, 258
- CapsuleModemapFrame
 - local ref \$insideCapsuleFunctionIfTrue, 253
- case, 267
 - defplist, 267
- catch, 224
 - defplist, 224
- catches
 - compOrCroak1, 557
 - compUniquely, 584
 - preparse1, 81
 - spad, 549
- category, 98, 270, 371
 - defplist, 98, 270, 371
- char
 - calledby isCategoryPackageName, 190
- char-eq, 458
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-TokTail, 424
 - defun, 458
- char-ne, 458
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-Selector, 427
 - defun, 458

- calledby checkSplitBrace, 495
 - calledby checkSplitOn, 498
 - calledby checkSplitPunctuation, 497
- charPosition
 - calledby checkAddSpaceSegments, 505
 - calledby checkExtract, 507
 - calledby checkGetArgs, 504
 - calledby checkSplitBackslash, 496
 - calledby checkSplitOn, 498
 - calledby checkSplitPunctuation, 497
 - calledby checkTrim, 506
 - calledby htcharPosition, 501
 - calledby removeBackslashes, 476
- chaseInferences
 - calledby compHas, 302
- checkAddBackSlashes, 511
 - calledby checkAddBackSlashes, 511
 - calledby checkDecorate, 508
 - calls checkAddBackSlashes, 511
 - calls maxindex, 511
 - calls strconc, 511
 - local ref \$charBack, 511
 - local ref \$charEscapeList, 511
 - defun, 511
- checkAddIndented
 - calledby checkRewrite, 471
- checkAddMacros, 501
 - calledby checkRewrite, 471
 - calls lassoc, 501
 - calls nreverse, 501
 - local ref \$HTmacs, 501
 - defun, 501
- checkAddPeriod, 483
 - calledby checkComments, 481
 - calls maxindex, 483
 - calls setelt, 483
 - defun, 483
- checkAddSpaces, 512
 - calledby checkComments, 481
 - calledby checkRewrite, 471
 - local ref \$charBlank, 512
 - local ref \$charFauxNewline, 512
 - defun, 512
- checkAddSpaceSegments, 505
 - calledby checkAddSpaceSegments, 505
 - calls charPosition, 505
 - calls checkAddSpaceSegments, 505
 - calls maxindex, 505
 - calls strconc, 505
 - local ref \$charBlank, 505
 - defun, 505
- checkAlphabetic, 491
 - calledby checkSkipIdentifierToken, 491
 - calledby checkSkipOpToken, 492
 - local ref \$charIdentifierEndings, 491
 - defun, 491
- checkAndDeclare, 298
 - calledby compDefineCapsuleFunction, 288
 - calls bright, 298
 - calls getArgumentMode, 298
 - calls modeEqual, 298
 - calls put, 298
 - calls sayBrightly, 298
 - defun, 298
- checkArguments, 486
 - calledby checkComments, 481
 - calledby checkRewrite, 471
 - calls checkHTargs, 486
 - calls hget, 486
 - local ref \$htMacroTable, 486
 - defun, 486
- checkBalance, 483
 - calledby checkComments, 481
 - local ref \$checkPrenAlist, 483
 - defun, 483
- checkBeginEnd, 484
 - calledby checkPrenAlist, 483
 - calls checkDocError, 484
 - calls hget, 484
 - calls ifcar, 484
 - calls ifcdr, 484
 - calls length, 484
 - calls member, 484
 - calls substring?, 484
 - local ref \$beginEndList, 484
 - local ref \$charBack, 484
 - local ref \$charLbrace, 484
 - local ref \$charRbrace, 484
 - local ref \$htMacroTable, 484
 - defun, 484
- checkComments, 480

- calls checkAddPeriod, 481
- calls checkAddSpaces, 481
- calls checkArguments, 481
- calls checkBalance, 481
- calls checkDecorate, 481
- calls checkFixCommonProblems, 481
- calls checkGetArgs, 481
- calls checkGetMargin, 480
- calls checkIeEg, 481
- calls checkIndentedLines, 480
- calls checkSplit2Words, 481
- calls checkTransformFirsts, 480
- calls nequal, 480
- calls newString2Words, 481
- calls pp, 481
- calls strconc, 481
- local def \$argl, 481
- local def \$checkErrorFlag, 481
- local ref \$attribute?, 481
- local ref \$checkErrorFlag, 481
- defun, 480
- checkDecorate, 508
 - calledby checkComments, 481
 - calls checkAddBackSlashes, 508
 - calls checkDocError, 508
 - calls hasNoVowels, 508
 - calls member, 508
 - local ref \$argl, 508
 - local ref \$charBack, 508
 - local ref \$charExclusions, 508
 - local ref \$charLbrace, 508
 - local ref \$charRbrace, 508
 - local ref \$checkingXmptex?, 508
 - defun, 508
- checkDecorateForHt, 477
 - calledby checkRewrite, 471
 - calls checkDocError, 477
 - calls member, 477
 - local ref \$charLbrace, 477
 - local ref \$charRbrace, 477
 - local ref \$checkingXmptex?, 477
 - defun, 477
- checkDocError, 478
 - calledby checkBeginEnd, 484
 - calledby checkDecorateForHt, 477
 - calledby checkDecorate, 508
 - calledby checkDocError1, 478
 - calledby checkFixCommonProblem, 508
 - calledby checkHTargs, 486
 - calledby checkPrenAlist, 483
 - calledby checkRecordHash, 473
 - calledby checkTexht, 476
 - calledby checkTransformFirsts, 488
 - calledby checkTrim, 506
 - calledby transDocList, 469
 - calls checkDocMessage, 478
 - calls concat, 478
 - calls sayBrightly, 479
 - calls saybrightly1, 478
 - local def \$checkErrorFlag, 479
 - local def \$exposeFlagHeading, 479
 - local ref \$checkErrorFlag, 479
 - local ref \$constructorName, 479
 - local ref \$exposeFlagHeading, 479
 - local ref \$exposeFlag, 479
 - local ref \$outStream, 479
 - local ref \$recheckingFlag, 479
 - defun, 478
- checkDocError1, 478
 - calledby transDocList, 469
 - calledby transDoc, 469
 - calls checkDocError, 478
 - local ref \$compileDocumentation, 478
 - defun, 478
- checkDocMessage, 479
 - calledby checkDocError, 478
 - calls concat, 479
 - calls getdatabase, 479
 - calls whoOwns, 479
 - local ref \$constructorName, 479
 - local ref \$x, 479
 - defun, 479
- checkExtract, 507
 - calledby transDoc, 469
 - calls charPosition, 507
 - calls firstNonBlankPosition, 507
 - calls length, 507
 - calls substring?, 507
 - defun, 507
- checkFixCommonProblem, 508
 - calledby checkRewrite, 471
 - calls checkDocError, 508

- calls ifcar, 508
- calls ifcdr, 508
- calls member, 508
- calls nequal, 508
- local ref \$HTspadmacros, 508
- local ref \$charLbrace, 508
- defun, 508
- checkFixCommonProblems
 - calledby checkComments, 481
- checkGetArgs, 504
 - calledby checkComments, 481
 - calledby checkGetArgs, 504
 - calledby checkRewrite, 471
 - calls charPosition, 504
 - calls checkGetArgs, 504
 - calls firstNonBlankPosition, 504
 - calls getMatchingRightPren, 504
 - calls maxindex, 504
 - calls nequal, 504
 - calls stringPrefix?, 504
 - calls trimString, 504
 - local ref \$charComma, 504
 - defun, 504
- checkGetLispFunctionName
 - calledby checkRecordHash, 473
- checkGetMargin, 493
 - calledby checkComments, 480
 - calls firstNonBlankPosition, 493
 - defun, 493
- checkGetParse, 475
 - calledby checkRecordHash, 473
 - calls ncParseFromString, 475
 - calls removeBackslashes, 475
 - defun, 475
- checkGetStringBeforeRightBrace
 - calledby checkRecordHash, 473
- checkHTargs, 486
 - calledby checkArguments, 486
 - calledby checkHTargs, 487
 - calls checkDocError, 486
 - calls checkHTargs, 487
 - calls checkLookForLeftBrace, 486
 - calls checkLookForRightBrace, 486
 - calls ifcdr, 487
 - defun, 486
- checkIeEg, 494
 - calledby checkComments, 481
 - calls checkIeEgfun, 494
 - calls nreverse, 494
 - defun, 494
- checkIeEgFun
 - calledby checkIeEgfun, 494
- checkIeEgfun, 494
 - calledby checkIeEg, 494
 - calls checkIeEgFun, 494
 - calls maxindex, 494
 - local ref \$charPeriod, 494
 - defun, 494
- checkIndentedLines, 502
 - calledby checkComments, 480
 - calls firstNonBlankPosition, 502
 - calls strconc, 502
 - local ref \$charFauxNewline, 502
 - defun, 502
- checkIsValidType
 - calledby checkRecordHash, 473
- checkLookForLeftBrace, 487
 - calledby checkHTargs, 486
 - calledby checkRecordHash, 472
 - calls nequal, 487
 - local ref \$charBlank, 487
 - local ref \$charLbrace, 487
 - defun, 487
- checkLookForRightBrace, 487
 - calledby checkHTargs, 486
 - calledby checkRecordHash, 472
 - local ref \$charLbrace, 487
 - local ref \$charRbrace, 487
 - defun, 487
- checkNumOfArgs, 499
 - calledby checkRecordHash, 473
 - calls abbreviation?, 499
 - calls constructor?, 499
 - calls getdatabase, 499
 - calls opOf, 499
 - defun, 499
- checkPrenAlist
 - calls assoc, 483
 - calls checkBeginEnd, 483
 - calls checkDocError, 483
 - calls checkSayBracket, 483
 - calls nequal, 483

- calls nreverse, 483
- calls rassoc, 483
- checkRecordHash, 472
 - calledby checkRewrite, 471
 - calls checkDocError, 473
 - calls checkGetLispFunctionName, 473
 - calls checkGetParse, 473
 - calls checkGetStringBeforeRightBrace, 473
 - calls checkIsValidType, 473
 - calls checkLookForLeftBrace, 472
 - calls checkLookForRightBrace, 472
 - calls checkNumOfArgs, 473
 - calls form2HtString, 473
 - calls getl, 473
 - calls hget, 473
 - calls hput, 473
 - calls ifcdr, 473
 - calls intern, 473
 - calls member, 472
 - calls opOf, 473
 - calls spadSysChoose, 473
 - local def \$glossHash, 473
 - local def \$htHash, 473
 - local def \$lispHash, 473
 - local def \$sysHash, 473
 - local ref \$HTlinks, 473
 - local ref \$HTlisplinks, 473
 - local ref \$charBack, 473
 - local ref \$currentSysList, 473
 - local ref \$glossHash, 473
 - local ref \$htHash, 473
 - local ref \$lispHash, 473
 - local ref \$name, 473
 - local ref \$origin, 473
 - local ref \$setOptions, 473
 - local ref \$sysHash, 473
 - defun, 472
- checkRemoveComments, 500
 - calledby checkRewrite, 471
 - calls checkTrimCommented, 500
 - defun, 500
- checkRewrite, 471
 - calls checkAddIndented, 471
 - calls checkAddMacros, 471
 - calls checkAddSpaces, 471
 - calls checkArguments, 471
 - calls checkDecorateForHt, 471
 - calls checkFixCommonProblem, 471
 - calls checkGetArgs, 471
 - calls checkRecordHash, 471
 - calls checkRemoveComments, 471
 - calls checkSplit2Words, 471
 - calls checkTexht, 471
 - calls newString2Words, 471
 - local ref \$argl, 471
 - local ref \$checkErrorFlag, 471
 - local ref \$checkingXmptex?, 471
 - defun, 471
- checkSayBracket, 486
 - calledby checkPrenAlist, 483
 - defun, 486
- checkSkipBlanks, 491
 - calledby checkTransformFirsts, 488
 - local ref \$charBlank, 491
 - defun, 491
- checkSkipIdentifierToken, 491
 - calledby checkSkipToken, 492
 - calls checkAlphabetic, 491
 - defun, 491
- checkSkipOpToken, 492
 - calledby checkSkipToken, 492
 - calls checkAlphabetic, 492
 - calls member, 492
 - local ref \$charDelimiters, 492
 - defun, 492
- checkSkipToken, 492
 - calledby checkTransformFirsts, 488
 - calls checkSkipIdentifierToken, 492
 - calls checkSkipOpToken, 492
 - defun, 492
- checkSplit2Words, 482
 - calledby checkComments, 481
 - calledby checkRewrite, 471
 - calls checkSplitBrace, 482
 - defun, 482
- checkSplitBackslash, 496
 - calledby checkSplitBackslash, 496
 - calledby checkSplitBrace, 495
 - calls charPosition, 496
 - calls checkSplitBackslash, 496
 - calls maxindex, 496
 - local ref \$charBack, 496

- defun, 496
- checkSplitBrace, 495
 - calledby checkSplit2Words, 482
 - calledby checkSplitBrace, 495
 - calls charp, 495
 - calls checkSplitBackslash, 495
 - calls checkSplitBrace, 495
 - calls checkSplitOn, 495
 - calls checkSplitPunctuation, 495
 - calls length, 495
 - defun, 495
- checkSplitOn, 498
 - calledby checkSplitBrace, 495
 - calledby checkSplitOn, 498
 - calls charPosition, 498
 - calls charp, 498
 - calls checkSplitOn, 498
 - calls maxindex, 498
 - local ref \$charBack, 499
 - local ref \$charSplitList, 499
 - defun, 498
- checkSplitPunctuation, 497
 - calledby checkSplitBrace, 495
 - calledby checkSplitPunctuation, 497
 - calls charPosition, 497
 - calls charp, 497
 - calls checkSplitPunctuation, 497
 - calls hget, 497
 - calls maxindex, 497
 - local ref \$charBack, 497
 - local ref \$charComma, 497
 - local ref \$charDash, 497
 - local ref \$charPeriod, 497
 - local ref \$charQuote, 497
 - local ref \$charSemiColon, 497
 - local ref \$htMacroTable, 497
 - defun, 497
- checkTexht, 476
 - calledby checkRewrite, 471
 - calls checkDocError, 476
 - calls ifcar, 476
 - calls nequal, 476
 - local ref \$charLbrace, 476
 - local ref \$charRbrace, 476
 - defun, 476
- checkTransformFirsts, 488
 - calledby checkComments, 480
 - calledby checkTransformFirsts, 488
 - calls checkDocError, 488
 - calls checkSkipBlanks, 488
 - calls checkSkipToken, 488
 - calls checkTransformFirsts, 488
 - calls fillerSpaces, 488
 - calls getMatchingRightPren, 488
 - calls getl, 488
 - calls lassoc, 488
 - calls leftTrim, 488
 - calls maxindex, 488
 - calls nequal, 488
 - calls pname, 488
 - calls strconc, 488
 - local ref \$charBack, 488
 - local ref \$checkPrenAlist, 488
 - defun, 488
- checkTrim, 506
 - calledby transDoc, 469
 - calls charPosition, 506
 - calls checkDocError, 506
 - calls nequal, 506
 - calls systemError, 506
 - local ref \$charBlank, 506
 - local ref \$charPlus, 506
 - local ref \$x, 506
 - defun, 506
- checkTrimCommented, 500
 - calledby checkRemoveComments, 500
 - calls htcharPosition, 500
 - calls length, 500
 - calls nequal, 500
 - defun, 500
- checkWarning, 521
 - calledby postCapsule, 367
 - calls concat, 521
 - calls postError, 521
 - defun, 521
- clearClams
 - calledby compileConstructor, 153
- clearConstructorCache
 - calledby compileConstructor1, 153
- coerce, 346
 - calledby coerceExit, 351
 - calledby coerceExtraHard, 349

- calledby coerceable, 350
- calledby compApplication, 574
- calledby compApplyModemap, 239
- calledby compAtSign, 352
- calledby compCase, 268
- calledby compCoerce1, 353
- calledby compCoerce, 352
- calledby compColonInside, 565
- calledby compForm1, 571
- calledby compHas, 302
- calledby compIf, 305
- calledby compIs, 312
- calledby convert, 568
- calls coerceEasy, 346
- calls coerceHard, 346
- calls coerceSubset, 346
- calls isSomeDomainVariable, 346
- calls keyedSystemError, 346
- calls msubst, 346
- calls rplac, 346
- calls stackMessage, 346
- local ref \$InteractiveMode, 346
- local ref \$Rep, 346
- local ref \$fromCoerceable, 346
- defun, 346
- coerceable, 350
 - calledby compFocompFormWithModemap, 581
 - calledby compForm1, 571
 - calls coerce, 350
 - calls pmatch, 350
 - calls sublis, 350
 - local ref \$fromCoerceable, 350
 - defun, 350
- coerceByModemap, 354
 - calledby compCoerce1, 353
 - calls genDeltaEntry, 354
 - calls isSubset, 354
 - calls modeEqual, 354
 - calls qcar, 354
 - calls qcdr, 354
 - defun, 354
- coerceEasy, 346
 - calledby coerce, 346
 - calls modeEqualSubst, 346
 - local ref \$EmptyMode, 346
 - local ref \$Exit, 347
 - local ref \$NoValueMode, 347
 - local ref \$Void, 347
 - defun, 346
- coerceExit, 351
 - calledby compRepeatOrCollect, 323
 - calls coerce, 351
 - calls replaceExitEsc, 351
 - calls resolve, 351
 - local ref \$ExitMode, 351
 - defun, 351
- coerceExtraHard, 349
 - calledby coerceHard, 348
 - calls autoCoerceByModemap, 349
 - calls coerce, 349
 - calls hasType, 349
 - calls isUnionMode, 349
 - calls member, 349
 - calls qcar, 349
 - calls qcdr, 349
 - local ref \$Expression, 349
 - defun, 349
- coerceHard, 348
 - calledby coerce, 346
 - calls coerceExtraHard, 348
 - calls extendsCategoryForm, 348
 - calls getmode, 348
 - calls get, 348
 - calls isCategoryForm, 348
 - calls modeEqual, 348
 - local def \$e, 348
 - local ref \$String, 348
 - local ref \$bootStrapMode, 348
 - local ref \$e, 348
 - defun, 348
- coerceSubset, 347
 - calledby coerce, 346
 - calls eval, 347
 - calls get, 347
 - calls isSubset, 347
 - calls lassoc, 347
 - calls maxSuperType, 347
 - calls msubst, 347
 - calls opOf, 347
 - defun, 347
- collect, 322, 373

- calledby floatexpid, 459
- defplist, 322, 373
- collectAndDeleteAssoc, 465
 - calledby collectComBlock, 465
 - local ref \$comblocklist, 465
 - defun, 465
- collectComBlock, 465
 - calledby recordDocumentation, 464
 - calls collectAndDeleteAssoc, 465
 - local def \$comblocklist, 465
 - defun, 465
- comma2Tuple, 376
 - calledby postComma, 376
 - calledby postConstruct, 377
 - calls postFlatten, 376
 - defun, 376
- comp, 557
 - calledby applyMapping, 562
 - calledby compAdd, 256
 - calledby compApplication, 574
 - calledby compApplyModemap, 239
 - calledby compApply, 563
 - calledby compArgumentsAndTryAgain, 585
 - calledby compAtSign, 352
 - calledby compBoolean, 308
 - calledby compCase1, 268
 - calledby compCoerce1, 353
 - calledby compColonInside, 565
 - calledby compCons1, 279
 - calledby compDefWhereClause, 205
 - calledby compDefineAddSignature, 133
 - calledby compExit, 301
 - calledby compExpressionList, 578
 - calledby compForMode, 315
 - calledby compForm1, 571
 - calledby compFromIf, 305
 - calledby compHasFormat, 303
 - calledby compIs, 312
 - calledby compLeave, 317
 - calledby compList, 570
 - calledby compOrCroak1, 556
 - calledby compPretend, 319
 - calledby compReduce1, 320
 - calledby compRepeatOrCollect, 323
 - calledby compReturn, 325
 - calledby compSeqItem, 328
 - calledby compSubsetCategory, 342
 - calledby compSuchthat, 343
 - calledby compUniquely, 584
 - calledby compVector, 344
 - calledby compWhere, 345
 - calledby compWithMappingMode1, 586
 - calledby compileConstructor1, 153
 - calledby doItIf, 265
 - calledby getSuccessEnvironment, 308
 - calledby outputComp, 337
 - calledby setqSetelt, 334
 - calledby setqSingle, 334
 - calledby spadCompileOrSetq, 152
 - calls compNoStacking, 557
 - local ref \$compStack, 558
 - uses \$exitModeStack, 558
 - defun, 557
- comp-tran
 - calledby compWithMappingMode1, 586
- comp2, 559
 - calledby compNoStacking1, 558
 - calledby compNoStacking, 558
 - calls addDomain, 559
 - calls comp3, 559
 - calls insert, 559
 - calls isDomainForm, 559
 - calls isFunctor, 559
 - calls nequal, 559
 - calls opOf, 559
 - uses \$bootStrapMode, 559
 - uses \$lisplib, 559
 - uses \$packagesUsed, 559
 - defun, 559
- comp3, 560
 - calledby comp2, 559
 - calledby compTypeOf, 564
 - calls addDomain, 560
 - calls applyMapping, 560
 - calls compApply, 560
 - calls compAtom, 560
 - calls compCoerce, 560
 - calls compColon, 560
 - calls compExpression, 560
 - calls compTypeOf, 560
 - calls compWithMappingMode, 560
 - calls getDomainsInScope, 560

- calls getmode, 560
- calls member[5], 560
- calls pname[5], 560
- calls stringPrefix?, 560
- uses \$e, 560
- uses \$insideCompTypeOf, 560
- defun, 560
- compAdd, 255
 - calls NRTgetLocalIndex, 256
 - calls compCapsule, 256
 - calls compOrCroak, 256
 - calls compSubDomain1, 256
 - calls compTuple2Record, 256
 - calls comp, 256
 - calls nreverse0, 256
 - calls qcar, 256
 - calls qcdr, 256
 - uses /editfile, 256
 - uses \$EmptyMode, 256
 - uses \$NRTaddForm, 256
 - uses \$addFormLhs, 256
 - uses \$addForm, 256
 - uses \$bootStrapMode, 256
 - uses \$functorForm, 256
 - uses \$packagesUsed, 256
 - defun, 255
- compAndDefine
 - calledby compileConstructor1, 153
- compApplication, 574
 - calledby compToApply, 573
 - calls coerce, 574
 - calls comp, 574
 - calls eltForm, 574
 - calls encodeItem, 574
 - calls getAbbreviation, 574
 - calls isCategoryForm, 574
 - calls length, 574
 - calls member, 574
 - calls nequal, 574
 - calls resolve, 574
 - calls strconc, 574
 - local ref \$Category, 574
 - local ref \$formatArgList, 574
 - local ref \$form, 574
 - local ref \$op, 574
 - local ref \$prefix, 574
- defun, 574
- compApply, 563
 - calledby comp3, 560
 - calls AddContour, 563
 - calls Pair, 563
 - calls comp, 563
 - calls removeEnv, 563
 - calls resolve, 563
 - local ref \$EmptyMode, 563
 - defun, 563
- compApplyModemap, 239
 - calledby compFocompFormWithModemap, 581
 - calledby getModemap, 239
 - calls coerce, 239
 - calls compMapCond, 239
 - calls comp, 239
 - calls genDeltaEntry, 239
 - calls length, 239
 - calls member, 239
 - calls pmatchWithSl, 239
 - calls sublis, 239
 - local def \$bindings, 239
 - local def \$e, 239
 - local ref \$bindings, 239
 - local ref \$e, 239
 - defun, 239
- compareMode2Arg
 - calledby hasSigInTargetCategory, 298
- compArgumentConditions, 294
 - calledby compDefineCapsuleFunction, 288
 - calls compOrCroak, 294
 - calls msubst, 294
 - local def \$argumentConditionList, 294
 - local ref \$Boolean, 294
 - local ref \$argumentConditionList, 294
 - defun, 294
- compArgumentsAndTryAgain, 585
 - calledby compForm, 571
 - calls compForm1, 585
 - calls comp, 585
 - uses \$EmptyMode, 585
 - defun, 585
- compAtom, 565
 - calledby comp3, 560
 - calls compAtomWithModemap, 565

- calls compList, 565
- calls compSymbol, 565
- calls compVector, 565
- calls convert, 565
- calls get, 565
- calls isSymbol, 565
- calls modeIsAggregateOf, 565
- calls primitiveType, 565, 566
- uses \$Expression, 566
- defun, 565
- compAtomWithModemap, 567
 - calledby compAtom, 565
 - calls convert, 567
 - calls modeEqual, 567
 - calls transImplementation, 567
 - local ref \$NoValueMode, 567
 - defun, 567
- compAtSign, 352
 - calledby compLambda, 315
 - calls addDomain, 352
 - calls coerce, 352
 - calls comp, 352
 - defun, 352
- compBoolean, 308
 - calledby compIf, 305
 - calls comp, 308
 - calls getInverseEnvironment, 308
 - calls getSuccessEnvironment, 308
 - defun, 308
- compCapsule, 258
 - calledby compAdd, 256
 - calledby compSubDomain, 340
 - calls addDomain, 258
 - calls bootStrapError, 258
 - calls compCapsuleInner, 258
 - uses \$bootStrapMode, 258
 - uses \$functorForm, 258
 - uses \$insideExpressionIfTrue, 258
 - uses editfile, 258
 - defun, 258
- compCapsuleInner, 259
 - calledby compCapsule, 258
 - calls addInformation, 259
 - calls compCapsuleItems, 259
 - calls mkpf, 259
 - calls processFunctor, 259
 - uses \$addForm, 259
 - uses \$form, 259
 - uses \$functorLocalParameters, 259
 - uses \$getDomainCode, 259
 - uses \$insideCategoryIfTrue, 259
 - uses \$insideCategoryPackageIfTrue, 259
 - uses \$signature, 259
 - defun, 259
- compCapsuleItems, 260
 - calledby compCapsuleInner, 259
 - calls compSingleCapsuleItem, 260
 - local def \$e, 260
 - local def \$myFunctorBody, 260
 - local def \$signatureOfForm, 260
 - local def \$suffix, 260
 - local def \$top-level, 260
 - local ref \$e, 260
 - local ref \$pred, 260
 - defun, 260
- compCase, 268
 - calls addDomain, 268
 - calls coerce, 268
 - calls compCase1, 268
 - defun, 268
- compCase1, 268
 - calledby compCase, 268
 - calls comp, 268
 - calls getModemapList, 268
 - calls modeEqual, 268
 - calls nreverse0, 268
 - uses \$Boolean, 268
 - uses \$EmptyMode, 268
 - defun, 268
- compCat, 270
 - calls getl, 270
 - defun, 270
- compCategory, 270
 - calls compCategoryItem, 270
 - calls mkExplicitCategoryFunction, 270
 - calls qcar, 270
 - calls qcdr, 270
 - calls resolve, 270
 - calls systemErrorHere, 270
 - local def \$atList, 271
 - local def \$sigList, 271
 - local def \$top-level, 271

- local ref \$atList, 271
 - local ref \$sigList, 271
 - defun, 270
- compCategoryItem, 271
 - calledby compCategoryItem, 271
 - calledby compCategory, 270
 - calls compCategoryItem, 271
 - calls mkpf, 271
 - calls qcar, 271
 - calls qcdr, 271
 - local ref \$atList, 271
 - local ref \$sigList, 271
 - defun, 271
- compCoerce, 352
 - calledby comp3, 560
 - calls addDomain, 352
 - calls coerce, 352
 - calls compCoerce1, 352
 - calls getmode, 352
 - defun, 352
- compCoerce1, 353
 - calledby compCoerce, 352
 - calls coerceByModemap, 353
 - calls coerce, 353
 - calls comp, 353
 - calls mkq, 353
 - calls msubst, 353
 - calls resolve, 353
 - defun, 353
- compColon, 275
 - calledby comp3, 560
 - calledby compColon, 275
 - calledby compMakeDeclaration, 593
 - calls addDomain, 275
 - calls assoc, 275
 - calls compColonInside, 275
 - calls compColon, 275
 - calls eqsubstlist, 275
 - calls genSomeVariable, 275
 - calls getDomainsInScope, 275
 - calls getmode, 275
 - calls isCategoryForm, 275
 - calls isDomainForm, 275
 - calls length, 275
 - calls makeCategoryForm, 275
 - calls member[5], 275
 - calls nreverse0, 275
 - calls put, 275
 - calls systemErrorHere, 275
 - calls take, 275
 - calls unknownTypeError, 275
 - uses \$FormalMapVariableList, 275
 - uses \$bootStrapMode, 275
 - uses \$insideCategoryIfTrue, 275
 - uses \$insideExpressionIfTrue, 275
 - uses \$insideFunctorIfTrue, 275
 - uses \$lhsOfColon, 275
 - uses \$noEnv, 275
 - defun, 275
- compColonInside, 564
 - calledby compColon, 275
 - calls addDomain, 564
 - calls coerce, 565
 - calls comp, 565
 - calls opOf, 565
 - calls stackSemanticError, 565
 - calls stackWarning, 565
 - uses \$EmptyMode, 565
 - uses \$newCompilerUnionFlag, 565
 - defun, 564
- compCons, 278
 - calls compCons1, 278
 - calls compForm, 278
 - defun, 278
- compCons1, 279
 - calledby compCons, 278
 - calls comp, 279
 - calls convert, 279
 - calls qcar, 279
 - calls qcdr, 279
 - uses \$EmptyMode, 279
 - defun, 279
- compConstruct, 280
 - calls compForm, 280
 - calls compList, 280
 - calls compVector, 280
 - calls convert, 280
 - calls getDomainsInScope, 280
 - calls modeIsAggregateOf, 280
 - defun, 280
- compConstructorCategory, 282
 - calls resolve, 282

- uses \$Category, 282
 - defun, 282
- compDefine, 282
 - calls compDefine1, 282
 - uses \$macroIfTrue, 282
 - uses \$packagesUsed, 282
 - uses \$tripleCache, 282
 - uses \$tripleHits, 282
 - defun, 282
- compDefine1, 283
 - calledby compDefine1, 283
 - calledby compDefineCategory1, 137
 - calledby compDefine, 282
 - calls addEmptyCapsuleIfNecessary, 283
 - calls compDefWhereClause, 283
 - calls compDefine1, 283
 - calls compDefineAddSignature, 283
 - calls compDefineCapsuleFunction, 283
 - calls compDefineCategory, 283
 - calls compDefineFunctor, 283
 - calls compInternalFunction, 283
 - calls getAbbreviation, 283
 - calls getSignatureFromMode, 283
 - calls getTargetFromRhs, 283
 - calls giveFormalParametersValues, 283
 - calls isDomainForm, 283
 - calls isMacro, 283
 - calls length, 283
 - calls macroExpand, 283
 - calls stackAndThrow, 283
 - calls strconc, 283
 - uses \$Category, 283
 - uses \$ConstructorNames, 283
 - uses \$EmptyMode, 283
 - uses \$NoValueMode, 283
 - uses \$formalArgList, 283
 - uses \$form, 283
 - uses \$insideCapsuleFunctionIfTrue, 283
 - uses \$insideCategoryIfTrue, 283
 - uses \$insideExpressionIfTrue, 283
 - uses \$insideFunctorIfTrue, 283
 - uses \$insideWhereIfTrue, 283
 - uses \$op, 283
 - uses \$prefix, 283
 - defun, 283
- compDefineAddSignature, 133
 - calledby compDefine1, 283
 - calls assoc, 133
 - calls comp, 133
 - calls getProplist, 133
 - calls hasFullSignature, 133
 - calls lassoc, 133
 - uses \$EmptyMode, 133
 - defun, 133
- compDefineCapsuleFunction, 288
 - calledby compDefine1, 283
 - calls NRTassignCapsuleFunctionSlot, 288
 - calls addArgumentConditions, 288
 - calls addDomain, 288
 - calls addStats, 288
 - calls checkAndDeclare, 288
 - calls compArgumentConditions, 288
 - calls compOrCroak, 288
 - calls compileCases, 288
 - calls formatUnabbreviated, 288
 - calls getArgumentModeOrMoan, 288
 - calls getSignature, 288
 - calls getmode, 288
 - calls get, 288
 - calls giveFormalParametersValues, 288
 - calls hasSigInTargetCategory, 288
 - calls length, 288
 - calls member, 288
 - calls mkq, 288
 - calls profileRecord, 288
 - calls put, 288
 - calls replaceExitEtc, 288
 - calls resolve, 288
 - calls sayBrightly, 288
 - calls stripOffArgumentConditions, 288
 - calls stripOffSubdomainConditions, 288
 - local def \$CapsuleDomainsInScope, 289
 - local def \$CapsuleModemapFrame, 289
 - local def \$argumentConditionList, 289
 - local def \$finalEnv, 289
 - local def \$formalArgList, 289
 - local def \$form, 289
 - local def \$functionLocations, 289
 - local def \$functionStats, 289
 - local def \$initCapsuleErrorCount, 289
 - local def \$insideCapsuleFunctionIfTrue, 289

- local def \$insideExpressionIfTrue, 289
- local def \$op, 289
- local def \$returnMode, 289
- local def \$signatureOfForm, 289
- local ref \$DomainsInScope, 288
- local ref \$compileOnlyCertainItems, 289
- local ref \$formalArgList, 288
- local ref \$functionLocations, 288
- local ref \$functionStats, 289
- local ref \$functorStats, 289
- local ref \$op, 288
- local ref \$profileCompiler, 289
- local ref \$returnMode, 289
- local ref \$semanticErrorStack, 288
- local ref \$signatureOfForm, 288
- defun, 288
- compDefineCategory, 170
 - calledby compDefine1, 283
 - calls compDefineCategory1, 170
 - calls compDefineLisplib, 170
 - uses \$domainShell, 171
 - uses \$insideFunctorIfTrue, 171
 - uses \$lisplibCategory, 171
 - uses \$lisplib, 171
 - defun, 170
- compDefineCategory1, 137
 - calledby compDefineCategory, 170
 - calls compDefine1, 137
 - calls compDefineCategory2, 137
 - calls makeCategoryPredicates, 137
 - calls mkCategoryPackage, 137
 - uses \$EmptyMode, 137
 - uses \$bootStrapMode, 137
 - uses \$categoryPredicateList, 137
 - uses \$insideCategoryPackageIfTrue, 137
 - uses \$lisplibCategory, 137
 - defun, 137
- compDefineCategory2, 142
 - calledby compDefineCategory1, 137
 - calls addBinding, 142
 - calls augLisplibModemapsFromCategory, 142
 - calls compMakeDeclaration, 142
 - calls compOrCroak, 142
 - calls compile, 142
 - calls computeAncestorsOf, 142
 - calls constructor?, 142
 - calls evalAndRwriteLispForm, 142
 - calls eval, 142
 - calls getArgumentModeOrMoan, 142
 - calls getParentsFor, 142
 - calls giveFormalParametersValues, 142
 - calls lisplibWrite, 142
 - calls mkConstructor, 142
 - calls mkq, 142
 - calls nequal, 142
 - calls opOf, 142
 - calls optFunctorBody, 142
 - calls removeZeroOne, 142
 - calls sublis, 142
 - calls take, 142
 - local def \$addForm, 143
 - local def \$definition, 142
 - local def \$domainShell, 143
 - local def \$extraParms, 142
 - local def \$formalArgList, 142
 - local def \$form, 142
 - local def \$frontier, 142
 - local def \$functionStats, 142
 - local def \$functorForm, 143
 - local def \$functorStats, 142
 - local def \$getDomainCode, 142
 - local def \$insideCategoryIfTrue, 142
 - local def \$lisplibAbbreviation, 143
 - local def \$lisplibAncestors, 143
 - local def \$lisplibCategory, 143
 - local def \$lisplibForm, 143
 - local def \$lisplibKind, 143
 - local def \$lisplibModemap, 143
 - local def \$lisplibParents, 143
 - local def \$op, 142
 - local def \$stop-level, 142
 - local ref \$FormalMapVariableList, 142
 - local ref \$TriangleVariableList, 142
 - local ref \$definition, 142
 - local ref \$extraParms, 142
 - local ref \$formalArgList, 142
 - local ref \$form, 142
 - local ref \$libFile, 142
 - local ref \$lisplibCategory, 142
 - local ref \$lisplib, 142
 - local ref \$op, 142

- uses \$prefix, 142
 - defun, 142
- compDefineFunctor, 183
 - calledby compDefine1, 283
 - calls compDefineFunctor1, 183
 - calls compDefineLisplib, 183
 - uses \$domainShell, 183
 - uses \$lisplib, 183
 - uses \$profileAlist, 183
 - uses \$profileCompiler, 183
 - defun, 183
- compDefineFunctor1, 183
 - calledby compDefineFunctor, 183
 - calls NRTgenInitialAttributeAlist, 183
 - calls NRTgetLocalIndex, 183
 - calls NRTgetLookupFunction, 184
 - calls NRTmakeSlot1Info, 184
 - calls augModemapsFromCategoryRep, 184
 - calls augModemapsFromCategory, 184
 - calls augmentLisplibModemapsFromFunctor, 184
 - calls compFunctorBody, 184
 - calls compMakeCategoryObject, 183
 - calls compMakeDeclaration, 183
 - calls compile, 184
 - calls computeAncestorsOf, 184
 - calls constructor?, 184
 - calls disallowNilAttribute, 183
 - calls evalAndRwriteLispForm, 184
 - calls getArgumentModeOrMoan, 183
 - calls getModemap, 183
 - calls getParentsFor, 184
 - calls getdatabase, 184
 - calls giveFormalParametersValues, 183
 - calls isCategoryPackageName, 183, 184
 - calls lisplibWrite, 184
 - calls makeFunctorArgumentParameters, 184
 - calls maxindex, 184
 - calls mkq, 184
 - calls nequal, 184
 - calls pname, 183
 - calls pp, 183
 - calls qcar, 184
 - calls qcdr, 184
 - calls remdup, 183
 - calls removeZeroOne, 184
 - calls reportOnFunctorCompilation, 184
 - calls sayBrightly, 183
 - calls simpBool, 184
 - calls strconc, 183
 - calls sublis, 184
 - uses \$CategoryFrame, 184
 - uses \$CheckVectorList, 184
 - uses \$FormalMapVariableList, 184
 - uses \$LocalDomainAlist, 184
 - uses \$NRTaddForm, 184
 - uses \$NRTaddList, 184
 - uses \$NRTattributeAlist, 184
 - uses \$NRTbase, 184
 - uses \$NRTdeltaLength, 184
 - uses \$NRTdeltaListComp, 184
 - uses \$NRTdeltaList, 184
 - uses \$NRTdomainFormList, 184
 - uses \$NRTloadTimeAlist, 184
 - uses \$NRTslot1Info, 184
 - uses \$NRTslot1PredicateList, 184
 - uses \$QuickCode, 185
 - uses \$Representation, 184
 - uses \$addForm, 184
 - uses \$attributesName, 184
 - uses \$bootStrapMode, 184
 - uses \$byteAddress, 185
 - uses \$byteVec, 185
 - uses \$compileOnlyCertainItems, 185
 - uses \$condAlist, 185
 - uses \$domainShell, 185
 - uses \$form, 185
 - uses \$functionLocations, 185
 - uses \$functionStats, 185
 - uses \$functorForm, 185
 - uses \$functorLocalParameters, 185
 - uses \$functorSpecialCases, 185
 - uses \$functorStats, 185
 - uses \$functorTarget, 185
 - uses \$functorsUsed, 185
 - uses \$genFVar, 185
 - uses \$genSDVar, 185
 - uses \$getDomainCode, 185
 - uses \$goGetList, 185
 - uses \$insideCategoryPackageIfTrue, 185
 - uses \$insideFunctorIfTrue, 185

- uses \$isOpPackageName, 185
- uses \$libFile, 185
- uses \$lisplibAbbreviation, 185
- uses \$lisplibAncestors, 185
- uses \$lisplibCategoriesExtended, 185
- uses \$lisplibCategory, 185
- uses \$lisplibForm, 185
- uses \$lisplibFunctionLocations, 185
- uses \$lisplibKind, 185
- uses \$lisplibMissingFunctions, 185
- uses \$lisplibModemap, 185
- uses \$lisplibOperationAlist, 185
- uses \$lisplibParents, 185
- uses \$lisplibSlot1, 185
- uses \$lisplib, 184
- uses \$lookupFunction, 185
- uses \$mutableDomains, 185
- uses \$mutableDomain, 185
- uses \$myFunctorBody, 185
- uses \$op, 185
- uses \$pairlis, 185
- uses \$setelt, 185
- uses \$signature, 185
- uses \$template, 185
- uses \$top-level, 184
- uses \$uncondAlist, 185
- uses \$viewNames, 185
- defun, 183
- compDefineLisplib, 171
 - calledby compDefineCategory, 170
 - calledby compDefineFunctor, 183
 - calls bright, 171
 - calls compileDocumentation, 171
 - calls filep, 171
 - calls fillerSpaces, 171
 - calls finalizeLisplib, 171
 - calls getConstructorAbbreviation, 171
 - calls getdatabase, 171
 - calls lisplibDoRename, 171
 - calls localdatabase, 171
 - calls rpackfile, 171
 - calls rshut, 171
 - calls sayMSG, 171
 - calls unloadOneConstructor, 171
 - calls updateCategoryFrameForCategory, 171
- calls updateCategoryFrameForConstructor, 171
- local def \$libFile, 172
- local def \$lisplibAbbreviation, 172
- local def \$lisplibAncestors, 172
- local def \$lisplibCategoriesExtended, 172
- local def \$lisplibCategory, 172
- local def \$lisplibForm, 172
- local def \$lisplibKind, 172
- local def \$lisplibModemapAlist, 172
- local def \$lisplibModemap, 172
- local def \$lisplibOperationAlist, 172
- local def \$lisplibParents, 171
- local def \$lisplibPredicates, 172
- local def \$lisplibSlot1, 172
- local def \$lisplibSuperDomain, 172
- local def \$lisplibVariableAlist, 172
- local def \$lisplib, 171
- local def \$newConlist, 172
- local def \$op, 171
- local ref \$algebraOutputStream, 171
- local ref \$compileDocumentation, 171
- local ref \$filep, 171
- local ref \$lisplibKind, 171
- local ref \$newConlist, 171
- local ref \$spadLibFT, 171
- defun, 171
- compDefWhereClause, 204
 - calledby compDefine1, 283
 - calls assocleft, 204
 - calls assocright, 204
 - calls comp, 205
 - calls concat, 204
 - calls delete, 204
 - calls getmode, 204
 - calls lassoc, 204
 - calls listOffIdentifiersIn, 204
 - calls orderByDependency, 204
 - calls pairList, 204
 - calls qcar, 204
 - calls qcdr, 204
 - calls union, 204
 - calls userError, 204
 - uses \$predAlist, 205
 - uses \$sigAlist, 205
 - defun, 204

- compElt, 300
 - calls addDomain, 300
 - calls compForm, 300
 - calls convert, 300
 - calls getDeltaEntry, 300
 - calls getModemapListFromDomain, 300
 - calls isDomainForm, 300
 - calls length, 300
 - calls nequal, 300
 - calls opOf, 300
 - calls stackMessage, 300
 - calls stackWarning, 300
 - uses \$One, 300
 - uses \$Zero, 300
 - defun, 300
- compExit, 301
 - calls comp, 301
 - calls modifyModeStack, 302
 - calls stackMessageIfNone, 302
 - uses \$exitModeStack, 302
 - defun, 301
- compExpression, 571
 - calledby comp3, 560
 - calls compForm, 571
 - calls getl, 571
 - uses \$insideExpressionIfTrue, 571
 - defun, 571
- compExpressionList, 578
 - calledby compForm1, 571
 - calls comp, 578
 - calls convert, 578
 - calls nreverse0, 578
 - local ref \$Expression, 578
 - defun, 578
- compFocompFormWithModemap, 581
 - calls coerceable, 581
 - calls compApplyModemap, 581
 - calls convert, 581
 - calls get, 581
 - calls identp, 581
 - calls isCategoryForm, 581
 - calls isFunctor, 581
 - calls last, 581
 - calls listOfSharpVars, 581
 - calls substituteIntoFunctorModemap, 581
 - local ref \$Category, 581
 - local ref \$FormalMapVariableList, 581
 - defun, 581
- compForm, 571
 - calledby compConstruct, 280
 - calledby compCons, 278
 - calledby compElt, 300
 - calledby compExpression, 571
 - calls compArgumentsAndTryAgain, 571
 - calls compForm1, 571
 - calls stackMessageIfNone, 571
 - defun, 571
- compForm1, 571
 - calledby compArgumentsAndTryAgain, 585
 - calledby compForm, 571
 - calls addDomain, 572
 - calls augModemapsFromDomain1, 572
 - calls coerceable, 571
 - calls coerce, 571
 - calls compExpressionList, 571
 - calls compForm2, 572
 - calls compOrCroak, 571
 - calls compToApply, 572
 - calls comp, 571
 - calls getFormModemaps, 572
 - calls length, 571
 - calls nreverse0, 572
 - calls outputComp, 571
 - uses \$EmptyMode, 572
 - uses \$Expression, 572
 - uses \$NumberOfArgsIfInteger, 572
 - defun, 571
- compForm2, 579
 - calledby compForm1, 572
 - calls PredImplies, 579
 - calls assoc, 579
 - calls compForm3, 579
 - calls compFormPartiallyBottomUp, 579
 - calls compUniquely, 579
 - calls isSimple, 579
 - calls length, 579
 - calls nreverse0, 579
 - calls sublis, 579
 - calls take, 579
 - uses \$EmptyMode, 579
 - uses \$TriangleVariableList, 579
 - defun, 579

- compForm3, 580
 - calledby compForm2, 579
 - calledby compFormPartiallyBottomUp, 584
 - calls compFormWithModemap, 580
 - local ref \$compUniquelyIfTrue, 581
 - defun, 580
 - throws, 580
- compFormMatch, 584
 - calledby compFormPartiallyBottomUp, 584
 - defun, 584
- compFormMode, 315
 - calledby compJoin, 313
 - calls comp, 315
 - local def \$compForModeIfTrue, 315
 - defun, 315
- compFormPartiallyBottomUp, 584
 - calledby compForm2, 579
 - calls compForm3, 584
 - calls compFormMatch, 584
 - defun, 584
- compFormWithModemap
 - calledby compForm3, 580
- compFromIf, 305
 - calledby compIf, 305
 - calls comp, 305
 - defun, 305
- compFuncorBody, 195
 - calledby compDefineFuncor1, 184
 - calls bootStrapError, 195
 - calls compOrCroak, 195
 - uses /editfile, 196
 - uses \$NRTaddForm, 196
 - uses \$bootStrapMode, 196
 - uses \$funcorForm, 196
 - defun, 195
- compHas, 302
 - calls chaseInferences, 302
 - calls coerce, 302
 - calls compHasFormat, 302
 - local def \$e, 302
 - local ref \$Boolean, 303
 - local ref \$e, 302
 - defun, 302
- compHasFormat, 303
 - calledby compHas, 302
 - calls comp, 303
 - calls isDomainForm, 303
 - calls length, 303
 - calls mkDomainConstructor, 303
 - calls mkList, 303
 - calls qcar, 303
 - calls qcdr, 303
 - calls sublislis, 303
 - calls take, 303
 - local ref \$EmptyEnvironment, 303
 - local ref \$EmptyMode, 303
 - local ref \$FormalMapVariableList, 303
 - local ref \$e, 303
 - local ref \$form, 303
 - defun, 303
- compIf, 304
 - calls canReturn, 304
 - calls coerce, 305
 - calls compBoolean, 305
 - calls compFromIf, 305
 - calls intersectionEnvironment, 305
 - calls quotify, 305
 - calls resolve, 305
 - uses \$Boolean, 305
 - defun, 304
- compile, 145
 - calledby compDefineCategory2, 142
 - calledby compDefineFuncor1, 184
 - calledby compileCases, 292
 - calls addStats, 146
 - calls constructMacro, 145
 - calls elapsedTime, 146
 - calls encodeFunctionName, 145
 - calls encodeItem, 145
 - calls getmode, 145
 - calls get, 145
 - calls kar, 145
 - calls member, 145
 - calls modeEqual, 145
 - calls nequal, 145
 - calls optimizeFunctionDef, 145
 - calls printStats, 146
 - calls putInLocalDomainReferences, 145
 - calls qcar, 145
 - calls qcdr, 145
 - calls sayBrightly, 145
 - calls spadCompileOrSetq, 146

- calls splitEncodedFunctionName, 145
- calls strconc, 145
- calls userError, 145
- local def \$functionStats, 146
- local def \$savableItems, 146
- local def \$suffix, 146
- local ref \$compileOnlyCertainItems, 146
- local ref \$doNotCompileJustPrint, 146
- local ref \$e, 146
- local ref \$functionStats, 146
- local ref \$functorForm, 146
- local ref \$insideCapsuleFunctionIfTrue, 146
- local ref \$lisplibItemsAlreadyThere, 146
- local ref \$lisplib, 146
- local ref \$macroIfTrue, 146
- local ref \$prefix, 146
- local ref \$saveableItems, 146
- local ref \$signatureOffForm, 146
- local ref \$splitUpItemsAlreadyThere, 146
- local ref \$suffix, 146
- defun, 145
- compile-lib-file, 598
 - calledby recompile-lib-file-if-necessary, 597
 - defun, 598
- compileCases, 291
 - calledby compDefineCapsuleFunction, 288
 - calls assocleft, 292
 - calls assocright, 292
 - calls compile, 292
 - calls eval, 291
 - calls getSpecialCaseAssoc, 292
 - calls get, 292
 - calls mkpf, 292
 - calls msubst, 292
 - calls outerProduct, 292
 - calls qcar, 291
 - calls qcdr, 292
 - local def \$specialCaseKeyList, 292
 - local ref \$getDomainCode, 292
 - local ref \$insideFunctorIfTrue, 292
 - defun, 291
- compileConstructor, 153
 - calledby spadCompileOrSetq, 152
 - calls clearClams, 153
 - calls compileConstructor1, 153
 - defun, 153
- compileConstructor1, 153
 - calledby compileConstructor, 153
 - calls clearConstructorCache, 153
 - calls compAndDefine, 153
 - calls comp, 153
 - calls getdatabase, 153
 - local def \$clamList, 153
 - local ref \$ConstructorCache, 153
 - local ref \$clamList, 153
 - local ref \$mutableDomain, 153
 - defun, 153
- compiled-function-p
 - calledby subname, 212
- compileDocumentation, 174
 - calledby compDefineLisplib, 171
 - calls finalizeDocumentation, 174
 - calls lisplibWrite, 174
 - calls make-input-filename, 174
 - calls rdefiostream, 174
 - calls replaceFile, 174
 - calls rpackfile, 174
 - calls rshut, 174
 - local ref \$EmptyMode, 174
 - local ref \$e, 174
 - local ref \$fcopy, 174
 - local ref \$spadLibFT, 174
 - defun, 174
- compileFileQuietly, 598
 - uses *standard-output*, 598
 - uses \$InteractiveMode, 598
 - defun, 598
- compiler, 532
 - calls compileSpad2Cmd, 533
 - calls compileSpadLispCmd, 533
 - calls findfile, 533
 - calls helpSpad2Cmd[5], 533
 - calls mergePathnames[5], 533
 - calls namestring[5], 533
 - calls pathnameType[5], 533
 - calls pathname[5], 533
 - calls selectOptionLC[5], 533
 - calls throwKeyedMsg, 533
 - uses /editfile, 533
 - uses \$newConlist, 533
 - uses \$options, 533

- defun, 532
- compilerDoit, 538
 - calledby compileSpad2Cmd, 536
 - calledby compilerDoitWithScreenedLisplib, 358
 - calls /RQ,LIB, 538
 - calls /rf[5], 538
 - calls /rq[5], 538
 - calls member[5], 538
 - calls opOf, 538
 - calls sayBrightly, 538
 - uses \$byConstructors, 538
 - uses \$constructorsSeen, 538
 - defun, 538
- compilerDoitWithScreenedLisplib
 - calledby compileSpad2Cmd, 536
 - calls compilerDoit, 358
 - calls embed, 358
 - calls rwrite, 358
 - calls unembed, 358
 - local ref \$libFile, 358
 - local ref \$saveableItems, 358
- compilerMessage
 - calledby augModemapsFromCategoryRep, 252
 - calledby augModemapsFromCategory, 245
- compileSpad2Cmd, 534
 - calledby compiler, 533
 - calls browserAutoloadOnceTrigger, 536
 - calls compilerDoitWithScreenedLisplib, 536
 - calls compilerDoit, 536
 - calls error, 536
 - calls extendLocalLibdb, 536
 - calls namestring[5], 536
 - calls nequal, 536
 - calls object2String, 536
 - calls pathnameType[5], 536
 - calls pathname[5], 536
 - calls sayKeyedMsg[5], 536
 - calls selectOptionLC[5], 536
 - calls spad2AsTranslatorAutoloadOnceTrigger, 536
 - calls spadPrompt, 536
 - calls strconc, 536
 - calls terminateSystemCommand[5], 536
 - calls throwKeyedMsg, 536
 - calls updateSourceFiles[5], 536
 - uses /editfile, 536
 - uses \$InteractiveMode, 536
 - uses \$QuickCode, 536
 - uses \$QuickLet, 536
 - uses \$compileOnlyCertainItems, 536
 - uses \$f, 536
 - uses \$m, 536
 - uses \$newComp, 536
 - uses \$newConlist, 536
 - uses \$options, 536
 - uses \$scanIfTrue, 536
 - uses \$sourceFileTypes, 536
 - defun, 534
- compileSpadLispCmd, 596
 - calledby compiler, 533
 - calls fnameMake[5], 596
 - calls fnameReadable?[5], 596
 - calls localdatabase[5], 596
 - calls namestring[5], 596
 - calls object2String, 596
 - calls pathnameDirectory[5], 596
 - calls pathnameName[5], 596
 - calls pathnameType[5], 596
 - calls pathname[5], 596
 - calls recompile-lib-file-if-necessary, 596
 - calls sayKeyedMsg[5], 596
 - calls selectOptionLC[5], 596
 - calls spadPrompt, 596
 - calls terminateSystemCommand[5], 596
 - calls throwKeyedMsg, 596
 - uses \$options, 596
 - defun, 596
- compileTimeBindingOf, 217
 - calledby optSpecialCall, 216
 - calls bpiname, 217
 - calls keyedSystemError, 217
 - calls moan, 217
 - defun, 217
- compImport, 312
 - calls addDomain, 312
 - uses \$NoValueMode, 312
 - defun, 312
- compInternalFunction, 287
 - calledby compDefine1, 283
 - calls identp, 287

- calls stackAndThrow, 287
 - defun, 287
- compIs, 312
 - calls coerce, 312
 - calls comp, 312
 - uses \$Boolean, 312
 - uses \$EmptyMode, 312
 - defun, 312
- compIterator
 - calledby compReduce1, 320
 - calledby compRepeatOrCollect, 323
- compJoin, 313
 - calls compForMode, 313
 - calls compJoin, getParms, 313
 - calls convert, 313
 - calls isCategoryForm, 313
 - calls nreverse0, 313
 - calls qcar, 313
 - calls qcdr, 313
 - calls stackSemanticError, 313
 - calls union, 313
 - calls wrapDomainSub, 313
 - uses \$Category, 313
 - defun, 313
- compJoin, getParms
 - calledby compJoin, 313
- compLambda, 315
 - calledby compWithMappingMode1, 586
 - calls argsToSig, 315
 - calls compAtSign, 315
 - calls qcar, 315
 - calls qcdr, 315
 - calls stackAndThrow, 315
 - defun, 315
- compLeave, 317
 - calls comp, 317
 - calls modifyModeStack, 317
 - uses \$exitModeStack, 317
 - uses \$leaveLevelStack, 317
 - defun, 317
- compList, 570
 - calledby compAtom, 565
 - calledby compConstruct, 280
 - calls comp, 570
 - defun, 570
- compMacro, 317
 - calls formatUnabbreviated, 317
 - calls macroExpand, 317
 - calls put, 317
 - calls qcar, 317
 - calls sayBrightly, 317
 - uses \$EmptyMode, 317
 - uses \$NoValueMode, 317
 - uses \$macroIfTrue, 317
 - defun, 317
- compMakeCategoryObject, 182
 - calledby compDefineFunctor1, 183
 - calledby getOperationAlist, 250
 - calledby getSlotFromFunctor, 181
 - calls isCategoryForm, 182
 - calls mkEvaluableCategoryForm, 182
 - local ref \$Category, 182
 - local ref \$e, 182
 - defun, 182
- compMakeDeclaration, 593
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 183
 - calledby compSetq1, 329
 - calledby compSubDomain1, 341
 - calledby compWithMappingMode1, 586
 - calls compColon, 593
 - uses \$insideExpressionIfTrue, 593
 - defun, 593
- compMapCond, 240
 - calledby compApplyModemap, 239
 - calls compMapCond', 240
 - local ref \$bindings, 241
 - defun, 240
- compMapCond', 241
 - calledby compMapCond, 240
 - calls compMapCond", 241
 - calls compMapConfFun, 241
 - calls stackMessage, 241
 - defun, 241
- compMapCond", 241
 - calledby compMapCond", 241
 - calledby compMapCond', 241
 - calls compMapCond", 241
 - calls get, 241
 - calls knownInfo, 241
 - calls stackMessage, 241
 - local ref \$Information, 241

- local ref \$e, 241
 - defun, 241
- compMapCondFun, 243
 - defun, 243
- compMapConfFun
 - calledby compMapCond', 241
- compNoStacking, 558
 - calledby compToApply, 573
 - calledby comp, 557
 - calls comp2, 558
 - calls compNoStacking1, 558
 - local ref \$compStack, 558
 - uses \$EmptyMode, 558
 - uses \$Representation, 558
 - defun, 558
- compNoStacking1, 558
 - calledby compNoStacking, 558
 - calls comp2, 558
 - calls get, 558
 - local ref \$compStack, 558
 - defun, 558
- compOrCroak, 556
 - calledby NRTgetLocalIndex, 192
 - calledby compAdd, 256
 - calledby compArgumentConditions, 294
 - calledby compDefineCapsuleFunction, 288
 - calledby compDefineCategory2, 142
 - calledby compForm1, 571
 - calledby compFunctorBody, 195
 - calledby compRepeatOrCollect, 323
 - calledby compSubDomain1, 341
 - calledby compTopLevel, 554
 - calledby doIt, 261
 - calledby getTargetFromRhs, 135
 - calledby makeCategoryForm, 278
 - calledby mkEvalableCategoryForm, 140
 - calledby substituteIntoFunctorModemap, 583
 - calls compOrCroak1, 556
 - defun, 556
- compOrCroak1, 556
 - calledby compOrCroak, 556
 - calls compOrCroak1,compactify, 556
 - calls comp, 556
 - calls displayComp, 557
 - calls displaySemanticErrors, 557
 - calls mkErrorExpr, 557
 - calls say, 557
 - calls stackSemanticError, 557
 - calls userError, 557
 - local def \$compStack, 557
 - uses \$compErrorMessageStack, 557
 - uses \$exitModeStack, 557
 - uses \$level, 557
 - uses \$scanIfTrue, 557
 - uses \$s, 557
 - catches, 557
 - defun, 556
- compOrCroak1,compactify, 595
 - calledby compOrCroak1,compactify, 595
 - calledby compOrCroak1, 556
 - calls compOrCroak1,compactify, 595
 - calls lassoc, 595
 - defun, 595
- compPretend, 318
 - calls addDomain, 318
 - calls comp, 319
 - calls nequal, 319
 - calls opOf, 319
 - calls stackSemanticError, 319
 - calls stackWarning, 319
 - uses \$EmptyMode, 319
 - uses \$newCompilerUnionFlag, 319
 - defun, 318
- compQuote, 320
 - defun, 320
- compReduce, 320
 - calls compReduce1, 320
 - uses \$formalArgList, 320
 - defun, 320
- compReduce1, 320
 - calledby compReduce, 320
 - calls compIterator, 320
 - calls comp, 320
 - calls getIdentity, 320
 - calls msubst, 321
 - calls nreverse0, 320
 - calls parseTran, 320
 - calls systemError, 320
 - uses \$Boolean, 321
 - uses \$endTestList, 321
 - uses \$e, 321

- uses \$initList, 321
 - uses \$sideEffectsList, 321
 - uses \$until, 321
 - defun, 320
- compRepeatOrCollect, 323
 - calls coerceExit, 323
 - calls compIterator, 323
 - calls compOrCroak, 323
 - calls comp, 323
 - calls length, 323
 - calls modeIsAggregateOf, 323
 - calls msubst, 323
 - calls stackMessage, 323
 - calls , 323
 - uses \$Boolean, 323
 - uses \$NoValueMode, 323
 - uses \$exitModeStack, 323
 - uses \$formalArgList, 323
 - uses \$leaveLevelStack, 323
 - uses \$until, 323
 - defun, 323
- compReturn, 325
 - calls comp, 325
 - calls modifyModeStack, 325
 - calls nequal, 325
 - calls resolve, 325
 - calls stackSemanticError, 325
 - calls userError, 325
 - uses \$exitModeStack, 325
 - uses \$returnMode, 325
 - defun, 325
- compSeq, 326
 - calls compSeq1, 326
 - uses \$exitModeStack, 326
 - defun, 326
- compSeq1, 326
 - calledby compSeq, 326
 - calls compSeqItem, 326
 - calls mkq, 326
 - calls nreverse0, 326
 - calls replaceExitEtc, 326
 - uses \$NoValueMode, 326
 - uses \$exitModeStack, 326
 - uses \$finalEnv, 326
 - uses \$insideExpressionIfTrue, 326
 - defun, 326
- compSeqItem, 328
 - calledby compSeq1, 326
 - calls comp, 328
 - calls macroExpand, 328
 - defun, 328
- compSetq, 329
 - calledby compSetq1, 329
 - calls compSetq1, 329
 - defun, 329
- compSetq1, 329
 - calledby compSetq, 329
 - calledby setqMultipleExplicit, 333
 - calledby setqMultiple, 331
 - calls compMakeDeclaration, 329
 - calls compSetq, 329
 - calls identp[5], 329
 - calls qcar, 329
 - calls qcdr, 330
 - calls setqMultiple, 330
 - calls setqSetelt, 330
 - calls setqSingle, 329
 - uses \$EmptyMode, 330
 - defun, 329
- compSingleCapsuleItem, 261
 - calledby compCapsuleItems, 260
 - calledby doItIf, 265
 - calledby doIt, 261
 - calls doit, 261
 - calls macroExpandInPlace, 261
 - local ref \$e, 261
 - local ref \$pred, 261
 - defun, 261
- compString, 339
 - calls resolve, 339
 - uses \$StringCategory, 339
 - defun, 339
- compSubDomain, 340
 - calls compCapsule, 340
 - calls compSubDomain1, 340
 - uses \$NRTaddForm, 340
 - uses \$addFormLhs, 340
 - uses \$addForm, 340
 - defun, 340
- compSubDomain1, 341
 - calledby compAdd, 256
 - calledby compSubDomain, 340

- calls addDomain, 341
- calls compMakeDeclaration, 341
- calls compOrCroak, 341
- calls evalAndRwriteLispForm, 341
- calls lispize, 341
- calls stackSemanticError, 341
- uses \$Boolean, 341
- uses \$CategoryFrame, 341
- uses \$EmptyMode, 341
- uses \$lisplibSuperDomain, 341
- uses \$op, 341
- defun, 341
- compSubsetCategory, 342
 - calls comp, 342
 - calls msubst, 342
 - calls put, 342
 - uses \$lhsOfColon, 342
 - defun, 342
- compSuchthat, 343
 - calls comp, 343
 - calls put, 343
 - uses \$Boolean, 343
 - defun, 343
- compSymbol, 569
 - calledby compAtom, 565
 - calls NRTgetLocalIndex, 569
 - calls errorRef, 569
 - calls getmode, 569
 - calls get, 569
 - calls isFunction, 569
 - calls member[5], 569
 - calls stackMessage, 569
 - uses \$Boolean, 569
 - uses \$Expression, 569
 - uses \$FormalMapVariableList, 569
 - uses \$NoValueMode, 569
 - uses \$NoValue, 569
 - uses \$Symbol, 569
 - uses \$compForModeIfTrue, 569
 - uses \$formalArgList, 569
 - uses \$functorLocalParameters, 569
 - defun, 569
- compToApply, 573
 - calledby compForm1, 572
 - calls compApplication, 573
 - calls compNoStacking, 573
 - local ref \$EmptyMode, 573
 - defun, 573
- compTopLevel, 554
 - calledby s-process, 551
 - calls compOrCroak, 554
 - calls newComp, 554
 - uses \$NRTderivedTargetIfTrue, 554
 - uses \$compTimeSum, 554
 - uses \$envHashTable, 554
 - uses \$forceAdd, 554
 - uses \$killOptimizeIfTrue, 554
 - uses \$packagesUsed, 554
 - uses \$resolveTimeSum, 554
 - defun, 554
- compTuple2Record, 258
 - calledby compAdd, 256
 - defun, 258
- compTypeOf, 564
 - calledby comp3, 560
 - calls comp3, 564
 - calls eqsubstlist, 564
 - calls get, 564
 - calls put, 564
 - uses \$FormalMapVariableList, 564
 - uses \$insideCompTypeOf, 564
 - defun, 564
- compUniquely, 584
 - calledby compForm2, 579
 - calls comp, 584
 - local def \$compUniquelyIfTrue, 585
 - catches, 584
 - defun, 584
- computeAncestorsOf
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
- compVector, 344
 - calledby compAtom, 565
 - calledby compConstruct, 280
 - calls comp, 344
 - uses \$EmptyVector, 344
 - defun, 344
- compWhere, 345
 - calls addContour, 345
 - calls comp, 345
 - calls deltaContour, 345
 - calls macroExpand, 345

- uses \$EmptyMode, 345
 - uses \$insideExpressionIfTrue, 345
 - uses \$insideWhereIfTrue, 345
 - defun, 345
- compWithMappingMode, 586
 - calledby comp3, 560
 - calls compWithMappingMode1, 586
 - uses \$formalArgList, 586
 - defun, 586
- compWithMappingMode1, 586
 - calledby compWithMappingMode, 586
 - calls comp-tran, 586
 - calls compLambda, 586
 - calls compMakeDeclaration, 586
 - calls comp, 586
 - calls extendsCategoryForm, 586
 - calls extractCodeAndConstructTriple, 586
 - calls freelist, 586
 - calls get, 586
 - calls hasFormalMapVariable, 586
 - calls isFunctor, 586
 - calls optimizeFunctionDef, 586
 - calls qcar, 586
 - calls qcdr, 586
 - calls stackAndThrow, 586
 - calls take, 586
 - uses \$CategoryFrame, 586
 - uses \$EmptyMode, 586
 - uses \$FormalMapVariableList, 586
 - uses \$QuickCode, 586
 - uses \$formalArgList, 586
 - uses \$formatArgList, 586
 - uses \$funnameTail, 586
 - uses \$funname, 586
 - uses \$skillOptimizeIfTrue, 586
 - defun, 586
- concat
 - calledby checkDocError, 478
 - calledby checkDocMessage, 479
 - calledby checkWarning, 521
 - calledby compDefWhereClause, 204
- cond, 226
 - defplist, 226
- cons, 278
 - defplist, 278
- consProplistOf
 - calledby getSuccessEnvironment, 309
 - calledby setqSingle, 334
- construct, 95, 280, 377
 - defplist, 95, 280, 377
- constructMacro, 151
 - calledby compile, 145
 - calls identp, 151
 - calls stackSemanticError, 151
 - defun, 151
- constructor?
 - calledby addDomain, 232
 - calledby checkNumOfArgs, 499
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby getAbbreviation, 285
 - calledby isFunctor, 233
- contained
 - calledby evalAndSub, 249
 - calledby parseCategory, 99
 - calledby spadCompileOrSetq, 152
 - calledby substVars, 168
- containsBang, 398
 - calledby aplTran, 395
 - calledby containsBang, 398
 - calls containsBang, 398
 - defun, 398
- convert, 568
 - calledby applyMapping, 562
 - calledby compAtomWithModemap, 567
 - calledby compAtom, 565
 - calledby compCons1, 279
 - calledby compConstruct, 280
 - calledby compElt, 300
 - calledby compExpressionList, 578
 - calledby compFocompFormWithModemap, 581
 - calledby compJoin, 313
 - calledby convertOrCroak, 328
 - calledby setqMultiple, 331
 - calledby setqSingle, 334
 - calls coerce, 568
 - calls resolve, 568
 - defun, 568
- convertOpAlist2compilerInfo, 112
 - calledby updateCategoryFrameForConstructor, 112

- defun, 112
- convertOrCroak, 328
 - calledby replaceExitEtc, 327
 - calls convert, 328
 - calls userError, 328
 - defun, 328
- copy
 - calledby modifyModeStack, 593
- copy-token
 - calledby PARSE-TokTail, 424
 - calledby advance-token, 456
- croak
 - calledby drop, 525
- curoutstream
 - usedby s-process, 551
 - usedby spad, 549
- current-char, 457
 - calledby Advance-Char, 607
 - calledby PARSE-FloatBasePart, 431
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-Selector, 427
 - calledby PARSE-TokTail, 424
 - calledby match-string, 448
 - calledby skip-blanks, 448
 - uses \$line, 457
 - uses current-line, 457
 - defun, 457
- current-fragment, 601
 - defvar, 601
- current-line, 604
 - usedby PARSE-Category, 418
 - usedby current-char, 457
 - usedby next-char, 458
 - defvar, 604
- current-symbol, 454
 - calledby PARSE-AnyId, 438
 - calledby PARSE-ElseClause, 445
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-Infix, 423
 - calledby PARSE-NewExpr, 411
 - calledby PARSE-OpenBrace, 440
 - calledby PARSE-OpenBracket, 440
 - calledby PARSE-Operation, 421
 - calledby PARSE-Prefix, 422
 - calledby PARSE-Primary1, 428
 - calledby PARSE-ReductionOp, 425
 - calledby PARSE-Selector, 427
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-SpecialKeyWord, 412
 - calledby PARSE-Suffix, 442
 - calledby PARSE-TokTail, 424
 - calledby PARSE-TokenList, 414
 - calledby isTokenDelimiter, 451
 - calls current-token, 454
 - calls make-symbol-of, 454
 - defun, 454
- current-token, 91, 455
 - calledby PARSE-FloatBasePart, 431
 - calledby PARSE-SpecialKeyWord, 412
 - calledby advance-token, 456
 - calledby current-symbol, 454
 - calledby match-advance-string, 449
 - calledby match-current-token, 453
 - calledby next-token, 456
 - calls try-get-token, 455
 - usedby advance-token, 456
 - usedby current-token, 455
 - uses \$token, 91
 - uses current-token, 455
 - uses valid-tokens, 455
 - defun, 455
 - defvar, 91
- curstrm
 - calledby s-process, 550
- dcq
 - calledby preparseReadLine, 85
- decodeScripts, 399
 - calledby decodeScripts, 399
 - calledby getScriptName, 399
 - calls decodeScripts, 399
 - calls qcar, 399
 - calls qcdr, 399
 - calls strconc, 399
 - defun, 399
- deepestExpression, 398
 - calledby deepestExpression, 398
 - calledby hasAplExtension, 397
 - calls deepestExpression, 398
 - defun, 398

- def, 101, 282
 - defplist, 101, 282
- def-process
 - calledby s-process, 551
- def-rename, 553
 - calledby s-process, 550
 - calls def-rename1, 553
 - defun, 553
- def-rename1, 554
 - calledby def-rename1, 554
 - calledby def-rename, 553
 - calls def-rename1, 554
 - defun, 554
- definition-name, 411
 - usedby PARSE-NewExpr, 411
 - defvar, 411
- defmacro
 - Bang, 460
 - line-clear, 604
 - must, 460
 - nth-stack, 524
 - pop-stack-1, 522
 - pop-stack-2, 523
 - pop-stack-3, 523
 - pop-stack-4, 523
 - reduce-stack-clear, 462
 - stack-/-empty, 89
 - star, 461
- defplist, 97, 224, 255, 343, 351, 368
 - + - >, 315
 - >, 384
 - <=, 121
 - ==>, 385
 - =>, 381
 - >, 108
 - >=, 106, 107
 - „, 376
 - , 222
 - /, 391
 - ., 100, 274, 374
 - ::, 99, 352, 375
 - :BF:, 369
 - ;;, 389
 - ==, 378
 - add, 366
 - and, 97
 - Block, 370
 - call, 213
 - capsule, 258
 - case, 267
 - catch, 224
 - category, 98, 270, 371
 - collect, 322, 373
 - cond, 226
 - cons, 278
 - construct, 95, 280, 377
 - def, 101, 282
 - dollargreaterequal, 104
 - dollargreaterthan, 104
 - dollarnotequal, 105
 - elt, 300
 - eq, 220
 - eqv, 106
 - exit, 301
 - has, 108, 302
 - if, 113, 304, 381
 - implies, 116
 - import, 312
 - In, 383
 - in, 117, 382
 - inby, 118
 - is, 119, 312
 - isnt, 119
 - Join, 120, 313, 383
 - leave, 121, 316
 - lessp, 222
 - let, 122, 329
 - letd, 123
 - ListCategory, 281
 - Mapping, 269
 - mdef, 123, 317
 - minus, 220
 - mkRecord, 229
 - not, 124
 - notequal, 125
 - or, 125
 - pretend, 126, 318, 386
 - qsminus, 221
 - quote, 319, 387
 - Record, 269
 - RecordCategory, 281
 - recordcopy, 231

- recordelt, 230
- reduce, 320, 387
- repeat, 322, 388
- return, 127, 325
- Scripts, 388
- segment, 127, 128
- seq, 218, 326
- setq, 329
- setrecordelt, 230
- Signature, 390
- spadcall, 223
- String, 339
- SubDomain, 340
- SubsetCategory, 342
- TupleCollect, 392
- Union, 269
- UnionCategory, 281
- vcons, 129
- vector, 343
- VectorCategory, 281
- where, 129, 344, 393
- with, 394
- defstruct
 - line, 603
 - reduction, 92
 - stack, 88
 - token, 90
- defun
 - /RQ,LIB, 539
 - /rf-1, 540
 - action, 461
 - add-parens-and-semis-to-line, 84
 - addArgumentConditions, 293
 - addclose, 524
 - addConstructorModemaps, 238
 - addDomain, 232
 - addEltModemap, 245
 - addEmptyCapsuleIfNecessary, 134
 - addModemap, 253
 - addModemap0, 254
 - addModemap1, 254
 - addModemapKnown, 253
 - addNewDomain, 236
 - addSuffix, 286
 - Advance-Char, 607
 - advance-token, 456
 - alistSize, 286
 - allLASSOCs, 194
 - aplTran, 395
 - aplTran1, 395
 - aplTranList, 397
 - applyMapping, 562
 - argsToSig, 592
 - assignError, 336
 - AssocBarGensym, 204
 - augLisplibModemapsFromCategory, 157
 - augmentLisplibModemapsFromFunctor, 193
 - augModemapsFromCategory, 245
 - augModemapsFromCategoryRep, 251
 - augModemapsFromDomain, 236
 - augModemapsFromDomain1, 237
 - autoCoerceByModemap, 354
 - blankp, 525
 - bootstrapError, 196
 - bumperrorcount, 517
 - canReturn, 306
 - char-eq, 458
 - char-ne, 458
 - checkAddBackSlashes, 511
 - checkAddMacros, 501
 - checkAddPeriod, 483
 - checkAddSpaces, 512
 - checkAddSpaceSegments, 505
 - checkAlphabetic, 491
 - checkAndDeclare, 298
 - checkArguments, 486
 - checkBalance, 483
 - checkBeginEnd, 484
 - checkComments, 480
 - checkDecorate, 508
 - checkDecorateForHt, 477
 - checkDocError, 478
 - checkDocError1, 478
 - checkDocMessage, 479
 - checkExtract, 507
 - checkFixCommonProblem, 508
 - checkGetArgs, 504
 - checkGetMargin, 493
 - checkGetParse, 475
 - checkHTargs, 486
 - checkIeEg, 494
 - checkIeEgfun, 494

- checkIndentedLines, 502
- checkLookForLeftBrace, 487
- checkLookForRightBrace, 487
- checkNumOfArgs, 499
- checkRecordHash, 472
- checkRemoveComments, 500
- checkRewrite, 471
- checkSayBracket, 486
- checkSkipBlanks, 491
- checkSkipIdentifierToken, 491
- checkSkipOpToken, 492
- checkSkipToken, 492
- checkSplit2Words, 482
- checkSplitBackslash, 496
- checkSplitBrace, 495
- checkSplitOn, 498
- checkSplitPunctuation, 497
- checkTexht, 476
- checkTransformFirsts, 488
- checkTrim, 506
- checkTrimCommented, 500
- checkWarning, 521
- coerce, 346
- coerceable, 350
- coerceByModemap, 354
- coerceEasy, 346
- coerceExit, 351
- coerceExtraHard, 349
- coerceHard, 348
- coerceSubset, 347
- collectAndDeleteAssoc, 465
- collectComBlock, 465
- comma2Tuple, 376
- comp, 557
- comp2, 559
- comp3, 560
- compAdd, 255
- compApplication, 574
- compApply, 563
- compApplyModemap, 239
- compArgumentConditions, 294
- compArgumentsAndTryAgain, 585
- compAtom, 565
- compAtomWithModemap, 567
- compAtSign, 352
- compBoolean, 308
- compCapsule, 258
- compCapsuleInner, 259
- compCapsuleItems, 260
- compCase, 268
- compCase1, 268
- compCat, 270
- compCategory, 270
- compCategoryItem, 271
- compCoerce, 352
- compCoerce1, 353
- compColon, 275
- compColonInside, 564
- compCons, 278
- compCons1, 279
- compConstruct, 280
- compConstructorCategory, 282
- compDefine, 282
- compDefine1, 283
- compDefineAddSignature, 133
- compDefineCapsuleFunction, 288
- compDefineCategory, 170
- compDefineCategory1, 137
- compDefineCategory2, 142
- compDefineFunctor, 183
- compDefineFunctor1, 183
- compDefineLisplib, 171
- compDefWhereClause, 204
- compElt, 300
- compExit, 301
- compExpression, 571
- compExpressionList, 578
- compFocompFormWithModemap, 581
- compForm, 571
- compForm1, 571
- compForm2, 579
- compForm3, 580
- compFormMatch, 584
- compForMode, 315
- compFormPartiallyBottomUp, 584
- compFromIf, 305
- compFunctorBody, 195
- compHas, 302
- compHasFormat, 303
- compIf, 304
- compile, 145
- compile-lib-file, 598

- compileCases, 291
- compileConstructor, 153
- compileConstructor1, 153
- compileDocumentation, 174
- compileFileQuietly, 598
- compiler, 532
- compilerDoit, 538
- compileSpad2Cmd, 534
- compileSpadLispCmd, 596
- compileTimeBindingOf, 217
- compImport, 312
- compInternalFunction, 287
- compIs, 312
- compJoin, 313
- compLambda, 315
- compLeave, 317
- compList, 570
- compMacro, 317
- compMakeCategoryObject, 182
- compMakeDeclaration, 593
- compMapCond, 240
- compMapCond', 241
- compMapCond", 241
- compMapCondFun, 243
- compNoStacking, 558
- compNoStacking1, 558
- compOrCroak, 556
- compOrCroak1, 556
- compOrCroak1,compactify, 595
- compPretend, 318
- compQuote, 320
- compReduce, 320
- compReduce1, 320
- compRepeatOrCollect, 323
- compReturn, 325
- compSeq, 326
- compSeq1, 326
- compSeqItem, 328
- compSetq, 329
- compSetq1, 329
- compSingleCapsuleItem, 261
- compString, 339
- compSubDomain, 340
- compSubDomain1, 341
- compSubsetCategory, 342
- compSuchthat, 343
- compSymbol, 569
- compToApply, 573
- compTopLevel, 554
- compTuple2Record, 258
- compTypeOf, 564
- compUniquely, 584
- compVector, 344
- compWhere, 345
- compWithMappingMode, 586
- compWithMappingMode1, 586
- constructMacro, 151
- containsBang, 398
- convert, 568
- convertOpAlist2compilerInfo, 112
- convertOrCroak, 328
- current-char, 457
- current-symbol, 454
- current-token, 455
- decodeScripts, 399
- deepestExpression, 398
- def-rename, 553
- def-rename1, 554
- disallowNilAttribute, 195
- displayMissingFunctions, 197
- displayPreCompilationErrors, 516
- doIt, 261
- doItIf, 265
- dollarTran, 459
- domainMember, 244
- drop, 525
- eltModemapFilter, 577
- encodeFunctionName, 148
- encodeItem, 150
- EqualBarGensym, 228
- errhuh, 403
- escape-keywords, 451
- escaped, 525
- evalAndRwriteLispForm, 169
- evalAndSub, 249
- extractCodeAndConstructTriple, 591
- finalizeDocumentation, 466
- finalizeLisplib, 176
- fincomblock, 526
- firstNonBlankPosition, 493
- fixUpPredicate, 161
- flattenSignatureList, 159

- floatexpid, 459
- formal2Pattern, 194
- freelist, 594
- genDomainOps, 202
- genDomainView, 201
- genDomainViewList, 201
- genDomainViewList0, 200
- get-a-line, 608
- get-token, 457
- getAbbreviation, 285
- getArgumentMode, 299
- getArgumentModeOrMoan, 156
- getCaps, 150
- getCategoryOpsAndAtts, 178
- getConstructorOpsAndAtts, 178
- getDomainsInScope, 234
- getFormModemaps, 575
- getFunctorOpsAndAtts, 181
- getInverseEnvironment, 310
- getMatchingRightPren, 492
- getModemap, 239
- getModemapList, 244
- getModemapListFromDomain, 244
- getOperationAlist, 250
- getScriptName, 399
- getSignature, 296
- getSignatureFromMode, 287
- getSlotFromCategoryForm, 179
- getSlotFromFunctor, 181
- getSpecialCaseAssoc, 293
- getSuccessEnvironment, 308
- getTargetFromRhs, 135
- getToken, 452
- getUnionMode, 311
- getUniqueModemap, 243
- getUniqueSignature, 243
- giveFormalParametersValues, 135
- hackforis, 402
- hackforis1, 403
- hasAplExtension, 397
- hasFormalMapVariable, 592
- hasFullSignature, 134
- hasNoVowels, 511
- hasSigInTargetCategory, 298
- hasType, 350
- htcharPosition, 501
- indent-pos, 526
- infixtok, 527
- initial-substring, 607
- initial-substring-p, 450
- initialize-prepare, 73
- initializeLisplib, 175
- insertModemap, 247
- interactiveModemapForm, 160
- is-console, 527
- isCategoryPackageName, 190
- isDomainConstructorForm, 339
- isDomainForm, 338
- isDomainSubst, 166
- isFunctor, 233
- isListConstructor, 103
- isMacro, 267
- isSuperDomain, 235
- isTokenDelimiter, 451
- isUnionMode, 311
- killColons, 391
- line-advance-char, 605
- line-at-end-p, 604
- line-current-segment, 606
- line-new-line, 606
- line-next-char, 605
- line-past-end-p, 605
- line-print, 604
- lispize, 342
- lisplibDoRename, 174
- lisplibWrite, 182
- loadIfNecessary, 111
- loadLibIfNecessary, 111
- macroExpand, 136
- macroExpandInPlace, 136
- macroExpandList, 137
- make-symbol-of, 454
- makeCategoryForm, 278
- makeCategoryPredicates, 138
- makeFunctorArgumentParameters, 198
- makeSimplePredicateOrNil, 518
- match-advance-string, 449
- match-current-token, 453
- match-next-token, 454
- match-string, 448
- match-token, 454
- maxSuperType, 338

- mergeModemap, 248
- mergeSignatureAndLocalVarAlists, 182
- meta-syntax-error, 459
- mkAbbrev, 286
- mkAlistOfExplicitCategoryOps, 158
- mkCategoryPackage, 139
- mkConstructor, 170
- mkDatabasePred, 195
- mkEvalableCategoryForm, 140
- mkExplicitCategoryFunction, 273
- mkList, 304
- mkNewModemapList, 246
- mkOpVec, 203
- mkRepititionAssoc, 149
- mkUnion, 356
- modeEqual, 357
- modeEqualSubst, 357
- modemapPattern, 169
- modifyModeStack, 593
- moveORsOutside, 167
- mustInstantiate, 274
- ncINTERPFILE, 595
- new2OldLisp, 518
- newString2Words, 503
- newWordFrom, 503
- next-char, 457
- next-line, 606
- next-tab-loc, 527
- next-token, 456
- nonblankloc, 528
- NRTassocIndex, 336
- NRTgetLocalIndex, 192
- NRTgetLookupFunction, 191
- NRTputInHead, 155
- NRTputInTail, 154
- opt-, 222
- optCall, 214
- optCallEval, 218
- optCallSpecially, 215
- optCatch, 225
- optCond, 227
- optCONDtail, 211
- optEQ, 220
- optIF2COND, 212
- optimize, 209
- optimizeFunctionDef, 208
- optional, 461
- optLESSP, 222
- optMINUS, 221
- optMkRecord, 229
- optPackageCall, 215
- optPredicateIfTrue, 211
- optQSMINUS, 221
- optRECORDCOPY, 231
- optRECORDELT, 230
- optSEQ, 218
- optSETRECORDELT, 231
- optSPADCALL, 223
- optSpecialCall, 216
- optSuchthat, 224
- optXLAMCond, 210
- orderByDependency, 207
- orderPredicateItems, 162
- orderPredTran, 163
- outputComp, 337
- PARSE-AnyId, 438
- PARSE-Application, 426
- parse-argument-designator, 520
- PARSE-Category, 417
- PARSE-Command, 412
- PARSE-CommandTail, 415
- PARSE-Conditional, 445
- PARSE-Data, 436
- PARSE-ElseClause, 445
- PARSE-Enclosure, 432
- PARSE-Exit, 443
- PARSE-Expr, 420
- PARSE-Expression, 419
- PARSE-Float, 429
- PARSE-FloatBase, 430
- PARSE-FloatBasePart, 430
- PARSE-FloatExponent, 431
- PARSE-FloatTok, 447
- PARSE-Form, 425
- PARSE-FormalParameter, 433
- PARSE-FormalParameterTok, 433
- PARSE-getSemanticForm, 422
- PARSE-GlyphTok, 438
- parse-identifier, 519
- PARSE-Import, 419
- PARSE-Infix, 423
- PARSE-InfixWith, 417

- PARSE-IntegerTok, 432
- PARSE-Iterator, 441
- PARSE-IteratorTail, 441
- parse-keyword, 520
- PARSE-Label, 427
- PARSE-LabelExpr, 446
- PARSE-Leave, 444
- PARSE-LedPart, 420
- PARSE-leftBindingPowerOf, 421
- PARSE-Loop, 446
- PARSE-Name, 435
- PARSE-NBGliphTok, 437
- PARSE-NewExpr, 411
- PARSE-NudPart, 420
- parse-number, 520
- PARSE-OpenBrace, 440
- PARSE-OpenBracket, 440
- PARSE-Operation, 421
- PARSE-Option, 416
- PARSE-Prefix, 422
- PARSE-Primary, 428
- PARSE-Primary1, 428
- PARSE-PrimaryNoFloat, 428
- PARSE-PrimaryOrQM, 415
- PARSE-Quad, 433
- PARSE-Qualification, 424
- PARSE-Reduction, 425
- PARSE-ReductionOp, 425
- PARSE-Return, 443
- PARSE-rightBindingPowerOf, 422
- PARSE-ScriptItem, 435
- PARSE-Scripts, 434
- PARSE-Seg, 444
- PARSE-Selector, 427
- PARSE-SemiColon, 443
- PARSE-Sequence, 439
- PARSE-Sequence1, 439
- PARSE-Sexpr, 436
- PARSE-Sexpr1, 436
- parse-spadstring, 518
- PARSE-SpecialCommand, 413
- PARSE-SpecialKeyWord, 412
- PARSE-Statement, 416
- PARSE-String, 433
- parse-string, 519
- PARSE-Suffix, 442
- PARSE-TokenCommandTail, 413
- PARSE-TokenList, 414
- PARSE-TokenOption, 414
- PARSE-TokTail, 424
- PARSE-VarForm, 434
- PARSE-With, 417
- parseAnd, 97
- parseAtom, 94
- parseAtSign, 98
- parseCategory, 99
- parseCoerce, 100
- parseColon, 100
- parseConstruct, 95
- parseDEF, 101
- parseDollarGreaterEqual, 104
- parseDollarGreaterThan, 104
- parseDollarLessEqual, 105
- parseDollarNotEqual, 105
- parseDropAssertions, 99
- parseEquivalence, 106
- parseExit, 107
- parseGreaterEqual, 107
- parseGreaterThan, 108
- parseHas, 108
- parseHasRhs, 110
- parseIf, 114
- parseIf,ifTran, 114
- parseImplies, 116
- parseIn, 117
- parseInBy, 118
- parseIs, 119
- parseIsnt, 120
- parseJoin, 120
- parseLeave, 121
- parseLessEqual, 122
- parseLET, 122
- parseLETD, 123
- parseLhs, 102
- parseMDEF, 123
- parseNot, 124
- parseNotEqual, 125
- parseOr, 125
- parsepiles, 84
- parsePretend, 126
- parseprint, 528
- parseReturn, 127

- parseSegment, 128
- parseSeq, 128
- parseTran, 93
- parseTranCheckForRecord, 517
- parseTranList, 95
- parseTransform, 93
- parseType, 98
- parseVCONS, 129
- parseWhere, 129
- Pop-Reduction, 524
- postAdd, 366
- postAtom, 361
- postAtSign, 369
- postBigFloat, 370
- postBlock, 370
- postBlockItem, 368
- postBlockItemList, 367
- postCapsule, 367
- postCategory, 371
- postcheck, 363
- postCollect, 373
- postCollect,finish, 372
- postColon, 375
- postColonColon, 375
- postComma, 376
- postConstruct, 377
- postDef, 378
- postDefArgs, 380
- postError, 364
- postExit, 381
- postFlatten, 376
- postFlattenLeft, 389
- postForm, 364
- postIf, 381
- postIn, 383
- postin, 382
- postInSeq, 382
- postIteratorList, 374
- postJoin, 384
- postMakeCons, 372
- postMapping, 384
- postMDef, 385
- postOp, 361
- postPretend, 386
- postQUOTE, 387
- postReduce, 387
- postRepeat, 388
- postScripts, 389
- postScriptsForm, 362
- postSemiColon, 389
- postSignature, 390
- postSlash, 391
- postTran, 360
- postTranList, 362
- postTranScripts, 362
- postTranSegment, 378
- postTransform, 359
- postTransformCheck, 363
- postTuple, 392
- postTupleCollect, 393
- postType, 369
- postWhere, 393
- postWith, 394
- preparse, 76
- preparse-echo, 88
- preparse1, 81
- preparseReadLine, 85
- preparseReadLine1, 87
- primitiveType, 568
- print-defun, 553
- print-package, 521
- processFunctor, 259
- push-reduction, 462
- putDomainsInScope, 235
- putInLocalDomainReferences, 154
- quote-if-string, 450
- read-a-line, 601
- recompile-lib-file-if-necessary, 597
- recordAttributeDocumentation, 463
- recordDocumentation, 464
- recordHeaderDocumentation, 464
- recordSignatureDocumentation, 463
- removeBackslashes, 476
- removeSuperfluousMapping, 391
- replaceExitEtc, 327
- replaceVars, 161
- reportOnFunctorCompilation, 197
- resolve, 356
- rwriteLispForm, 170
- s-process, 550
- setDefOp, 394
- seteltModemapFilter, 577

- setqMultiple, 330
- setqMultipleExplicit, 333
- setqSetelt, 334
- setqSingle, 334
- signatureTran, 163
- skip-blanks, 448
- skip-ifblock, 86
- skip-to-endif, 528
- spad, 549
- spad-fixed-arg, 598
- spadCompileOrSetq, 151
- splitEncodedFunctionName, 149
- stack-clear, 89
- stack-load, 89
- stack-pop, 90
- stack-push, 89
- storeblanks, 607
- stripOffArgumentConditions, 296
- stripOffSubdomainConditions, 295
- subname, 212
- substituteCategoryArguments, 237
- substituteIntoFunctorModemap, 583
- substNames, 251
- substVars, 168
- token-install, 92
- token-lookahead-type, 449
- token-print, 92
- transDoc, 469
- transDocList, 468
- transformAndRecheckComments, 470
- transformOperationAlist, 179
- transImplementation, 567
- transIs, 102
- transIs1, 102
- translabel, 515
- translabel1, 515
- TruthP, 249
- try-get-token, 455
- tuple2List, 521
- uncons, 330
- underscore, 452
- unget-tokens, 452
- unknownTypeError, 233
- unloadOneConstructor, 173
- unTuple, 403
- updateCategoryFrameForCategory, 113
- updateCategoryFrameForConstructor, 112
- whoOwns, 480
- wrapDomainSub, 274
- writeLib1, 176
- defvar
 - \$BasicPredicates, 211
 - \$EmptyMode, 131
 - \$FormalMapVariableList, 250
 - \$NoValueMode, 131
 - \$byConstructors, 599
 - \$comblocklist, 525
 - \$constructorsSeen, 599
 - \$defstack, 401
 - \$echolinestack, 72
 - \$index, 72
 - \$is-eqlist, 402
 - \$is-gensymlist, 402
 - \$is-spill-list, 401
 - \$is-spill, 401
 - \$linelist, 72
 - \$prepare-last-line, 72
 - \$vl, 402
 - current-fragment, 601
 - current-line, 604
 - current-token, 91
 - definition-name, 411
 - initial-gensym, 402
 - lablasoc, 411
 - meta-error-handler, 458
 - next-token, 91
 - nonblank, 91
 - ParseMode, 411
 - prior-token, 90
 - reduce-stack, 462
 - tmptok, 410
 - tok, 410
 - valid-tokens, 91
 - XTokenReader, 457
- delete
 - calledby compDefWhereClause, 204
 - calledby getInverseEnvironment, 310
 - calledby orderPredTran, 163
 - calledby putDomainsInScope, 235
- deltaContour
 - calledby compWhere, 345
- digitp[5]

- called by PARSE-FloatBasePart, 431
 - called by PARSE-FloatBase, 430
 - called by floatexpid, 459
- disallowNilAttribute, 195
 - calledby compDefineFunctor1, 183
 - defun, 195
- displayComp
 - calledby compOrCroak1, 557
- displayMissingFunctions, 197
 - calledby reportOnFunctorCompilation, 197
 - calls bright, 197
 - calls formatUnabbreviatedSig, 197
 - calls getmode, 197
 - calls member, 197
 - calls sayBrightly, 197
 - uses \$CheckVectorList, 198
 - uses \$env, 198
 - uses \$formalArgList, 198
 - defun, 197
- displayPreCompilationErrors, 516
 - calledby s-process, 550
 - calls length, 516
 - calls nequal, 516
 - calls remdup, 516
 - calls sayBrightly, 516
 - calls sayMath, 516
 - local ref \$InteractiveMode, 516
 - local ref \$postStack, 516
 - local ref \$stopOp, 516
 - defun, 516
- displaySemanticErrors
 - calledby compOrCroak1, 557
 - calledby reportOnFunctorCompilation, 197
 - calledby s-process, 551
- displayWarnings
 - calledby reportOnFunctorCompilation, 197
- doIt, 261
 - calledby doIt, 261
 - calls NRTgetLocalIndex, 261
 - calls bright, 261
 - calls cannotDo, 261
 - calls compOrCroak, 261
 - calls compSingleCapsuleItem, 261
 - calls doItIf, 261
 - calls doIt, 261
 - calls formatUnabbreviated, 261
- calls get, 261
- calls insert, 261
- calls isDomainForm, 261
- calls isMacro, 261
- calls kar, 261
- calls lastnode, 261
- calls member, 261
- calls opOf, 261
- calls put, 261
- calls qcar, 261
- calls qcdr, 261
- calls sayBrightly, 261
- calls stackSemanticError, 261
- calls stackWarning, 261
- calls sublis, 261
- local def \$LocalDomainAlist, 262
- local def \$Representation, 262
- local def \$e, 262
- local def \$functorLocalParameters, 262
- local def \$functorsUsed, 262
- local def \$genno, 262
- local def \$packagesUsed, 262
- local ref \$EmptyMode, 262
- local ref \$LocalDomainAlist, 262
- local ref \$NRTopt, 262
- local ref \$NonMentionableDomainNames, 262
- local ref \$QuickCode, 262
- local ref \$Representation, 262
- local ref \$e, 262
- local ref \$functorLocalParameters, 262
- local ref \$functorsUsed, 262
- local ref \$packagesUsed, 262
- local ref \$predl, 261
- local ref \$signatureOfForm, 262
- defun, 261
- doit
 - calledby compSingleCapsuleItem, 261
- doItIf, 265
 - calledby doIt, 261
 - calls compSingleCapsuleItem, 265
 - calls comp, 265
 - calls getSuccessEnvironment, 265
 - calls localExtras, 265
 - calls rplaca, 265
 - calls rplacd, 265

- calls `userError`, 265
- local def `$e`, 265
- local def `$functorLocalParameters`, 265
- local ref `$Boolean`, 265
- local ref `$e`, 265
- local ref `$functorLocalParameters`, 265
- local ref `$getDomainCode`, 265
- local ref `$predl`, 265
- defun, 265
- `dollargreaterequal`, 104
 - defplist, 104
- `dollargreaterthan`, 104
 - defplist, 104
- `dollarnotequal`, 105
 - defplist, 105
- `dollarTran`, 459
 - calledby `PARSE-Qualification`, 424
 - uses `$InteractiveMode`, 459
 - defun, 459
- `domainMember`, 244
 - calledby `addDomain`, 232
 - calls `modeEqual`, 244
 - defun, 244
- `doSystemCommand`[5]
 - called by `prepare1`, 81
- `drop`, 525
 - calledby `add-parens-and-semis-to-line`, 84
 - calledby `drop`, 525
 - calls `croak`, 525
 - calls `drop`, 525
 - calls `take`, 525
 - defun, 525
- Echo-Meta
 - usedby `preparse-echo`, 88
- `echo-meta`
 - usedby `/rf-1`, 540
 - usedby `spad`, 549
- `echo-meta`[5]
 - called by `/RQ.LIB`, 539
- `editfile`
 - usedby `compCapsule`, 258
- `elapsedTime`
 - calledby `compile`, 146
- `elemn`
 - calledby `PARSE-Operation`, 421
- calledby `PARSE-leftBindingPowerOf`, 422
- calledby `PARSE-rightBindingPowerOf`, 422
- `elt`, 300
 - defplist, 300
- `eltForm`
 - calledby `compApplication`, 574
- `eltModemapFilter`, 577
 - calledby `getFormModemaps`, 576
 - calls `isConstantId`, 577
 - calls `qcar`, 577
 - calls `qcdr`, 577
 - calls `stackMessage`, 577
 - defun, 577
- `embed`
 - calledby `compilerDoitWithScreenedLisplib`, 358
- `encodeFunctionName`, 148
 - calledby `compile`, 145
 - calls `encodeItem`, 148
 - calls `getAbbreviation`, 148
 - calls `internl`, 148
 - calls `length`, 148
 - calls `mkRepititionAssoc`, 148
 - calls `msubst`, 148
 - calls `stringimage`, 148
 - local def `$lisplibSignatureAlist`, 148
 - local ref `$lisplibSignatureAlist`, 148
 - local ref `$lisplib`, 148
 - defun, 148
- `encodeItem`, 150
 - calledby `applyMapping`, 562
 - calledby `compApplication`, 574
 - calledby `compile`, 145
 - calledby `encodeFunctionName`, 148
 - calls `getCaps`, 150
 - calls `identp`, 150
 - calls `pname`, 150
 - calls `qcar`, 150
 - calls `stringimage`, 150
 - defun, 150
- `eq`, 220
 - defplist, 220
- `eqcar`
 - calledby `PARSE-OpenBrace`, 440
 - calledby `PARSE-OpenBracket`, 440
 - calledby `getToken`, 452

- calledby hackforis1, 403
- eqsubstlist
 - calledby compColon, 275
 - calledby compTypeOf, 564
 - calledby getSignatureFromMode, 287
 - calledby isDomainConstructorForm, 339
 - calledby substNames, 251
 - calledby substituteIntoFunctorModemap, 583
- EqualBarGensym, 228
 - calledby AssocBarGensym, 204
 - calledby optCond, 227
 - calls gensymp, 228
 - local def \$GensymAssoc, 228
 - local ref \$GensymAssoc, 228
 - defun, 228
- eqv, 106
 - defplist, 106
- erase
 - calledby initializeLisplib, 175
- errhuh, 403
 - calls systemError, 403
 - defun, 403
- error
 - calledby compileSpad2Cmd, 536
 - calledby processFunctor, 259
- errorRef
 - calledby compSymbol, 569
- errors
 - usedby initializeLisplib, 175
- Escape-Character
 - usedby token-lookahead-type, 449
- escape-keywords, 451
 - calledby quote-if-string, 450
 - local ref \$keywords, 451
 - defun, 451
- escaped, 525
 - calledby preparse1, 81
 - defun, 525
- eval
 - calledby coerceSubset, 347
 - calledby compDefineCategory2, 142
 - calledby compileCases, 291
 - calledby evalAndRwriteLispForm, 169
 - calledby getSlotFromCategoryForm, 179
 - calledby optCallEval, 218
 - evalAndRwriteLispForm, 169
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby compSubDomain1, 341
 - calls eval, 169
 - calls rwriteLispForm, 169
 - defun, 169
 - evalAndSub, 249
 - calledby augModemapsFromCategoryRep, 251
 - calledby augModemapsFromCategory, 245
 - calls contained, 249
 - calls getOperationAlist, 249
 - calls get, 249
 - calls isCategory, 249
 - calls put, 249
 - calls substNames, 249
 - local def \$lhsOfColon, 249
 - defun, 249
- exit, 301
 - defplist, 301
- expand-tabs
 - calledby preparseReadLine1, 87
- extendLocalLibdb
 - calledby compileSpad2Cmd, 536
- extendsCategoryForm
 - calledby coerceHard, 348
 - calledby compWithMappingMode1, 586
- extractCodeAndConstructTriple, 591
 - calledby compWithMappingMode1, 586
 - defun, 591
- FactoredForm
 - calledby optCallEval, 218
- File-Closed
 - usedby read-a-line, 601
- file-closed
 - usedby spad, 549
- filep
 - calledby compDefineLisplib, 171
- fillerSpaces
 - calledby checkTransformFirsts, 488
 - calledby compDefineLisplib, 171
- finalizeDocumentation, 466
 - calledby compileDocumentation, 174
 - calledby finalizeLisplib, 176

- calls assocleft, 466
- calls bright, 466
- calls form2String, 466
- calls formatOpSignature, 466
- calls macroExpand, 466
- calls msubst, 466
- calls remdup, 466
- calls sayKeyedMsg, 466
- calls sayMSG, 466
- calls strconc, 466
- calls stringimage, 466
- calls sublis, 466
- calls transDocList, 466
- local ref \$FormalMapVariableList, 466
- local ref \$comblocklist, 466
- local ref \$docList, 466
- local ref \$e, 466
- local ref \$lisplibForm, 466
- local ref \$op, 466
- defun, 466
- finalizeLisplib, 176
 - calledby compDefineLisplib, 171
 - calls NRTgenInitialAttributeAlist, 176
 - calls finalizeDocumentation, 176
 - calls getConstructorOpsAndAtts, 176
 - calls lisplibWrite, 176
 - calls mergeSignatureAndLocalVarAlists, 176
 - calls namestring, 176
 - calls profileWrite, 176
 - calls removeZeroOne, 176
 - calls sayMSG, 176
 - local def \$NRTslot1PredicateList, 177
 - local def \$lisplibCategory, 177
 - local def \$pairlis, 177
 - local ref \$/editfile, 176
 - local ref \$FormalMapVariableList, 177
 - local ref \$libFile, 176
 - local ref \$lisplibAbbreviation, 177
 - local ref \$lisplibAncestors, 177
 - local ref \$lisplibAttributes, 177
 - local ref \$lisplibCategory, 176
 - local ref \$lisplibForm, 176
 - local ref \$lisplibKind, 176
 - local ref \$lisplibModemapAlist, 176
 - local ref \$lisplibModemap, 176, 177
 - local ref \$lisplibParents, 177
 - local ref \$lisplibPredicates, 177
 - local ref \$lisplibSignatureAlist, 177
 - local ref \$lisplibSlot1, 177
 - local ref \$lisplibSuperDomain, 177
 - local ref \$lisplibVariableAlist, 177
 - local ref \$profileCompiler, 177
 - local ref \$spadLibFT, 177
 - defun, 176
- fincomblock, 526
 - calledby prepare1, 81
 - calls prepare-echo, 526
 - uses \$EchoLineStack, 526
 - uses \$comblocklist, 526
 - defun, 526
- findfile
 - calledby compiler, 533
- firstNonBlankPosition, 493
 - calledby checkExtract, 507
 - calledby checkGetArgs, 504
 - calledby checkGetMargin, 493
 - calledby checkIndentedLines, 502
 - calls maxindex, 493
 - calls nequal, 493
 - defun, 493
- fixUpPredicate, 161
 - calledby interactiveModemapForm, 160
 - calls length, 161
 - calls moveORsOutside, 162
 - calls orderPredicateItems, 161
 - calls qcar, 161
 - calls qcdr, 161
 - defun, 161
- flattenSignatureList, 159
 - calledby flattenSignatureList, 159
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calls flattenSignatureList, 159
 - calls qcar, 159
 - calls qcdr, 159
 - defun, 159
- floatexpid, 459
 - calledby PARSE-FloatExponent, 431
 - calls collect, 459
 - calls digitp[5], 459
 - calls identp[5], 459

- calls maxindex, 459
- calls pname[5], 459
- calls spadreduce, 459
- calls step, 459
- defun, 459
- fnameMake[5]
 - called by compileSpadLispCmd, 596
- fnameReadable?[5]
 - called by compileSpadLispCmd, 596
- form2HtString
 - calledby checkRecordHash, 473
- form2String
 - calledby NRTgetLookupFunction, 191
 - calledby finalizeDocumentation, 466
- formal2Pattern, 194
 - calledby augmentLisplibModemapsFrom-Functor, 193
 - calls pairList, 194
 - calls sublis, 194
 - local ref \$PatternVariableList, 195
 - defun, 194
- formatOpSignature
 - calledby finalizeDocumentation, 466
- formatUnabbreviated
 - calledby compDefineCapsuleFunction, 288
 - calledby compMacro, 317
 - calledby doIt, 261
- formatUnabbreviatedSig
 - calledby displayMissingFunctions, 197
- fp-output-stream
 - calledby is-console, 527
- freelist, 594
 - calledby compWithMappingModel, 586
 - calledby freelist, 594
 - calls assq[5], 594
 - calls freelist, 594
 - calls getmode, 594
 - calls identp[5], 594
 - calls unionq, 594
 - defun, 594
- function
 - calledby optSpecialCall, 216
- functionp
 - calledby loadLibIfNecessary, 111
- genDeltaEntry
 - calledby coerceByModemap, 354
 - calledby compApplyModemap, 239
 - calledby transImplementation, 567
- genDomainOps, 202
 - calledby genDomainView, 201
 - calls addModemap, 202
 - calls getOperationAlist, 202
 - calls mkDomainConstructor, 202
 - calls mkq, 202
 - calls substNames, 202
 - uses \$ConditionalOperators, 202
 - uses \$e, 202
 - uses \$getDomainCode, 202
 - defun, 202
- genDomainView, 201
 - calledby genDomainViewList, 201
 - calls augModemapsFromCategory, 201
 - calls genDomainOps, 201
 - calls member, 201
 - calls mkDomainConstructor, 201
 - calls qcar, 201
 - calls qcdr, 201
 - uses \$e, 201
 - uses \$getDomainCode, 201
 - defun, 201
- genDomainViewList, 201
 - calledby genDomainViewList, 201
 - calls genDomainViewList, 201
 - calls genDomainView, 201
 - calls isCategoryForm, 201
 - calls qcdr, 201
 - uses \$EmptyEnvironment, 201
 - defun, 201
- genDomainViewList0, 200
 - calledby makeFunctorArgumentParameters, 198
 - calls getDomainViewList, 200
 - defun, 200
- genSomeVariable
 - calledby compColon, 275
 - calledby setqMultiple, 331
- gensymp
 - calledby EqualBarGensym, 228
- genvar
 - calledby hasAplExtension, 397
- genVariable

- calledby `setqMultipleExplicit`, 333
- calledby `setqMultiple`, 331
- get
 - calledby `applyMapping`, 562
 - calledby `autoCoerceByModemap`, 354
 - calledby `coerceHard`, 348
 - calledby `coerceSubset`, 347
 - calledby `compAtom`, 565
 - calledby `compDefineCapsuleFunction`, 288
 - calledby `compFocompFormWithModemap`, 581
 - calledby `compMapCond`, 241
 - calledby `compNoStacking1`, 558
 - calledby `compSymbol`, 569
 - calledby `compTypeOf`, 564
 - calledby `compWithMappingModel1`, 586
 - calledby `compileCases`, 292
 - calledby `compile`, 145
 - calledby `doIt`, 261
 - calledby `evalAndSub`, 249
 - calledby `getArgumentMode`, 299
 - calledby `getDomainsInScope`, 234
 - calledby `getFormModemaps`, 576
 - calledby `getInverseEnvironment`, 310
 - calledby `getModemapListFromDomain`, 244
 - calledby `getModemapList`, 244
 - calledby `getModemap`, 239
 - calledby `getSignature`, 296
 - calledby `getSuccessEnvironment`, 309
 - calledby `giveFormalParametersValues`, 135
 - calledby `hasFullSignature`, 134
 - calledby `hasType`, 350
 - calledby `isFunctor`, 233
 - calledby `isMacro`, 267
 - calledby `isSuperDomain`, 235
 - calledby `isUnionMode`, 311
 - calledby `maxSuperType`, 338
 - calledby `mkEvalableCategoryForm`, 141
 - calledby `optCallSpecially`, 215
 - calledby `outputComp`, 337
 - calledby `parseHasRhs`, 110
 - calledby `setqSingle`, 334
- get-a-line, 608
 - calledby `initialize-prepare`, 73
 - calledby `prepareReadLine1`, 87
 - calls `is-console`, 608
 - calls `mkprompt[5]`, 608
 - calls `read-a-line`, 608
 - defun, 608
- get-internal-run-time
 - calledby `s-process`, 551
- get-token, 457
 - calledby `try-get-token`, 455
 - calls `XTokenReader`, 457
 - uses `XTokenReader`, 457
 - defun, 457
- getAbbreviation, 285
 - calledby `applyMapping`, 562
 - calledby `compApplication`, 574
 - calledby `compDefine1`, 283
 - calledby `encodeFunctionName`, 148
 - calls `assq`, 285
 - calls `constructor?`, 285
 - calls `mkAbbrev`, 285
 - calls `rplac`, 285
 - local def `$abbreviationTable`, 285
 - local ref `$abbreviationTable`, 285
 - defun, 285
- getArgumentMode, 299
 - calledby `checkAndDeclare`, 298
 - calledby `getArgumentModeOrMoan`, 156
 - calledby `hasSigInTargetCategory`, 298
 - calls `get`, 299
 - defun, 299
- getArgumentModeOrMoan, 156
 - calledby `compDefineCapsuleFunction`, 288
 - calledby `compDefineCategory2`, 142
 - calledby `compDefineFunctor1`, 183
 - calls `getArgumentMode`, 156
 - calls `stackSemanticError`, 156
 - defun, 156
- getCaps, 150
 - calledby `encodeItem`, 150
 - calls `l-case`, 150
 - calls `maxindex`, 150
 - calls `strconc`, 150
 - calls `stringimage`, 150
 - defun, 150
- getCategoryOpsAndAtts, 178
 - calledby `getConstructorOpsAndAtts`, 178
 - calls `getSlotFromCategoryForm`, 179
 - calls `transformOperationAlist`, 178

- defun, 178
- getConstructorAbbreviation
 - calledby compDefineLisplib, 171
- getConstructorOpsAndAtts, 178
 - calledby finalizeLisplib, 176
 - calls getCategoryOpsAndAtts, 178
 - calls getFunctorOpsAndAtts, 178
 - defun, 178
- getdatabase
 - calledby augModemapsFromDomain, 236
 - calledby checkDocMessage, 479
 - calledby checkNumOfArgs, 499
 - calledby compDefineFunctor1, 184
 - calledby compDefineLisplib, 171
 - calledby compileConstructor1, 153
 - calledby getOperationAlist, 250
 - calledby isFunctor, 233
 - calledby loadLibIfNecessary, 111
 - calledby macroExpandList, 137
 - calledby mkCategoryPackage, 139
 - calledby mkEvaluableCategoryForm, 140
 - calledby parseHas, 108
 - calledby updateCategoryFrameForCategory, 113
 - calledby updateCategoryFrameForConstructor, 112
 - calledby whoOwns, 480
- getDeltaEntry
 - calledby compElt, 300
- getDomainsInScope, 234
 - calledby addDomain, 232
 - calledby augModemapsFromDomain, 236
 - calledby comp3, 560
 - calledby compColon, 275
 - calledby compConstruct, 280
 - calledby putDomainsInScope, 235
 - calls get, 234
 - local ref \$CapsuleDomainsInScope, 234
 - local ref \$insideCapsuleFunctionIfTrue, 234
 - defun, 234
- getDomainViewList
 - calledby genDomainViewList0, 200
- getExportCategory
 - calledby NRTgetLookupFunction, 191
- getFormModemaps, 575
 - calledby compForm1, 572
 - calledby getFormModemaps, 576
 - calls eltModemapFilter, 576
 - calls getFormModemaps, 576
 - calls get, 576
 - calls last, 576
 - calls length, 576
 - calls nequal, 576
 - calls nreverse0, 576
 - calls qcar, 575
 - calls qcdr, 576
 - calls stackMessage, 576
 - local ref \$insideCategoryPackageIfTrue, 576
 - defun, 575
- getFunctorOpsAndAtts, 181
 - calledby getConstructorOpsAndAtts, 178
 - calls getSlotFromFunctor, 181
 - calls transformOperationAlist, 181
 - defun, 181
- getIdentity
 - calledby compReduce1, 320
- getInverseEnvironment, 310
 - calledby compBoolean, 308
 - calls delete, 310
 - calls getUnionMode, 310
 - calls get, 310
 - calls identp, 310
 - calls isDomainForm, 310
 - calls member, 310
 - calls mkpf, 310
 - calls put, 310
 - calls qcar, 310
 - calls qcdr, 310
 - local ref \$EmptyEnvironment, 310
 - defun, 310
- getl
 - calledby PARSE-Operation, 421
 - calledby PARSE-ReductionOp, 425
 - calledby PARSE-leftBindingPowerOf, 421
 - calledby PARSE-rightBindingPowerOf, 422
 - calledby addConstructorModemaps, 238
 - calledby augModemapsFromDomain1, 237
 - calledby checkRecordHash, 473
 - calledby checkTransformFirsts, 488
 - calledby compCat, 270

- calledby compExpression, 571
- calledby loadLibIfNecessary, 111
- calledby mustInstantiate, 274
- calledby optSpecialCall, 216
- calledby optimize, 209
- calledby parseTran, 93
- getMatchingRightPren, 492
 - calledby checkGetArgs, 504
 - calledby checkTransformFirsts, 488
 - calls maxindex, 492
 - defun, 492
- getmode
 - calledby addDomain, 232
 - calledby augModemapsFromDomain1, 237
 - calledby coerceHard, 348
 - calledby comp3, 560
 - calledby compCoerce, 352
 - calledby compColon, 275
 - calledby compDefWhereClause, 204
 - calledby compDefineCapsuleFunction, 288
 - calledby compSymbol, 569
 - calledby compile, 145
 - calledby displayMissingFunctions, 197
 - calledby freelist, 594
 - calledby getSignatureFromMode, 287
 - calledby getSignature, 296
 - calledby getUnionMode, 311
 - calledby isUnionMode, 311
 - calledby setqSingle, 334
- getModemap, 239
 - calledby compDefineFunctor1, 183
 - calls compApplyModemap, 239
 - calls get, 239
 - calls sublis, 239
 - defun, 239
- getModemapList, 244
 - calledby autoCoerceByModemap, 354
 - calledby compCase1, 268
 - calledby getUniqueModemap, 243
 - calls getModemapListFromDomain, 244
 - calls get, 244
 - calls nreverse0, 244
 - calls qcar, 244
 - calls qcdr, 244
 - defun, 244
- getModemapListFromDomain, 244
 - calledby compElt, 300
 - calledby getModemapList, 244
 - calls get, 244
 - defun, 244
- getmodeOrMapping
 - calledby augModemapsFromDomain1, 237
- getOperationAlist, 250
 - calledby evalAndSub, 249
 - calledby genDomainOps, 202
 - calls compMakeCategoryObject, 250
 - calls getdatabase, 250
 - calls isFunctor, 250
 - calls stackMessage, 250
 - calls systemError, 250
 - uses \$domainShell, 250
 - uses \$e, 250
 - uses \$functorForm, 250
 - uses \$insideFunctorIfTrue, 250
 - defun, 250
- getOperationAlistFromLisplib
 - calledby mkOpVec, 203
- getParentsFor
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
- getPrincipalView
 - calledby mkOpVec, 203
- getProplist
 - calledby addModemap1, 254
 - calledby compDefineAddSignature, 133
 - calledby getSuccessEnvironment, 308
 - calledby loadLibIfNecessary, 111
- getProplist[5]
 - called by setqSingle, 334
- getScriptName, 399
 - calledby postScriptsForm, 362
 - calledby postScripts, 389
 - calls decodeScripts, 399
 - calls identp[5], 399
 - calls intern1, 399
 - calls pname[5], 399
 - calls postError, 399
 - defun, 399
- getSignature, 296
 - calledby compDefineCapsuleFunction, 288
 - calls SourceLevelSubsume, 296
 - calls getmode, 296

- calls get, 296
- calls knownInfo, 296
- calls length, 296
- calls printSignature, 296
- calls qcar, 296
- calls qcdr, 296
- calls remdup, 296
- calls say, 296
- calls stackSemanticError, 296
- local ref \$e, 297
- defun, 296
- getSignatureFromMode, 287
 - calledby compDefine1, 283
 - calledby hasSigInTargetCategory, 298
 - calls eqsubstlist, 287
 - calls getmode, 287
 - calls length, 287
 - calls nequal, 287
 - calls opOf, 287
 - calls qcar, 287
 - calls qcdr, 287
 - calls stackAndThrow, 287
 - calls take, 287
 - local ref \$FormalMapVariableList, 287
 - defun, 287
- getSlotFromCategoryForm, 179
 - calledby getCategoryOpsAndAtts, 179
 - calls eval, 179
 - calls systemErrorHere, 179
 - calls take, 179
 - local ref \$FormalMapVariableList, 179
 - defun, 179
- getSlotFromFunctor, 181
 - calledby getFunctorOpsAndAtts, 181
 - calls compMakeCategoryObject, 181
 - calls systemErrorHere, 181
 - local ref \$e, 181
 - local ref \$lisplibOperationAlist, 181
 - defun, 181
- getSpecialCaseAssoc, 293
 - calledby compileCases, 292
 - local ref \$functorForm, 293
 - local ref \$functorSpecialCases, 293
 - defun, 293
- getSuccessEnvironment, 308
 - calledby compBoolean, 308
- calledby doItIf, 265
- calls addBinding, 309
- calls comp, 308
- calls consPropListOf, 309
- calls getPropList, 308
- calls get, 309
- calls identp, 308
- calls isDomainForm, 308
- calls put, 308
- calls qcar, 308
- calls qcdr, 308
- calls removeEnv, 309
- local ref \$EmptyEnvironment, 309
- local ref \$EmptyMode, 309
- defun, 308
- getTargetFromRhs, 135
 - calledby compDefine1, 283
 - calledby getTargetFromRhs, 135
 - calls compOrCroak, 135
 - calls getTargetFromRhs, 135
 - calls stackSemanticError, 135
 - defun, 135
- getToken, 452
 - calledby PARSE-OpenBrace, 440
 - calledby PARSE-OpenBracket, 440
 - calls eqcar, 452
 - defun, 452
- getUnionMode, 311
 - calledby getInverseEnvironment, 310
 - calls getmode, 311
 - calls isUnionMode, 311
 - defun, 311
- getUniqueModemap, 243
 - calledby getUniqueSignature, 243
 - calls getModemapList, 243
 - calls qslessp, 243
 - calls stackWarning, 243
 - defun, 243
- getUniqueSignature, 243
 - calls getUniqueModemap, 243
 - defun, 243
- giveFormalParametersValues, 135
 - calledby compDefine1, 283
 - calledby compDefineCapsuleFunction, 288
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 183

- calls get, 135
- calls put, 135
- defun, 135
- hackforis, 402
 - calls hackforis1, 402
 - defun, 402
- hackforis1, 403
 - calledby hackforis, 402
 - calls eqcar, 403
 - calls kar, 403
 - defun, 403
- has, 108, 302
 - defplist, 108, 302
- hasAplExtension, 397
 - calledby aplTran1, 395
 - calls aplTran1, 397
 - calls deepestExpression, 397
 - calls genvar, 397
 - calls msubst, 397
 - calls nreverse0, 397
 - defun, 397
- hasFormalMapVariable, 592
 - calledby compWithMappingMode1, 586
 - calls ScanOrPairVec[5], 592
 - local def \$formalMapVariables, 592
 - defun, 592
- hasFullSignature, 134
 - calledby compDefineAddSignature, 133
 - calls get, 134
 - defun, 134
- hasNoVowels, 511
 - calledby checkDecorate, 508
 - calls maxindex, 511
 - defun, 511
- hasSigInTargetCategory, 298
 - calledby compDefineCapsuleFunction, 288
 - calls bright, 298
 - calls compareMode2Arg, 298
 - calls getArgumentMode, 298
 - calls getSignatureFromMode, 298
 - calls length, 298
 - calls remdup, 298
 - calls stackWarning, 298
 - local ref \$domainShell, 298
 - defun, 298
- hasType, 350
 - calledby coerceExtraHard, 349
 - calls get, 350
 - defun, 350
- helpSpad2Cmd[5]
 - called by compiler, 533
- hget
 - calledby checkArguments, 486
 - calledby checkBeginEnd, 484
 - calledby checkRecordHash, 473
 - calledby checkSplitPunctuation, 497
- hput
 - calledby checkRecordHash, 473
- htcharPosition, 501
 - calledby checkTrimCommented, 500
 - calledby htcharPosition, 501
 - calls charPosition, 501
 - calls htcharPosition, 501
 - calls length, 501
 - calls nequal, 501
 - local ref \$charBack, 501
 - defun, 501
- identp
 - calledby addDomain, 232
 - calledby compFocompFormWithModemap, 581
 - calledby compInternalFunction, 287
 - calledby constructMacro, 151
 - calledby encodeItem, 150
 - calledby getInverseEnvironment, 310
 - calledby getSuccessEnvironment, 308
 - calledby isFunctor, 233
 - calledby mkExplicitCategoryFunction, 273
 - calledby subrname, 212
- identp[5]
 - called by PARSE-FloatExponent, 431
 - called by compSetq1, 329
 - called by floatexpid, 459
 - called by freelist, 594
 - called by getScriptName, 399
 - called by postTransform, 359
 - called by setqSingle, 334
- if, 113, 304, 381
 - defplist, 113, 304, 381
- ifcar

- calledby checkBeginEnd, 484
- calledby checkFixCommonProblem, 508
- calledby checkTexht, 476
- calledby preparse, 76
- ifcdr
 - calledby checkBeginEnd, 484
 - calledby checkFixCommonProblem, 508
 - calledby checkHTargs, 487
 - calledby checkRecordHash, 473
 - calledby recordAttributeDocumentation, 463
- implies, 116
 - defplist, 116
- import, 312
 - defplist, 312
- In, 383
 - defplist, 383
- in, 117, 382
 - defplist, 117, 382
- inby, 118
 - defplist, 118
- incExitLevel
 - calledby parseIf,ifTran, 114
- indent-pos, 526
 - calledby preparse1, 81
 - defun, 526
- infixtok, 527
 - calledby add-parens-and-semis-to-line, 84
 - calls string2id-n, 527
 - defun, 527
- init-boot/spad-reader[5]
 - called by spad, 549
- initial-gensym, 402
 - defvar, 402
- initial-substring, 607
 - calledby preparseReadLine, 85
 - calledby skip-ifblock, 86
 - calledby skip-to-endif, 528
 - calls mismatch, 607
 - defun, 607
- initial-substring-p, 450
 - calledby match-string, 448
 - calls string-not-greaterp, 450
 - defun, 450
- initialize-preparse, 73
 - calledby spad, 549
- calls get-a-line, 73
 - uses \$echolinestack, 73
 - uses \$index, 73
 - uses \$linelist, 73
 - uses \$preparse-last-line, 73
 - defun, 73
- initializeLisplib, 175
 - calls LAM,FILEACTQ, 175
 - calls addoptions, 175
 - calls erase, 175
 - calls pathnameTypeId, 175
 - calls writeLib1, 175
 - local def \$libFile, 175
 - local def \$lisplibAbbreviation, 175
 - local def \$lisplibAncestors, 175
 - local def \$lisplibForm, 175
 - local def \$lisplibKind, 175
 - local def \$lisplibModemapAlist, 175
 - local def \$lisplibModemap, 175
 - local def \$lisplibOpAlist, 175
 - local def \$lisplibOperationAlist, 175
 - local def \$lisplibSignatureAlist, 175
 - local def \$lisplibSuperDomain, 175
 - local def \$lisplibVariableAlist, 175
 - local ref \$erase, 175
 - local ref \$libFile, 175
 - uses /editfile, 175
 - uses /major-version, 175
 - uses errors, 175
 - defun, 175
- insert
 - calledby comp2, 559
 - calledby doIt, 261
- insertAlist
 - calledby transformOperationAlist, 180
- insertModemap, 247
 - calledby mkNewModemapList, 246
 - defun, 247
- insertWOC
 - calledby orderPredTran, 163
- Integer
 - calledby optCallEval, 218
- interactiveModemapForm, 160
 - calledby augLisplibModemapsFromCategory, 157

- calledby augmentLisplibModemapsFrom-
 Functor, 193
- calls fixUpPredicate, 160
- calls modemapPattern, 160
- calls nequal, 160
- calls qcar, 160
- calls qcdr, 160
- calls replaceVars, 160
- calls substVars, 160
- local ref \$FormalMapVariableList, 160
- local ref \$PatternVariableList, 160
- defun, 160
- intern
 - calledby checkRecordHash, 473
- internl
 - calledby encodeFunctionName, 148
 - calledby getScriptName, 399
 - calledby postForm, 364
 - calledby substituteCategoryArguments, 237
- intersection
 - calledby orderByDependency, 207
- intersectionEnvironment
 - calledby compIf, 305
 - calledby replaceExitEtc, 327
- intersectionq
 - calledby orderPredTran, 163
- ioclear
 - calledby spad, 549
- is, 119, 312
 - defplist, 119, 312
- is-console, 527
 - calledby get-a-line, 608
 - calledby preparse1, 81
 - calledby print-defun, 553
 - calls fp-output-stream, 527
 - uses *terminal-io*, 527
 - defun, 527
- isAlmostSimple
 - calledby makeSimplePredicateOrNil, 518
- isCategory
 - calledby augModemapsFromCategoryRep, 252
 - calledby evalAndSub, 249
- isCategoryForm
 - calledby addDomain, 232
 - calledby applyMapping, 562
 - calledby augLisplibModemapsFromCat-
 egory, 157
 - calledby coerceHard, 348
 - calledby compApplication, 574
 - calledby compColon, 275
 - calledby compFocompFormWithModemap, 581
 - calledby compJoin, 313
 - calledby compMakeCategoryObject, 182
 - calledby genDomainViewList, 201
 - calledby isDomainConstructorForm, 339
 - calledby isDomainForm, 338
 - calledby makeCategoryForm, 278
 - calledby makeFunctorArgumentParamet-
 ers, 198
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calledby mkDatabasePred, 195
 - calledby signatureTran, 163
- isCategoryPackageName, 190
 - calledby compDefineFunctor1, 183, 184
 - calledby substNames, 251
 - calls char, 190
 - calls maxindex, 190
 - calls pname, 190
 - defun, 190
- isConstantId
 - calledby eltModemapFilter, 577
 - calledby seteltModemapFilter, 577
- isDomainConstructorForm, 339
 - calledby isDomainForm, 338
 - calls eqsubstlist, 339
 - calls isCategoryForm, 339
 - calls qcar, 339
 - calls qcdr, 339
 - local ref \$FormalMapVariableList, 339
 - defun, 339
- isDomainForm, 338
 - calledby comp2, 559
 - calledby compColon, 275
 - calledby compDefine1, 283
 - calledby compElt, 300
 - calledby compHasFormat, 303
 - calledby doIt, 261
 - calledby getInverseEnvironment, 310
 - calledby getSuccessEnvironment, 308

- calledby setqSingle, 334
- calls isCategoryForm, 338
- calls isDomainConstructorForm, 338
- calls isFunctor, 338
- calls kar, 338
- calls qcar, 338
- calls qcdr, 338
- local ref \$SpecialDomainNames, 338
- defun, 338
- isDomainInScope
 - calledby setqSingle, 334
- isDomainSubst, 166
 - calledby orderPredTran, 163
 - defun, 166
- isFunction
 - calledby compSymbol, 569
- isFunctor, 233
 - calledby addDomain, 232
 - calledby comp2, 559
 - calledby compFocompFormWithModemap, 581
 - calledby compWithMappingModel, 586
 - calledby getOperationAlist, 250
 - calledby isDomainForm, 338
 - calls constructor?, 233
 - calls getdatabase, 233
 - calls get, 233
 - calls identp, 233
 - calls opOf, 233
 - calls updateCategoryFrameForCategory, 233
 - calls updateCategoryFrameForConstructor, 233
 - local ref \$CategoryFrame, 233
 - local ref \$InteractiveMode, 234
 - defun, 233
- isListConstructor, 103
 - calledby transIs, 102
 - calls member, 103
 - defun, 103
- isLiteral
 - calledby addDomain, 232
- isMacro, 267
 - calledby compDefine1, 283
 - calledby doIt, 261
 - calls get, 267
 - calls qcar, 267
 - calls qcdr, 267
 - defun, 267
- isnt, 119
 - defplist, 119
- isSimple
 - calledby compForm2, 579
 - calledby makeSimplePredicateOrNil, 518
- isSomeDomainVariable
 - calledby coerce, 346
- isSubset
 - calledby coerceByModemap, 354
 - calledby coerceSubset, 347
 - calledby isSuperDomain, 235
- isSuperDomain, 235
 - calledby mergeModemap, 248
 - calls get, 235
 - calls isSubset, 235
 - calls lassoc, 235
 - calls opOf, 235
 - defun, 235
- isSymbol
 - calledby compAtom, 565
- isTokenDelimiter, 451
 - calledby PARSE-TokenList, 414
 - calls current-symbol, 451
 - defun, 451
- isUnionMode, 311
 - calledby coerceExtraHard, 349
 - calledby getUnionMode, 311
 - calls getmode, 311
 - calls get, 311
 - defun, 311
- Join, 120, 313, 383
 - defplist, 120, 313, 383
- JoinInner
 - calledby mkCategoryPackage, 139
- kar
 - calledby addEmptyCapsuleIfNecessary, 134
 - calledby augModemapsFromDomain1, 237
 - calledby augModemapsFromDomain, 236
 - calledby compile, 145
 - calledby doIt, 261
 - calledby hackforis1, 403

- calledby isDomainForm, 338
- calledby optCallSpecially, 215
- keyedSystemError
 - calledby NRTputInHead, 155
 - calledby coerce, 346
 - calledby compileTimeBindingOf, 217
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calledby optRECORDELT, 230
 - calledby optSETRECORDELT, 231
 - calledby optSpecialCall, 216
 - calledby substituteIntoFunctorModemap, 583
 - calledby transformOperationAlist, 180
- killColons, 391
 - calledby killColons, 391
 - calledby postSignature, 390
 - calls killColons, 391
 - defun, 391
- knownInfo
 - calledby addModemap, 253
 - calledby compMapCond", 241
 - calledby getSignature, 296
- l-case
 - calledby getCaps, 150
- labasoc
 - usedby PARSE-Data, 436
- lablasoc, 411
 - defvar, 411
- LAM,EVALANDFILEACTQ
 - calledby spadCompileOrSetq, 152
- LAM,FILEACTQ
 - calledby initializeLisplib, 175
- lassoc
 - calledby NRTputInTail, 154
 - calledby addModemap1, 254
 - calledby augLisplibModemapsFromCategory, 157
 - calledby checkAddMacros, 501
 - calledby checkTransformFirsts, 488
 - calledby coerceSubset, 347
 - calledby compDefWhereClause, 204
 - calledby compDefineAddSignature, 133
 - calledby compOrCroak1,compactify, 595
 - calledby isSuperDomain, 235
 - calledby loadLibIfNecessary, 111
 - calledby mergeSignatureAndLocalVarAlists, 182
 - calledby optCallSpecially, 215
 - calledby translable1, 515
- lassq
 - calledby transformOperationAlist, 180
- last
 - calledby compFocompFormWithModemap, 581
 - calledby getFormModemaps, 576
 - calledby parseSeq, 128
 - calledby setqMultipleExplicit, 333
- lastnode
 - calledby NRTputInHead, 155
 - calledby doIt, 261
- leave, 121, 316
 - defplist, 121, 316
- leftTrim
 - calledby checkTransformFirsts, 488
- length
 - calledby checkBeginEnd, 484
 - calledby checkExtract, 507
 - calledby checkSplitBrace, 495
 - calledby checkTrimCommented, 500
 - calledby compApplication, 574
 - calledby compApplyModemap, 239
 - calledby compColon, 275
 - calledby compDefine1, 283
 - calledby compDefineCapsuleFunction, 288
 - calledby compElt, 300
 - calledby compForm1, 571
 - calledby compForm2, 579
 - calledby compHasFormat, 303
 - calledby compRepeatOrCollect, 323
 - calledby displayPreCompilationErrors, 516
 - calledby encodeFunctionName, 148
 - calledby fixUpPredicate, 161
 - calledby getFormModemaps, 576
 - calledby getSignatureFromMode, 287
 - calledby getSignature, 296
 - calledby hasSigInTargetCategory, 298
 - calledby htcharPosition, 501
 - calledby mkOpVec, 203
 - calledby modeEqualSubst, 357
 - calledby optMkRecord, 229

- calledby postScriptsForm, 362
 - calledby removeBackslashes, 476
 - calledby setqMultiple, 331
- lessp, 222
 - defplist, 222
- let, 122, 329
 - defplist, 122, 329
- letd, 123
 - defplist, 123
- line, 603
 - usedby match-string, 448
 - usedby spad, 549
 - defstruct, 603
- Line-Advance-Char
 - calledby Advance-Char, 607
- line-advance-char, 605
 - uses \$line, 605
 - defun, 605
- Line-At-End-P
 - calledby Advance-Char, 607
- line-at-end-p, 604
 - calledby next-char, 458
 - uses \$line, 604
 - defun, 604
- line-clear, 604
 - uses \$line, 604
 - defmacro, 604
- line-current-char
 - calledby match-advance-string, 449
- line-current-index
 - calledby match-advance-string, 449
- line-current-segment, 606
 - calledby unget-tokens, 452
 - defun, 606
- Line-New-Line
 - calledby read-a-line, 601
- line-new-line, 606
 - calledby unget-tokens, 453
 - uses \$line, 606
 - defun, 606
- line-next-char, 605
 - calledby next-char, 458
 - uses \$line, 605
 - defun, 605
- line-number
 - calledby PARSE-Category, 418
- calledby unget-tokens, 453
- line-past-end-p, 605
 - calledby match-advance-string, 449
 - calledby match-string, 448
 - uses \$line, 605
 - defun, 605
- line-print, 604
 - local ref \$out-stream, 604
 - uses \$line, 604, 606
 - defun, 604
- lispize, 342
 - calledby compSubDomain1, 341
 - calls optimize, 342
 - defun, 342
- lisplibDoRename, 174
 - calledby compDefineLisplib, 171
 - calls replaceFile, 174
 - local ref \$spadLibFT, 174
 - defun, 174
- lisplibWrite, 182
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby compileDocumentation, 174
 - calledby finalizeLisplib, 176
 - calls rwrite128, 182
 - local ref \$lisplib, 182
 - defun, 182
- List
 - calledby optCallEval, 218
- ListCategory, 281
 - defplist, 281
- listOfIdentifiersIn
 - calledby compDefWhereClause, 204
- listOfPatternIds
 - calledby augmentLisplibModemapsFrom-Functor, 193
 - calledby orderPredTran, 163
- listOfSharpVars
 - calledby compFocompFormWithModemap, 581
- listOrVectorElementNode
 - calledby augModemapsFromDomain, 236
- loadIfNecessary, 111
 - calledby parseHasRhs, 110
 - calls loadLibIfNecessary, 111
 - defun, 111

- loadLib
 - calledby loadLibIfNecessary, 111
- loadLibIfNecessary, 111
 - calledby loadIfNecessary, 111
 - calledby loadLibIfNecessary, 111
 - calls functionp, 111
 - calls getPropList, 111
 - calls getdatabase, 111
 - calls getl, 111
 - calls lassoc, 111
 - calls loadLibIfNecessary, 111
 - calls loadLib, 111
 - calls macrop, 111
 - calls throwKeyedMsg, 111
 - calls updateCategoryFrameForCategory, 111
 - calls updateCategoryFrameForConstructor, 111
 - local ref \$CategoryFrame, 111
 - local ref \$InteractiveMode, 111
 - defun, 111
- localdatabase
 - calledby compDefineLisplib, 171
- localdatabase[5]
 - called by compileSpadLispCmd, 596
- localExtras
 - calledby doItIf, 265
- lt
 - calledby PARSE-Operation, 421
- macroExpand, 136
 - calledby compDefine1, 283
 - calledby compMacro, 317
 - calledby compSeqItem, 328
 - calledby compWhere, 345
 - calledby finalizeDocumentation, 466
 - calledby macroExpandInPlace, 136
 - calledby macroExpandList, 137
 - calledby macroExpand, 136
 - calls macroExpandList, 136
 - calls macroExpand, 136
 - defun, 136
- macroExpandInPlace, 136
 - calledby compSingleCapsuleItem, 261
 - calls macroExpand, 136
 - defun, 136
- macroExpandList, 137
 - calledby macroExpand, 136
 - calls getdatabase, 137
 - calls macroExpand, 137
 - defun, 137
- macrop
 - calledby loadLibIfNecessary, 111
- make-float
 - calledby PARSE-Float, 429
- make-full-cvec
 - calledby prepare1, 81
- make-input-filename
 - calledby compileDocumentation, 174
- make-reduction
 - calledby push-reduction, 462
- make-symbol-of, 454
 - calledby PARSE-Expression, 419
 - calledby current-symbol, 454
 - uses \$token, 454
 - defun, 454
- makeCategoryForm, 278
 - calledby compColon, 275
 - calls compOrCroak, 278
 - calls isCategoryForm, 278
 - local ref \$EmptyMode, 278
 - defun, 278
- makeCategoryPredicates, 138
 - calledby compDefineCategory1, 137
 - uses \$FormalMapVariableList, 138
 - uses \$TriangleVariableList, 138
 - uses \$mvl, 138
 - uses \$tvl, 138
 - defun, 138
- makeFunctorArgumentParameters, 198
 - calledby compDefineFunctor1, 184
 - calls assq, 198
 - calls genDomainViewList0, 198
 - calls isCategoryForm, 198
 - calls msubst, 198
 - calls qcar, 198
 - calls qcdr, 198
 - calls union, 198
 - uses \$ConditionalOperators, 199
 - uses \$alternateViewList, 199
 - uses \$forceAdd, 199
 - defun, 198

- makeInitialModemapFrame[5]
 - called by spad, 549
- makeInputFilename[5]
 - called by /rf-1, 540
- makeLiteral
 - calledby addEltModemap, 245
- makeNonAtomic
 - calledby parseHas, 109
- makeSimplePredicateOrNil, 518
 - calledby parseIf,ifTran, 114
 - calls isAlmostSimple, 518
 - calls isSimple, 518
 - calls wrapSEQExit, 518
 - defun, 518
- mapInto
 - calledby parseSeq, 128
 - calledby parseWhere, 129
- Mapping, 269
 - defplist, 269
- match-advance-string, 449
 - calledby PARSE-Category, 417
 - calledby PARSE-Command, 412
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Enclosure, 432
 - calledby PARSE-Exit, 443
 - calledby PARSE-FloatBasePart, 430
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-Form, 425
 - calledby PARSE-Import, 419
 - calledby PARSE-IteratorTail, 441
 - calledby PARSE-Iterator, 441
 - calledby PARSE-Label, 427
 - calledby PARSE-Leave, 444
 - calledby PARSE-Loop, 446
 - calledby PARSE-Option, 416
 - calledby PARSE-Primary1, 429
 - calledby PARSE-PrimaryOrQM, 415
 - calledby PARSE-Quad, 433
 - calledby PARSE-Qualification, 424
 - calledby PARSE-Return, 443
 - calledby PARSE-ScriptItem, 435
 - calledby PARSE-Scripts, 434
 - calledby PARSE-Selector, 427
 - calledby PARSE-SemiColon, 443
 - calledby PARSE-Sequence, 439
 - calledby PARSE-Sexpr1, 436
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-Statement, 416
 - calledby PARSE-TokenOption, 414
 - calledby PARSE-With, 417
 - calls current-token, 449
 - calls line-current-char, 449
 - calls line-current-index, 449
 - calls line-past-end-p, 449
 - calls match-string, 449
 - calls quote-if-string, 449
 - uses \$line, 449
 - uses \$token, 449
 - defun, 449
- match-current-token, 453
 - calledby PARSE-GlyphTok, 438
 - calledby PARSE-NBGlyphTok, 437
 - calledby PARSE-Operation, 421
 - calledby PARSE-SpecialKeyWord, 412
 - calledby parse-argument-designator, 520
 - calledby parse-identifier, 519
 - calledby parse-keyword, 520
 - calledby parse-number, 520
 - calledby parse-spadstring, 518
 - calledby parse-string, 519
 - calls current-token, 453
 - calls match-token, 453
 - defun, 453
- match-next-token, 454
 - calledby PARSE-ReductionOp, 425
 - calls match-token, 454
 - calls next-token, 454
 - defun, 454
- match-string, 448
 - calledby PARSE-AnyId, 438
 - calledby PARSE-NewExpr, 411
 - calledby PARSE-Primary1, 429
 - calledby match-advance-string, 449
 - calls current-char, 448
 - calls initial-substring-p, 448
 - calls line-past-end-p, 448
 - calls skip-blanks, 448
 - calls subseq, 448
 - calls unget-tokens, 448
 - uses \$line, 448
 - uses line, 448
 - defun, 448

- match-token, 454
 - calledby match-current-token, 453
 - calledby match-next-token, 454
 - calls token-symbol, 454
 - calls token-type, 454
 - defun, 454
- Matrix
 - calledby optCallEval, 218
- maxindex
 - calledby checkAddBackSlashes, 511
 - calledby checkAddPeriod, 483
 - calledby checkAddSpaceSegments, 505
 - calledby checkGetArgs, 504
 - calledby checkIeEgfun, 494
 - calledby checkSplitBackslash, 496
 - calledby checkSplitOn, 498
 - calledby checkSplitPunctuation, 497
 - calledby checkTransformFirsts, 488
 - calledby compDefineFunctor1, 184
 - calledby firstNonBlankPosition, 493
 - calledby floatexpid, 459
 - calledby getCaps, 150
 - calledby getMatchingRightPren, 492
 - calledby hasNoVowels, 511
 - calledby isCategoryPackageName, 190
 - calledby prepare1, 81
 - calledby prepareReadLine1, 87
 - calledby translabel1, 515
- maxSuperType, 338
 - calledby coerceSubset, 347
 - calledby maxSuperType, 338
 - calledby setqSingle, 334
 - calls get, 338
 - calls maxSuperType, 338
 - defun, 338
- mbpip
 - calledby subname, 212
- mdef, 123, 317
 - defplist, 123, 317
- member
 - calledby addDomain, 232
 - calledby applyMapping, 562
 - calledby augLisplibModemapsFromCategory, 157
 - calledby augModemapsFromDomain, 236
 - calledby augmentLisplibModemapsFromFunctor, 193
 - calledby autoCoerceByModemap, 354
 - calledby checkBeginEnd, 484
 - calledby checkDecorateForHt, 477
 - calledby checkDecorate, 508
 - calledby checkFixCommonProblem, 508
 - calledby checkRecordHash, 472
 - calledby checkSkipOpToken, 492
 - calledby coerceExtraHard, 349
 - calledby compApplication, 574
 - calledby compApplyModemap, 239
 - calledby compDefineCapsuleFunction, 288
 - calledby compile, 145
 - calledby displayMissingFunctions, 197
 - calledby doIt, 261
 - calledby genDomainView, 201
 - calledby getInverseEnvironment, 310
 - calledby isListConstructor, 103
 - calledby mkNewModemapList, 246
 - calledby orderByDependency, 207
 - calledby orderPredTran, 163
 - calledby parseHasRhs, 110
 - calledby parseHas, 109
 - calledby putDomainsInScope, 235
 - calledby transformOperationAlist, 180
- member[5]
 - called by comp3, 560
 - called by compColon, 275
 - called by compSymbol, 569
 - called by compilerDoit, 538
- mergeModemap, 248
 - calledby mkNewModemapList, 246
 - calls TruthP, 248
 - calls isSuperDomain, 248
 - local ref \$forceAdd, 248
 - defun, 248
- mergePathnames[5]
 - called by compiler, 533
- mergeSignatureAndLocalVarAlists, 182
 - calledby finalizeLisplib, 176
 - calls lassoc, 182
 - defun, 182
- meta-error-handler, 458
 - calledby meta-syntax-error, 459
 - usedby meta-syntax-error, 459

- defvar, 458
- meta-syntax-error, 459
 - calledby must, 460
 - calls meta-error-handler, 459
 - uses meta-error-handler, 459
- defun, 459
- minus, 220
 - defplist, 220
- mismatch
 - calledby initial-substring, 607
- mkAbbrev, 286
 - calledby getAbbreviation, 285
 - calls addSuffix, 286
 - calls alistSize, 286
 - defun, 286
- mkAlistOfExplicitCategoryOps, 158
 - calledby augLisplibModemapsFromCategory, 157
 - calledby augmentLisplibModemapsFromFunctor, 193
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calls assocleft, 158
 - calls flattenSignatureList, 158
 - calls isCategoryForm, 158
 - calls keyedSystemError, 158
 - calls mkAlistOfExplicitCategoryOps, 158
 - calls nreverse0, 158
 - calls qcar, 158
 - calls qcdr, 158
 - calls remdup, 158
 - calls union, 158
 - local ref \$e, 158
 - defun, 158
- mkAutoLoad
 - calledby unloadOneConstructor, 173
- mkCategoryPackage, 139
 - calledby compDefineCategory1, 137
 - calls JoinInner, 139
 - calls abbreviationsSpad2Cmd, 139
 - calls assoc, 139
 - calls getdatabase, 139
 - calls msubst, 139
 - calls pname, 139
 - calls strconc, 139
 - calls sublislis, 139
 - uses \$FormalMapVariableList, 139
 - uses \$categoryPredicateList, 139
 - uses \$e, 139
 - uses \$options, 139
 - defun, 139
- mkConstructor, 170
 - calledby compDefineCategory2, 142
 - calledby mkConstructor, 170
 - calls mkConstructor, 170
 - defun, 170
- mkDatabasePred, 195
 - calledby augmentLisplibModemapsFromFunctor, 193
 - calls isCategoryForm, 195
 - local ref \$e, 195
 - defun, 195
- mkDomainConstructor
 - calledby bootStrapError, 196
 - calledby compHasFormat, 303
 - calledby genDomainOps, 202
 - calledby genDomainView, 201
- mkErrorExpr
 - calledby compOrCroak1, 557
- mkEvalableCategoryForm, 140
 - calledby compMakeCategoryObject, 182
 - calledby mkEvalableCategoryForm, 140
 - calls compOrCroak, 140
 - calls getdatabase, 140
 - calls get, 141
 - calls mkEvalableCategoryForm, 140
 - calls mkq, 141
 - calls qcar, 140
 - calls qcdr, 140
 - local def \$e, 141
 - local ref \$CategoryFrame, 141
 - local ref \$CategoryNames, 141
 - local ref \$Category, 141
 - local ref \$EmptyMode, 141
 - local ref \$e, 141
 - defun, 140
- mkExplicitCategoryFunction, 273
 - calledby compCategory, 270
 - calls identp, 273
 - calls mkq, 273
 - calls mustInstantiate, 273
 - calls nequal, 273

- calls remdup, 273
- calls union, 273
- calls wrapDomainSub, 273
- defun, 273
- mkList, 304
 - calledby compHasFormat, 303
 - defun, 304
- mkNewModemapList, 246
 - calledby addModemap1, 254
 - calls assoc, 246
 - calls insertModemap, 246
 - calls member, 246
 - calls mergeModemap, 246
 - calls nequal, 246
 - calls nreverse0, 246
 - calls qcar, 246
 - calls qcdr, 246
 - local ref \$InteractiveMode, 246
 - local ref \$forceAdd, 246
 - defun, 246
- mkOpVec, 203
 - calls AssocBarGensym, 203
 - calls assoc, 203
 - calls assq, 203
 - calls getOperationAlistFromLisplib, 203
 - calls getPrincipalView, 203
 - calls length, 203
 - calls msubst, 203
 - calls opOf, 203
 - calls qcar, 203
 - calls qcdr, 203
 - calls sublis, 203
 - uses Undef, 203
 - uses \$FormalMapVariableList, 203
 - defun, 203
- mkpf
 - calledby augLisplibModemapsFromCategory, 157
 - calledby augmentLisplibModemapsFromFunctor, 193
 - calledby compCapsuleInner, 259
 - calledby compCategoryItem, 271
 - calledby compileCases, 292
 - calledby getInverseEnvironment, 310
 - calledby stripOffSubdomainConditions, 295
- mkprogn
 - calledby setqMultiple, 331
- mkprompt[5]
 - called by get-a-line, 608
- mkq
 - calledby addArgumentConditions, 294
 - calledby bootStrapError, 196
 - calledby compCoerce1, 353
 - calledby compDefineCapsuleFunction, 288
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby compSeq1, 326
 - calledby genDomainOps, 202
 - calledby mkEvalableCategoryForm, 141
 - calledby mkExplicitCategoryFunction, 273
 - calledby optSpecialCall, 216
 - calledby spadCompileOrSetq, 152
- mkRecord, 229
 - defplst, 229
- mkRepititionAssoc, 149
 - calledby encodeFunctionName, 148
 - calls qcar, 149
 - calls qcdr, 149
 - defun, 149
- mkUnion, 356
 - calledby resolve, 356
 - calls qcar, 356
 - calls qcdr, 356
 - calls union, 356
 - local ref \$Rep, 356
 - defun, 356
- moan
 - calledby compileTimeBindingOf, 217
 - calledby parseExit, 107
 - calledby parseReturn, 127
- modeEqual, 357
 - calledby autoCoerceByModemap, 354
 - calledby checkAndDeclare, 298
 - calledby coerceByModemap, 354
 - calledby coerceHard, 348
 - calledby compAtomWithModemap, 567
 - calledby compCase1, 268
 - calledby compile, 145
 - calledby domainMember, 244
 - calledby modeEqualSubst, 357
 - calledby resolve, 356
 - defun, 357

- modeEqualSubst, 357
 - calledby coerceEasy, 346
 - calledby modeEqualSubst, 357
 - calls length, 357
 - calls modeEqualSubst, 357
 - calls modeEqual, 357
 - defun, 357
- modeIsAggregateOf
 - calledby compAtom, 565
 - calledby compConstruct, 280
 - calledby compRepeatOrCollect, 323
- modemapPattern, 169
 - calledby interactiveModemapForm, 160
 - calls qcar, 169
 - calls qcdr, 169
 - calls rassoc, 169
 - local ref \$PatternVariableList, 169
 - defun, 169
- modifyModeStack, 593
 - calledby compExit, 302
 - calledby compLeave, 317
 - calledby compReturn, 325
 - calls copy, 593
 - calls resolve, 593
 - calls say, 593
 - calls setelt, 593
 - uses \$exitModeStack, 594
 - uses \$reportExitModeStack, 593
 - defun, 593
- moveORsOutside, 167
 - calledby fixUpPredicate, 162
 - calledby moveORsOutside, 167
 - calls moveORsOutside, 167
 - defun, 167
- msubst
 - calledby addConstructorModemaps, 238
 - calledby addModemap1, 254
 - calledby augModemapsFromCategoryRep, 252
 - calledby augmentLisplibModemapsFrom-Functor, 193
 - calledby coerceSubset, 347
 - calledby coerce, 346
 - calledby compArgumentConditions, 294
 - calledby compCoerce1, 353
 - calledby compReduce1, 321
 - calledby compRepeatOrCollect, 323
 - calledby compSubsetCategory, 342
 - calledby compileCases, 292
 - calledby encodeFunctionName, 148
 - calledby finalizeDocumentation, 466
 - calledby hasAplExtension, 397
 - calledby makeFunctorArgumentParameters, 198
 - calledby mkCategoryPackage, 139
 - calledby mkOpVec, 203
 - calledby parseDollarGreaterEqual, 104
 - calledby parseDollarGreaterThan, 104
 - calledby parseDollarLessEqual, 105
 - calledby parseDollarNotEqual, 106
 - calledby parseNotEqual, 125
 - calledby parseTransform, 93
 - calledby parseType, 98
 - calledby replaceVars, 161
 - calledby stripOffArgumentConditions, 296
 - calledby substVars, 168
 - calledby substituteCategoryArguments, 237
- must, 460
 - calledby PARSE-Category, 417
 - calledby PARSE-Command, 412
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Enclosure, 432
 - calledby PARSE-Exit, 443
 - calledby PARSE-FloatBasePart, 431
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-Float, 429
 - calledby PARSE-Form, 425
 - calledby PARSE-Import, 419
 - calledby PARSE-Infix, 423
 - calledby PARSE-Iterator, 441
 - calledby PARSE-LabelExpr, 446
 - calledby PARSE-Label, 427
 - calledby PARSE-Leave, 444
 - calledby PARSE-Loop, 446
 - calledby PARSE-NewExpr, 411
 - calledby PARSE-Option, 416
 - calledby PARSE-Prefix, 423
 - calledby PARSE-Primary1, 428
 - calledby PARSE-Qualification, 424
 - calledby PARSE-Reduction, 425
 - calledby PARSE-Return, 443

- calledby PARSE-ScriptItem, 435
- calledby PARSE-Scripts, 434
- calledby PARSE-Selector, 427
- calledby PARSE-SemiColon, 443
- calledby PARSE-Sequence, 439
- calledby PARSE-Sexpr1, 436
- calledby PARSE-SpecialCommand, 413
- calledby PARSE-Statement, 416
- calledby PARSE-TokenOption, 414
- calledby PARSE-With, 417
- calls meta-syntax-error, 460
- defmacro, 460
- mustInstantiate, 274
 - calledby mkExplicitCategoryFunction, 273
 - calls getl, 274
 - calls qcar, 274
 - local ref \$DummyFunctorNames, 274
 - defun, 274
- namestring
 - calledby bootStrapError, 196
 - calledby finalizeLisplib, 176
- namestring[5]
 - called by compileSpad2Cmd, 536
 - called by compileSpadLispCmd, 596
 - called by compiler, 533
- ncINTERPFILE, 595
 - calledby /rf-1, 540
 - calls SpadInterpretStream[5], 595
 - uses \$EchoLines, 595
 - uses \$ReadingFile, 595
 - defun, 595
- ncParseFromString
 - calledby checkGetParse, 475
- nequal
 - calledby applyMapping, 562
 - calledby checkComments, 480
 - calledby checkFixCommonProblem, 508
 - calledby checkGetArgs, 504
 - calledby checkLookForLeftBrace, 487
 - calledby checkPrenAlist, 483
 - calledby checkTexht, 476
 - calledby checkTransformFirsts, 488
 - calledby checkTrimCommented, 500
 - calledby checkTrim, 506
 - calledby comp2, 559
 - calledby compApplication, 574
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby compElt, 300
 - calledby compPretend, 319
 - calledby compReturn, 325
 - calledby compileSpad2Cmd, 536
 - calledby compile, 145
 - calledby displayPreCompilationErrors, 516
 - calledby firstNonBlankPosition, 493
 - calledby getFormModemaps, 576
 - calledby getSignatureFromMode, 287
 - calledby htcharPosition, 501
 - calledby interactiveModemapForm, 160
 - calledby mkExplicitCategoryFunction, 273
 - calledby mkNewModemapList, 246
 - calledby postDef, 379
 - calledby postError, 364
 - calledby resolve, 356
 - calledby setqMultipleExplicit, 333
 - calledby setqSingle, 334
 - calledby substituteIntoFunctorModemap, 583
- new2OldLisp, 518
 - calledby s-process, 550
 - calls new2OldTran, 518
 - calls postTransform, 518
 - defun, 518
- new2OldTran
 - calledby new2OldLisp, 518
- newComp
 - calledby compTopLevel, 554
- newString2Words, 503
 - calledby checkComments, 481
 - calledby checkRewrite, 471
 - calls newWordFrom, 503
 - calls nreverse0, 503
 - defun, 503
- newWordFrom, 503
 - calledby newString2Words, 503
 - local ref \$charBlank, 503
 - local ref \$charFauxNewline, 503
 - local ref \$stringFauxNewline, 503
 - defun, 503
- next-char, 457
 - calledby PARSE-FloatBase, 430

- calls line-at-end-p, 458
- calls line-next-char, 458
- uses current-line, 458
- defun, 457
- next-line, 606
 - calledby Advance-Char, 607
 - local ref \$in-stream, 606
 - local ref \$line-handler, 606
 - defun, 606
- next-tab-loc, 527
 - defun, 527
- next-token, 91, 456
 - calledby match-next-token, 454
 - calls current-token, 456
 - calls try-get-token, 456
 - usedby next-token, 456
 - uses \$token, 91
 - uses next-token, 456
 - uses valid-tokens, 456
 - defun, 456
 - defvar, 91
- nonblank, 91
 - defvar, 91
- nonblankloc, 528
 - calledby add-parens-and-semis-to-line, 84
 - calls blankp, 528
 - defun, 528
- normalizeStatAndStringify
 - calledby reportOnFunctorCompilation, 197
- not, 124
 - defplist, 124
- notequal, 125
 - defplist, 125
- nreverse
 - calledby checkAddMacros, 501
 - calledby checkIeEg, 494
 - calledby checkPrenAlist, 483
 - calledby transDoc, 469
- nreverse0
 - calledby aplTran1, 395
 - calledby compAdd, 256
 - calledby compCase1, 268
 - calledby compColon, 275
 - calledby compExpressionList, 578
 - calledby compForm1, 572
 - calledby compForm2, 579
 - calledby compJoin, 313
 - calledby compReduce1, 320
 - calledby compSeq1, 326
 - calledby getFormModemaps, 576
 - calledby getModemapList, 244
 - calledby hasAplExtension, 397
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calledby mkNewModemapList, 246
 - calledby newString2Words, 503
 - calledby outputComp, 337
 - calledby parseHas, 109
 - calledby postCategory, 371
 - calledby postDef, 379
 - calledby postIf, 381
 - calledby postMDef, 385
 - calledby setqMultiple, 330
 - calledby substNames, 251
 - calledby transIs1, 102
- NRTaddInner
 - calledby NRTgetLocalIndex, 192
- NRTassignCapsuleFunctionSlot
 - calledby compDefineCapsuleFunction, 288
- NRTassocIndex, 336
 - calledby NRTgetLocalIndex, 192
 - calledby NRTputInHead, 155
 - calledby NRTputInTail, 155
 - calledby setqSingle, 334
 - local ref \$NRTaddForm, 336
 - local ref \$NRTbase, 336
 - local ref \$NRTdeltaLength, 336
 - local ref \$NRTdeltaList, 336
 - local ref \$found, 336
 - defun, 336
- NRTextendsCategory1
 - calledby NRTgetLookupFunction, 191
- NRTgenInitialAttributeAlist
 - calledby compDefineFunctor1, 183
 - calledby finalizeLisplib, 176
- NRTgetLocalIndex, 192
 - calledby compAdd, 256
 - calledby compDefineFunctor1, 183
 - calledby compSymbol, 569
 - calledby doIt, 261
 - calls NRTaddInner, 192
 - calls NRTassocIndex, 192

- calls compOrCroak, 192
- calls rplaca, 192
- local def \$EmptyMode, 192
- local def \$NRTbase, 192
- local def \$e, 192
- local ref \$NRTaddForm, 192
- local ref \$NRTdeltaLength, 192
- local ref \$NRTdeltaListComp, 192
- local ref \$NRTdeltaList, 192
- local ref \$formalArgList, 192
- defun, 192
- NRTgetLookupFunction, 191
 - calledby compDefineFunctor1, 184
 - calls NRTtextendsCategory1, 191
 - calls bright, 191
 - calls form2String, 191
 - calls getExportCategory, 191
 - calls sayBrightlyNT, 191
 - calls sayBrightly, 191
 - calls sublis, 191
 - local def \$why, 191
 - local ref \$pairlis, 191
 - local ref \$why, 191
 - defun, 191
- NRTmakeSlot1Info
 - calledby compDefineFunctor1, 184
- NRTputInHead, 155
 - calledby NRTputInHead, 155
 - calledby NRTputInTail, 155
 - calls NRTassocIndex, 155
 - calls NRTputInHead, 155
 - calls NRTputInTail, 155
 - calls keyedSystemError, 155
 - calls lastnode, 155
 - local ref \$elt, 155
 - defun, 155
- NRTputInTail, 154
 - calledby NRTputInHead, 155
 - calledby putInLocalDomainReferences, 154
 - calls NRTassocIndex, 155
 - calls NRTputInHead, 155
 - calls lassoc, 154
 - calls rplaca, 155
 - local ref \$devalueList, 155
 - local ref \$elt, 155
 - defun, 154
- nsubst
 - calledby substVars, 168
- nth-stack, 524
 - calledby PARSE-Category, 418
 - calledby PARSE-Sexpr1, 436
 - calls reduction-value, 524
 - calls stack-store, 524
 - defmacro, 524
- object2String
 - calledby compileSpad2Cmd, 536
 - calledby compileSpadLispCmd, 596
- opFf
 - calledby parseDEF, 101
- opOf
 - calledby augModemapsFromDomain, 236
 - calledby checkNumOfArgs, 499
 - calledby checkRecordHash, 473
 - calledby coerceSubset, 347
 - calledby comp2, 559
 - calledby compColonInside, 565
 - calledby compDefineCategory2, 142
 - calledby compElt, 300
 - calledby compPretend, 319
 - calledby compilerDoit, 538
 - calledby doIt, 261
 - calledby getSignatureFromMode, 287
 - calledby isFunctor, 233
 - calledby isSuperDomain, 235
 - calledby mkOpVec, 203
 - calledby optCallSpecially, 215
 - calledby parseHas, 108
 - calledby parseLET, 122
 - calledby parseMDEF, 123
 - calledby recordAttributeDocumentation, 463
- opt-, 222
 - defun, 222
- optCall, 214
 - calledby optSPADCALL, 223
 - calls optCallSpecially, 214
 - calls optPackageCall, 214
 - calls optimize, 214
 - calls rplac, 214
 - calls systemErrorHere, 214
 - local ref \$QuickCode, 214

- local ref \$bootStrapMode, 214
 - defun, 214
- optCallEval, 218
 - calledby optSpecialCall, 216
 - calls FactoredForm, 218
 - calls Integer, 218
 - calls List, 218
 - calls Matrix, 218
 - calls PrimitivArray, 218
 - calls Vector, 218
 - calls eval, 218
 - calls qcar, 218
 - defun, 218
- optCallSpecially, 215
 - calledby optCall, 214
 - calls get, 215
 - calls kar, 215
 - calls lassoc, 215
 - calls opOf, 215
 - calls optSpecialCall, 215
 - local ref \$e, 215
 - local ref \$getDomainCode, 215
 - local ref \$optimizableConstructorNames, 215
 - local ref \$specialCaseKeyList, 215
 - defun, 215
- optCatch, 225
 - calls optimize, 225
 - calls qcar, 225
 - calls qcdr, 225
 - calls rplac, 225
 - local ref \$InteractiveMode, 225
 - defun, 225
- optCond, 227
 - calls EqualBarGensym, 227
 - calls TruthP, 227
 - calls qcar, 227
 - calls qcdr, 227
 - calls rplacd, 227
 - calls rplac, 227
 - defun, 227
- optCONDtail, 211
 - calledby optCONDtail, 211
 - calledby optXLAMCond, 210
 - calls optCONDtail, 211
 - local ref \$true, 211
 - defun, 211
- optEQ, 220
 - defun, 220
- optFunctorBody
 - calledby compDefineCategory2, 142
- optIF2COND, 212
 - calledby optIF2COND, 212
 - calledby optimize, 209
 - calls optIF2COND, 212
 - local ref \$true, 212
 - defun, 212
- optimize, 209
 - calledby lispize, 342
 - calledby optCall, 214
 - calledby optCatch, 225
 - calledby optSpecialCall, 216
 - calledby optimizeFunctionDef, 208
 - calledby optimize, 209
 - calls getl, 209
 - calls optIF2COND, 209
 - calls optimize, 209
 - calls prettyprint, 209
 - calls qcar, 209
 - calls qcdr, 209
 - calls rplac, 209
 - calls say, 209
 - calls subrname, 209
 - defun, 209
- optimizeFunctionDef, 208
 - calledby compWithMappingMode1, 586
 - calledby compile, 145
 - calls bright, 208
 - calls optimize, 208
 - calls pp, 208
 - calls qcar, 208
 - calls qcdr, 208
 - calls rplac, 208
 - calls sayBrightlyI, 208
 - local ref \$reportOptimization, 208
 - defun, 208
- optional, 461
 - calledby PARSE-Application, 426
 - calledby PARSE-Category, 417
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Expr, 420

- calledby PARSE-Form, 425
- calledby PARSE-Import, 419
- calledby PARSE-Infix, 423
- calledby PARSE-IteratorTail, 441
- calledby PARSE-Iterator, 441
- calledby PARSE-Prefix, 423
- calledby PARSE-Primary1, 428
- calledby PARSE-PrimaryNoFloat, 428
- calledby PARSE-ScriptItem, 435
- calledby PARSE-Seg, 444
- calledby PARSE-Sequence1, 439
- calledby PARSE-Sexpr1, 436
- calledby PARSE-SpecialCommand, 413
- calledby PARSE-Statement, 416
- calledby PARSE-Suffix, 442
- calledby PARSE-TokenCommandTail, 414
- calledby PARSE-VarForm, 434
- defun, 461
- optionlist
 - usedby spad, 549
- optLESSP, 222
 - defun, 222
- optMINUS, 221
 - defun, 221
- optMkRecord, 229
 - calls length, 229
 - defun, 229
- optPackageCall, 215
 - calledby optCall, 214
 - calls rplaca, 215
 - calls rplacd, 215
 - defun, 215
- optPredicateIfTrue, 211
 - calledby optXLAMCond, 210
 - local ref \$BasicPredicates, 211
 - defun, 211
- optQSMINUS, 221
 - defun, 221
- optRECORDCOPY, 231
 - defun, 231
- optRECORDELT, 230
 - calls keyedSystemError, 230
 - defun, 230
- optSEQ, 218
 - defun, 218
- optSETRECORDELT, 231
 - calls keyedSystemError, 231
 - defun, 231
- optSPADCALL, 223
 - calls optCall, 223
 - local ref \$InteractiveMode, 223
 - defun, 223
- optSpecialCall, 216
 - calledby optCallSpecially, 215
 - calls compileTimeBindingOf, 216
 - calls function, 216
 - calls getl, 216
 - calls keyedSystemError, 216
 - calls mkq, 216
 - calls optCallEval, 216
 - calls optimize, 216
 - calls rplaca, 216
 - calls rplacw, 216
 - calls rplac, 216
 - local ref \$QuickCode, 216
 - local ref \$Undef, 216
 - defun, 216
- optSuchthat, 224
 - defun, 224
- optXLAMCond, 210
 - calledby optXLAMCond, 210
 - calls optCONDtail, 210
 - calls optPredicateIfTrue, 210
 - calls optXLAMCond, 210
 - calls qcar, 210
 - calls qcdr, 210
 - calls rplac, 210
 - defun, 210
- or, 125
 - defplist, 125
- orderByDependency, 207
 - calledby compDefWhereClause, 204
 - calls intersection, 207
 - calls member, 207
 - calls remdup, 207
 - calls say, 207
 - calls userError, 207
 - defun, 207
- orderPredicateItems, 162
 - calledby fixUpPredicate, 161
 - calls orderPredTran, 162
 - calls qcar, 162

- calls qcdr, 162
- calls signatureTran, 162
- defun, 162
- orderPredTran, 163
 - calledby orderPredicateItems, 162
 - calls delete, 163
 - calls insertWOC, 163
 - calls intersectionq, 163
 - calls isDomainSubst, 163
 - calls listOfPatternIds, 163
 - calls member, 163
 - calls qcar, 163
 - calls qcdr, 163
 - calls setdifference, 163
 - calls unionq, 163
 - defun, 163
- outerProduct
 - calledby compileCases, 292
- outputComp, 337
 - calledby compForm1, 571
 - calledby outputComp, 337
 - calledbysetqSingle, 334
 - calls comp, 337
 - calls get, 337
 - calls nreverse0, 337
 - calls outputComp, 337
 - calls qcar, 337
 - calls qcdr, 337
 - local ref \$Expression, 337
 - defun, 337
- pack
 - calledby quote-if-string, 450
- Pair
 - calledby compApply, 563
- pairList
 - calledby compDefWhereClause, 204
 - calledby formal2Pattern, 194
- PARSE-AnyId, 438
 - calledby PARSE-Sexpr1, 436
 - calls action, 438
 - calls advance-token, 438
 - calls current-symbol, 438
 - calls match-string, 438
 - calls parse-identifier, 438
 - calls parse-keyword, 438
 - calls push-reduction, 438
 - defun, 438
- PARSE-Application, 426
 - calledby PARSE-Application, 426
 - calledby PARSE-Category, 418
 - calledby PARSE-Form, 426
 - calls PARSE-Application, 426
 - calls PARSE-Primary, 426
 - calls PARSE-Selector, 426
 - calls optional, 426
 - calls pop-stack-1, 426
 - calls pop-stack-2, 426
 - calls push-reduction, 426
 - calls star, 426
 - defun, 426
- parse-argument-designator, 520
 - calledby PARSE-FormalParameterTok, 433
 - calls advance-token, 521
 - calls match-current-token, 520
 - calls push-reduction, 520
 - calls token-symbol, 520
 - defun, 520
- PARSE-Category, 417
 - calledby PARSE-Category, 417
 - calls PARSE-Application, 418
 - calls PARSE-Category, 417
 - calls PARSE-Expression, 417
 - calls action, 418
 - calls bang, 417
 - calls line-number, 418
 - calls match-advance-string, 417
 - calls must, 417
 - calls nth-stack, 418
 - calls optional, 417
 - calls pop-stack-1, 418
 - calls pop-stack-2, 417
 - calls pop-stack-3, 417
 - calls push-reduction, 417
 - calls recordAttributeDocumentation, 418
 - calls recordSignatureDocumentation, 418
 - calls star, 418
 - uses current-line, 418
 - defun, 417
- PARSE-Command, 412
 - calls PARSE-SpecialCommand, 412
 - calls PARSE-SpecialKeyword, 412

- calls match-advance-string, 412
- calls must, 412
- calls push-reduction, 412
- defun, 412
- PARSE-CommandTail, 415
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-SpecialCommand, 413
 - calls PARSE-CommandTail, 415
 - calls PARSE-Option, 415
 - calls action, 415
 - calls bang, 415
 - calls optional, 415
 - calls pop-stack-1, 415
 - calls pop-stack-2, 415
 - calls push-reduction, 415
 - calls star, 415
 - calls systemCommand[5], 415
 - defun, 415
- PARSE-Conditional, 445
 - calledby PARSE-ElseClause, 445
 - calls PARSE-ElseClause, 445
 - calls PARSE-Expression, 445
 - calls bang, 445
 - calls match-advance-string, 445
 - calls must, 445
 - calls optional, 445
 - calls pop-stack-1, 445
 - calls pop-stack-2, 445
 - calls pop-stack-3, 445
 - calls push-reduction, 445
 - defun, 445
- PARSE-Data, 436
 - calledby PARSE-Primary1, 429
 - calls PARSE-Sexpr, 436
 - calls action, 436
 - calls pop-stack-1, 436
 - calls push-reduction, 436
 - calls translablel, 436
 - uses labasoc, 436
 - defun, 436
- PARSE-ElseClause, 445
 - calledby PARSE-Conditional, 445
 - calls PARSE-Conditional, 445
 - calls PARSE-Expression, 446
 - calls current-symbol, 445
 - defun, 445
- PARSE-Enclosure, 432
 - calledby PARSE-Primary1, 429
 - calls PARSE-Expr, 432
 - calls match-advance-string, 432
 - calls must, 432
 - calls pop-stack-1, 432
 - calls push-reduction, 432
 - defun, 432
- PARSE-Exit, 443
 - calls PARSE-Expression, 443
 - calls match-advance-string, 443
 - calls must, 443
 - calls pop-stack-1, 444
 - calls push-reduction, 443
 - defun, 443
- PARSE-Expr, 420
 - calledby PARSE-Enclosure, 432
 - calledby PARSE-Expression, 419
 - calledby PARSE-Import, 419
 - calledby PARSE-Iterator, 441
 - calledby PARSE-LabelExpr, 446
 - calledby PARSE-Loop, 446
 - calledby PARSE-Primary1, 429
 - calledby PARSE-Reduction, 425
 - calledby PARSE-ScriptItem, 435
 - calledby PARSE-SemiColon, 443
 - calledby PARSE-Statement, 416
 - calls PARSE-LedPart, 420
 - calls PARSE-NudPart, 420
 - calls optional, 420
 - calls pop-stack-1, 420
 - calls push-reduction, 420
 - calls star, 420
 - defun, 420
- PARSE-Expression, 419
 - calledby PARSE-Category, 417
 - calledby PARSE-Conditional, 445
 - calledby PARSE-ElseClause, 446
 - calledby PARSE-Exit, 443
 - calledby PARSE-Infix, 423
 - calledby PARSE-Iterator, 441
 - calledby PARSE-Leave, 444
 - calledby PARSE-Prefix, 423
 - calledby PARSE-Return, 443
 - calledby PARSE-Seg, 444
 - calledby PARSE-Sequence1, 439

- calledby PARSE-SpecialCommand, 413
- calls PARSE-Expr, 419
- calls PARSE-rightBindingPowerOf, 419
- calls make-symbol-of, 419
- calls pop-stack-1, 419
- calls push-reduction, 419
- uses ParseMode, 419
- uses prior-token, 419
- defun, 419
- PARSE-Float, 429
 - calledby PARSE-Primary, 428
 - calledby PARSE-Selector, 427
 - calls PARSE-FloatBase, 429
 - calls PARSE-FloatExponent, 429
 - calls make-float, 429
 - calls must, 429
 - calls pop-stack-1, 429
 - calls pop-stack-2, 429
 - calls pop-stack-3, 429
 - calls pop-stack-4, 429
 - calls push-reduction, 429
 - defun, 429
- PARSE-FloatBase, 430
 - calledby PARSE-Float, 429
 - calls PARSE-FloatBasePart, 430
 - calls PARSE-IntegerTok, 430
 - calls char-eq, 430
 - calls char-ne, 430
 - calls current-char, 430
 - calls current-symbol, 430
 - calls digitp[5], 430
 - calls must, 430
 - calls next-char, 430
 - calls push-reduction, 430
 - defun, 430
- PARSE-FloatBasePart, 430
 - calledby PARSE-FloatBase, 430
 - calls PARSE-IntegerTok, 431
 - calls current-char, 431
 - calls current-token, 431
 - calls digitp[5], 431
 - calls match-advance-string, 430
 - calls must, 431
 - calls push-reduction, 431
 - calls token-nonblank, 431
 - defun, 430
- PARSE-FloatExponent, 431
 - calledby PARSE-Float, 429
 - calls PARSE-IntegerTok, 431
 - calls action, 431
 - calls advance-token, 431
 - calls current-char, 431
 - calls current-symbol, 431
 - calls floatexpid, 431
 - calls identp[5], 431
 - calls match-advance-string, 431
 - calls must, 431
 - calls push-reduction, 431
 - defun, 431
- PARSE-FloatTok, 447
 - calls bfp-, 447
 - calls parse-number, 447
 - calls pop-stack-1, 447
 - calls push-reduction, 447
 - local ref \$boot, 447
 - defun, 447
- PARSE-Form, 425
 - calledby PARSE-NudPart, 420
 - calls PARSE-Application, 426
 - calls bang, 425
 - calls match-advance-string, 425
 - calls must, 425
 - calls optional, 425
 - calls pop-stack-1, 426
 - calls push-reduction, 425
 - defun, 425
- PARSE-FormalParameter, 433
 - calledby PARSE-Primary1, 429
 - calls PARSE-FormalParameterTok, 433
 - defun, 433
- PARSE-FormalParameterTok, 433
 - calledby PARSE-FormalParameter, 433
 - calls parse-argument-designator, 433
 - defun, 433
- PARSE-getSemanticForm, 422
 - calledby PARSE-Operation, 421
 - calls PARSE-Infix, 422
 - calls PARSE-Prefix, 422
 - defun, 422
- PARSE-GlyphTok, 438
 - calledby PARSE-Quad, 433
 - calledby PARSE-Seg, 444

- calledby PARSE-Sexpr1, 437
 - calls action, 438
 - calls advance-token, 438
 - calls match-current-token, 438
 - uses tok, 438
 - defun, 438
- parse-identifier, 519
 - calledby PARSE-AnyId, 438
 - calledby PARSE-Name, 435
 - calls advance-token, 519
 - calls match-current-token, 519
 - calls push-reduction, 519
 - calls token-symbol, 519
 - defun, 519
- PARSE-Import, 419
 - calls PARSE-Expr, 419
 - calls bang, 419
 - calls match-advance-string, 419
 - calls must, 419
 - calls optional, 419
 - calls pop-stack-1, 419
 - calls pop-stack-2, 419
 - calls push-reduction, 419
 - calls star, 419
 - defun, 419
- PARSE-Infix, 423
 - calledby PARSE-getSemanticForm, 422
 - calls PARSE-Expression, 423
 - calls PARSE-TokTail, 423
 - calls action, 423
 - calls advance-token, 423
 - calls current-symbol, 423
 - calls must, 423
 - calls optional, 423
 - calls pop-stack-1, 423
 - calls pop-stack-2, 423
 - calls push-reduction, 423
 - defun, 423
- PARSE-InfixWith, 417
 - calls PARSE-With, 417
 - calls pop-stack-1, 417
 - calls pop-stack-2, 417
 - calls push-reduction, 417
 - defun, 417
- PARSE-IntegerTok, 432
 - calledby PARSE-FloatBasePart, 431
 - calledby PARSE-FloatBase, 430
 - calledby PARSE-FloatExponent, 431
 - calledby PARSE-Primary1, 429
 - calledby PARSE-Sexpr1, 436
 - calls parse-number, 432
 - defun, 432
- PARSE-Iterator, 441
 - calledby PARSE-IteratorTail, 441
 - calledby PARSE-Loop, 446
 - calls PARSE-Expression, 441
 - calls PARSE-Expr, 441
 - calls PARSE-Primary, 441
 - calls match-advance-string, 441
 - calls must, 441
 - calls optional, 441
 - calls pop-stack-1, 441
 - calls pop-stack-2, 441
 - calls pop-stack-3, 441
 - defun, 441
- PARSE-IteratorTail, 441
 - calledby PARSE-Sequence1, 439
 - calls PARSE-Iterator, 441
 - calls bang, 441
 - calls match-advance-string, 441
 - calls optional, 441
 - calls star, 441
 - defun, 441
- parse-keyword, 520
 - calledby PARSE-AnyId, 438
 - calls advance-token, 520
 - calls match-current-token, 520
 - calls push-reduction, 520
 - calls token-symbol, 520
 - defun, 520
- PARSE-Label, 427
 - calledby PARSE-LabelExpr, 446
 - calledby PARSE-Leave, 444
 - calls PARSE-Name, 427
 - calls match-advance-string, 427
 - calls must, 427
 - defun, 427
- PARSE-LabelExpr, 446
 - calls PARSE-Expr, 446
 - calls PARSE-Label, 446
 - calls must, 446
 - calls pop-stack-1, 446

- calls pop-stack-2, 446
- calls push-reduction, 446
- defun, 446
- PARSE-Leave, 444
 - calls PARSE-Expression, 444
 - calls PARSE-Label, 444
 - calls match-advance-string, 444
 - calls must, 444
 - calls pop-stack-1, 444
 - calls push-reduction, 444
 - defun, 444
- PARSE-LedPart, 420
 - calledby PARSE-Expr, 420
 - calls PARSE-Operation, 420
 - calls pop-stack-1, 420
 - calls push-reduction, 420
 - defun, 420
- PARSE-leftBindingPowerOf, 421
 - calledby PARSE-Operation, 421
 - calls elemn, 422
 - calls getl, 421
 - defun, 421
- PARSE-Loop, 446
 - calls PARSE-Expr, 446
 - calls PARSE-Iterator, 446
 - calls match-advance-string, 446
 - calls must, 446
 - calls pop-stack-1, 446
 - calls pop-stack-2, 446
 - calls push-reduction, 446
 - calls star, 446
 - defun, 446
- PARSE-Name, 435
 - calledby PARSE-Label, 427
 - calledby PARSE-VarForm, 434
 - calls parse-identifier, 435
 - calls pop-stack-1, 435
 - calls push-reduction, 435
 - defun, 435
- PARSE-NBGliphTok, 437
 - calledby PARSE-Sexpr1, 436
 - calls action, 437
 - calls advance-token, 437
 - calls match-current-token, 437
 - uses tok, 437
 - defun, 437
- PARSE-NewExpr, 411
 - calledby spad, 549
 - calls PARSE-Statement, 411
 - calls action, 411
 - calls current-symbol, 411
 - calls match-string, 411
 - calls must, 411
 - calls processSynonyms[5], 411
 - uses definition-name, 411
 - defun, 411
- PARSE-NudPart, 420
 - calledby PARSE-Expr, 420
 - calls PARSE-Form, 420
 - calls PARSE-Operation, 420
 - calls PARSE-Reduction, 420
 - calls pop-stack-1, 421
 - calls push-reduction, 420
 - uses rbp, 421
 - defun, 420
- parse-number, 520
 - calledby PARSE-FloatTok, 447
 - calledby PARSE-IntegerTok, 432
 - calls advance-token, 520
 - calls match-current-token, 520
 - calls push-reduction, 520
 - calls token-symbol, 520
 - defun, 520
- PARSE-OpenBrace, 440
 - calledby PARSE-Sequence, 439
 - calls action, 440
 - calls advance-token, 440
 - calls current-symbol, 440
 - calls eqcar, 440
 - calls getToken, 440
 - calls push-reduction, 440
 - defun, 440
- PARSE-OpenBracket, 440
 - calledby PARSE-Sequence, 439
 - calls action, 440
 - calls advance-token, 440
 - calls current-symbol, 440
 - calls eqcar, 440
 - calls getToken, 440
 - calls push-reduction, 440
 - defun, 440
- PARSE-Operation, 421

- calledby PARSE-LedPart, 420
- calledby PARSE-NudPart, 420
- calls PARSE-getSemanticForm, 421
- calls PARSE-leftBindingPowerOf, 421
- calls PARSE-rightBindingPowerOf, 421
- calls action, 421
- calls current-symbol, 421
- calls elemn, 421
- calls getl, 421
- calls lt, 421
- calls match-current-token, 421
- uses ParseMode, 421
- uses rbp, 421
- uses tmptok, 421
- defun, 421
- PARSE-Option, 416
 - calledby PARSE-CommandTail, 415
 - calls PARSE-PrimaryOrQM, 416
 - calls match-advance-string, 416
 - calls must, 416
 - calls star, 416
 - defun, 416
- PARSE-Prefix, 422
 - calledby PARSE-getSemanticForm, 422
 - calls PARSE-Expression, 423
 - calls PARSE-TokTail, 423
 - calls action, 422
 - calls advance-token, 423
 - calls current-symbol, 422
 - calls must, 423
 - calls optional, 423
 - calls pop-stack-1, 423
 - calls pop-stack-2, 423
 - calls push-reduction, 422, 423
 - defun, 422
- PARSE-Primary, 428
 - calledby PARSE-Application, 426
 - calledby PARSE-Iterator, 441
 - calledby PARSE-PrimaryOrQM, 415
 - calledby PARSE-Selector, 427
 - calls PARSE-Float, 428
 - calls PARSE-PrimaryNoFloat, 428
 - defun, 428
- PARSE-Primary1, 428
 - calledby PARSE-Primary1, 428
 - calledby PARSE-PrimaryNoFloat, 428
 - calledby PARSE-Qualification, 424
 - calls PARSE-Data, 429
 - calls PARSE-Enclosure, 429
 - calls PARSE-Expr, 429
 - calls PARSE-FormalParameter, 429
 - calls PARSE-IntegerTok, 429
 - calls PARSE-Primary1, 428
 - calls PARSE-Quad, 429
 - calls PARSE-Sequence, 429
 - calls PARSE-String, 429
 - calls PARSE-VarForm, 428
 - calls current-symbol, 428
 - calls match-advance-string, 429
 - calls match-string, 429
 - calls must, 428
 - calls optional, 428
 - calls pop-stack-1, 428
 - calls pop-stack-2, 428
 - calls push-reduction, 428
 - local ref \$boot, 429
 - defun, 428
- PARSE-PrimaryNoFloat, 428
 - calledby PARSE-Primary, 428
 - calledby PARSE-Selector, 427
 - calls PARSE-Primary1, 428
 - calls PARSE-TokTail, 428
 - calls optional, 428
 - defun, 428
- PARSE-PrimaryOrQM, 415
 - calledby PARSE-Option, 416
 - calledby PARSE-PrimaryOrQM, 415
 - calledby PARSE-SpecialCommand, 413
 - calls PARSE-PrimaryOrQM, 415
 - calls PARSE-Primary, 415
 - calls match-advance-string, 415
 - calls push-reduction, 415
 - defun, 415
- PARSE-Quad, 433
 - calledby PARSE-Primary1, 429
 - calls PARSE-GlyphTok, 433
 - calls match-advance-string, 433
 - calls push-reduction, 433
 - uses \$boot, 433
 - defun, 433
- PARSE-Qualification, 424
 - calledby PARSE-TokTail, 424

- calls PARSE-Primary1, 424
- calls dollarTran, 424
- calls match-advance-string, 424
- calls must, 424
- calls pop-stack-1, 424
- calls push-reduction, 424
- defun, 424
- PARSE-Reduction, 425
 - calledby PARSE-NudPart, 420
 - calls PARSE-Expr, 425
 - calls PARSE-ReductionOp, 425
 - calls must, 425
 - calls pop-stack-1, 425
 - calls pop-stack-2, 425
 - calls push-reduction, 425
 - defun, 425
- PARSE-ReductionOp, 425
 - calledby PARSE-Reduction, 425
 - calls action, 425
 - calls advance-token, 425
 - calls current-symbol, 425
 - calls getl, 425
 - calls match-next-token, 425
 - defun, 425
- PARSE-Return, 443
 - calls PARSE-Expression, 443
 - calls match-advance-string, 443
 - calls must, 443
 - calls pop-stack-1, 443
 - calls push-reduction, 443
 - defun, 443
- PARSE-rightBindingPowerOf, 422
 - calledby PARSE-Expression, 419
 - calledby PARSE-Operation, 421
 - calls elemn, 422
 - calls getl, 422
 - defun, 422
- PARSE-ScriptItem, 435
 - calledby PARSE-ScriptItem, 435
 - calledby PARSE-Scripts, 434
 - calls PARSE-Expr, 435
 - calls PARSE-ScriptItem, 435
 - calls match-advance-string, 435
 - calls must, 435
 - calls optional, 435
 - calls pop-stack-1, 435
 - calls pop-stack-2, 435
 - calls push-reduction, 435
 - defun, 435
- PARSE-Scripts, 434
 - calledby PARSE-VarForm, 434
 - calls PARSE-ScriptItem, 434
 - calls match-advance-string, 434
 - calls must, 434
 - defun, 434
- PARSE-Seg, 444
 - calls PARSE-Expression, 444
 - calls PARSE-GlyphTok, 444
 - calls bang, 444
 - calls optional, 444
 - calls pop-stack-1, 445
 - calls pop-stack-2, 445
 - calls push-reduction, 444
 - defun, 444
- PARSE-Selector, 427
 - calledby PARSE-Application, 426
 - calls PARSE-Float, 427
 - calls PARSE-PrimaryNoFloat, 427
 - calls PARSE-Primary, 427
 - calls char-ne, 427
 - calls current-char, 427
 - calls current-symbol, 427
 - calls match-advance-string, 427
 - calls must, 427
 - calls pop-stack-1, 427
 - calls pop-stack-2, 427
 - calls push-reduction, 427
 - uses \$boot, 427
 - defun, 427
- PARSE-SemiColon, 443
 - calls PARSE-Expr, 443
 - calls match-advance-string, 443
 - calls must, 443
 - calls pop-stack-1, 443
 - calls pop-stack-2, 443
 - calls push-reduction, 443
 - defun, 443
- PARSE-Sequence, 439
 - calledby PARSE-Primary1, 429
 - calls PARSE-OpenBrace, 439
 - calls PARSE-OpenBracket, 439

- calls PARSE-Sequence1, 439
- calls match-advance-string, 439
- calls must, 439
- calls pop-stack-1, 439
- calls push-reduction, 439
- defun, 439
- PARSE-Sequence1, 439
 - calledby PARSE-Sequence, 439
 - calls PARSE-Expression, 439
 - calls PARSE-IteratorTail, 439
 - calls optional, 439
 - calls pop-stack-1, 439
 - calls pop-stack-2, 439
 - calls push-reduction, 439
 - defun, 439
- PARSE-Sexpr, 436
 - calledby PARSE-Data, 436
 - calls PARSE-Sexpr1, 436
 - defun, 436
- PARSE-Sexpr1, 436
 - calledby PARSE-Sexpr1, 436
 - calledby PARSE-Sexpr, 436
 - calls PARSE-AnyId, 436
 - calls PARSE-GlyphTok, 437
 - calls PARSE-IntegerTok, 436
 - calls PARSE-NBGlyphTok, 436
 - calls PARSE-Sexpr1, 436
 - calls PARSE-String, 437
 - calls action, 436
 - calls bang, 437
 - calls match-advance-string, 436
 - calls must, 436
 - calls nth-stack, 436
 - calls optional, 436
 - calls pop-stack-1, 437
 - calls pop-stack-2, 436
 - calls push-reduction, 436
 - calls star, 437
 - defun, 436
- parse-spadstring, 518
 - calledby PARSE-String, 433
 - calls advance-token, 518
 - calls match-current-token, 518
 - calls push-reduction, 518
 - calls token-symbol, 518
 - defun, 518
- PARSE-SpecialCommand, 413
 - calledby PARSE-Command, 412
 - calledby PARSE-SpecialCommand, 413
 - calls PARSE-CommandTail, 413
 - calls PARSE-Expression, 413
 - calls PARSE-PrimaryOrQM, 413
 - calls PARSE-SpecialCommand, 413
 - calls PARSE-TokenCommandTail, 413
 - calls PARSE-TokenList, 413
 - calls action, 413
 - calls bang, 413
 - calls current-symbol, 413
 - calls match-advance-string, 413
 - calls must, 413
 - calls optional, 413
 - calls pop-stack-1, 413
 - calls push-reduction, 413
 - calls star, 413
 - local ref \$noParseCommands, 413
 - local ref \$tokenCommands, 413
 - defun, 413
- PARSE-SpecialKeyWord, 412
 - calledby PARSE-Command, 412
 - calls action, 412
 - calls current-symbol, 412
 - calls current-token, 412
 - calls match-current-token, 412
 - calls token-symbol, 412
 - calls unAbbreviateKeyword[5], 412
 - defun, 412
- PARSE-Statement, 416
 - calledby PARSE-NewExpr, 411
 - calls PARSE-Expr, 416
 - calls match-advance-string, 416
 - calls must, 416
 - calls optional, 416
 - calls pop-stack-1, 416
 - calls pop-stack-2, 416
 - calls push-reduction, 416
 - calls star, 416
 - defun, 416
- PARSE-String, 433
 - calledby PARSE-Primary1, 429
 - calledby PARSE-Sexpr1, 437
 - calls parse-spadstring, 433
 - defun, 433

- parse-string, 519
 - calls advance-token, 519
 - calls match-current-token, 519
 - calls push-reduction, 519
 - calls token-symbol, 519
 - defun, 519
- PARSE-Suffix, 442
 - calls PARSE-TokTail, 442
 - calls action, 442
 - calls advance-token, 442
 - calls current-symbol, 442
 - calls optional, 442
 - calls pop-stack-1, 442
 - calls push-reduction, 442
 - defun, 442
- PARSE-TokenCommandTail, 413
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-TokenCommandTail, 414
 - calls PARSE-TokenCommandTail, 414
 - calls PARSE-TokenOption, 414
 - calls action, 414
 - calls atEndOfLine, 414
 - calls bang, 413
 - calls optional, 414
 - calls pop-stack-1, 414
 - calls pop-stack-2, 414
 - calls push-reduction, 414
 - calls star, 414
 - calls systemCommand[5], 414
 - defun, 413
- PARSE-TokenList, 414
 - calledby PARSE-SpecialCommand, 413
 - calledby PARSE-TokenOption, 414
 - calls action, 414
 - calls advance-token, 414
 - calls current-symbol, 414
 - calls isTokenDelimiter, 414
 - calls push-reduction, 414
 - calls star, 414
 - defun, 414
- PARSE-TokenOption, 414
 - calledby PARSE-TokenCommandTail, 414
 - calls PARSE-TokenList, 414
 - calls match-advance-string, 414
 - calls must, 414
 - defun, 414
- PARSE-TokTail, 424
 - calledby PARSE-Infix, 423
 - calledby PARSE-Prefix, 423
 - calledby PARSE-PrimaryNoFloat, 428
 - calledby PARSE-Suffix, 442
 - calls PARSE-Qualification, 424
 - calls action, 424
 - calls char-eq, 424
 - calls copy-token, 424
 - calls current-char, 424
 - calls current-symbol, 424
 - uses \$boot, 424
 - defun, 424
- PARSE-VarForm, 434
 - calledby PARSE-Primary1, 428
 - calls PARSE-Name, 434
 - calls PARSE-Scripts, 434
 - calls optional, 434
 - calls pop-stack-1, 434
 - calls pop-stack-2, 434
 - calls push-reduction, 434
 - defun, 434
- PARSE-With, 417
 - calledby PARSE-InfixWith, 417
 - calls match-advance-string, 417
 - calls must, 417
 - calls pop-stack-1, 417
 - calls push-reduction, 417
 - defun, 417
- parseAnd, 97
 - calledby parseAnd, 97
 - calls parseAnd, 97
 - calls parseIf, 97
 - calls parseTranList, 97
 - calls parseTran, 97
 - uses \$InteractiveMode, 97
 - defun, 97
- parseAtom, 94
 - calledby parseTran, 93
 - calls parseLeave, 94
 - uses \$NoValue, 94
 - defun, 94
- parseAtSign, 98
 - calls parseTran, 98
 - calls parseType, 98
 - uses \$InteractiveMode, 98

- defun, 98
- parseCategory, 99
 - calls contained, 99
 - calls parseDropAssertions, 99
 - calls parseTranList, 99
 - defun, 99
- parseCoerce, 100
 - calls parseTran, 100
 - calls parseType, 100
 - uses \$InteractiveMode, 100
 - defun, 100
- parseColon, 100
 - calls parseTran, 100
 - calls parseType, 100
 - local ref \$insideConstructIfTrue, 100
 - uses \$InteractiveMode, 100
 - defun, 100
- parseConstruct, 95
 - calledby parseTran, 93
 - calls parseTranList, 95
 - uses \$insideConstructIfTrue, 95
 - defun, 95
- parseDEF, 101
 - calls opFf, 101
 - calls parseLhs, 101
 - calls parseTranCheckForRecord, 101
 - calls parseTranList, 101
 - calls setDefOp, 101
 - uses \$lhs, 101
 - defun, 101
- parseDollarGreaterEqual, 104
 - calls msubst, 104
 - calls parseTran, 104
 - uses \$op, 104
 - defun, 104
- parseDollarGreaterThan, 104
 - calls msubst, 104
 - calls parseTran, 104
 - uses \$op, 104
 - defun, 104
- parseDollarLessEqual, 105
 - calls msubst, 105
 - calls parseTran, 105
 - uses \$op, 105
 - defun, 105
- parseDollarNotEqual, 105
 - calls msubst, 106
 - calls parseTran, 105
 - uses \$op, 106
 - defun, 105
- parseDropAssertions, 99
 - calledby parseCategory, 99
 - calledby parseDropAssertions, 99
 - calls parseDropAssertions, 99
 - defun, 99
- parseEquivalence, 106
 - calls parseIf, 106
 - defun, 106
- parseExit, 107
 - calls moan, 107
 - calls parseTran, 107
 - defun, 107
- parseGreaterEqual, 107
 - calls parseTran, 107
 - uses \$op, 107
 - defun, 107
- parseGreaterThan, 108
 - calls parseTran, 108
 - uses \$op, 108
 - defun, 108
- parseHas, 108
 - calls getdatabase, 108
 - calls makeNonAtomic, 109
 - calls member, 109
 - calls nreverse0, 109
 - calls opOf, 108
 - calls parseHasRhs, 109
 - calls parseType, 109
 - calls qcar, 108
 - calls qcdr, 108
 - calls unabbrevAndLoad, 108
 - uses \$CategoryFrame, 109
 - uses \$InteractiveMode, 109
 - defun, 108
- parseHasRhs, 110
 - calledby parseHas, 109
 - calls abbreviation?, 110
 - calls get, 110
 - calls loadIfNecessary, 110
 - calls member, 110
 - calls qcar, 110
 - calls qcdr, 110

- calls unabbrevAndLoad, 110
- uses \$CategoryFrame, 110
- defun, 110
- parseIf, 114
 - calledby parseAnd, 97
 - calledby parseEquivalence, 106
 - calledby parseImplies, 116
 - calledby parseOr, 126
 - calls parseIf,ifTran, 114
 - calls parseTran, 114
 - defun, 114
- parseIf,ifTran, 114
 - calledby parseIf,ifTran, 114
 - calledby parseIf, 114
 - calls incExitLevel, 114
 - calls makeSimplePredicateOrNil, 114
 - calls parseIf,ifTran, 114
 - calls parseTran, 114
 - uses \$InteractiveMode, 114
 - defun, 114
- parseImplies, 116
 - calls parseIf, 116
 - defun, 116
- parseIn, 117
 - calledby parseInBy, 118
 - calls parseTran, 117
 - calls postError, 117
 - defun, 117
- parseInBy, 118
 - calls bright, 118
 - calls parseIn, 118
 - calls parseTran, 118
 - calls postError, 118
 - defun, 118
- parseIs, 119
 - calls parseTran, 119
 - calls transIs, 119
 - defun, 119
- parseIsnt, 120
 - calls parseTran, 120
 - calls transIs, 120
 - defun, 120
- parseJoin, 120
 - calls parseTranList, 120
 - defun, 120
- parseLeave, 121
 - calledby parseAtom, 94
 - calls parseTran, 121
 - defun, 121
- parseLessEqual, 122
 - calls parseTran, 122
 - uses \$op, 122
 - defun, 122
- parseLET, 122
 - calls opOf, 122
 - calls parseTranCheckForRecord, 122
 - calls parseTran, 122
 - calls transIs, 122
 - defun, 122
- parseLETD, 123
 - calls parseTran, 123
 - calls parseType, 123
 - defun, 123
- parseLhs, 102
 - calledby parseDEF, 101
 - calls parseTran, 102
 - calls transIs, 102
 - defun, 102
- parseMDEF, 123
 - calls opOf, 123
 - calls parseTranCheckForRecord, 123
 - calls parseTranList, 123
 - calls parseTran, 123
 - uses \$lhs, 123
 - defun, 123
- ParseMode, 411
 - usedby PARSE-Expression, 419
 - usedby PARSE-Operation, 421
 - defvar, 411
- parseNot, 124
 - calls parseTran, 124
 - uses \$InteractiveMode, 124
 - defun, 124
- parseNotEqual, 125
 - calls msubst, 125
 - calls parseTran, 125
 - uses \$op, 125
 - defun, 125
- parseOr, 125
 - calledby parseOr, 126
 - calls parseIf, 126
 - calls parseOr, 126

- calls parseTranList, 126
- calls parseTran, 125
- defun, 125
- parsepiles, 84
 - calledby preparse1, 81
 - calls add-parens-and-semis-to-line, 84
 - defun, 84
- parsePretend, 126
 - calls parseTran, 126
 - calls parseType, 126
 - defun, 126
- parseprint, 528
 - calledby preparse, 76
 - defun, 528
- parseReturn, 127
 - calls moan, 127
 - calls parseTran, 127
 - defun, 127
- parseSegment, 128
 - calls parseTran, 128
 - defun, 128
- parseSeq, 128
 - calls last, 128
 - calls mapInto, 128
 - calls postError, 128
 - calls transSeq, 128
 - defun, 128
- parseTran, 93
 - calledby compReduce1, 320
 - calledby parseAnd, 97
 - calledby parseAtSign, 98
 - calledby parseCoerce, 100
 - calledby parseColon, 100
 - calledby parseDollarGreaterEqual, 104
 - calledby parseDollarGreaterThan, 104
 - calledby parseDollarLessEqual, 105
 - calledby parseDollarNotEqual, 105
 - calledby parseExit, 107
 - calledby parseGreaterEqual, 107
 - calledby parseGreaterThan, 108
 - calledby parseIf,ifTran, 114
 - calledby parseIf, 114
 - calledby parseInBy, 118
 - calledby parseIn, 117
 - calledby parseIsnt, 120
 - calledby parseIs, 119
 - calledby parseLETD, 123
 - calledby parseLET, 122
 - calledby parseLeave, 121
 - calledby parseLessEqual, 122
 - calledby parseLhs, 102
 - calledby parseMDEF, 123
 - calledby parseNotEqual, 125
 - calledby parseNot, 124
 - calledby parseOr, 125
 - calledby parsePretend, 126
 - calledby parseReturn, 127
 - calledby parseSegment, 128
 - calledby parseTranCheckForRecord, 517
 - calledby parseTranList, 95
 - calledby parseTransform, 93
 - calledby parseTran, 93
 - calledby parseType, 98
 - calls get1, 93
 - calls parseAtom, 93
 - calls parseConstruct, 93
 - calls parseTranList, 93
 - calls parseTran, 93
 - uses \$op, 93
 - defun, 93
- parseTranCheckForRecord, 517
 - calledby parseDEF, 101
 - calledby parseLET, 122
 - calledby parseMDEF, 123
 - calls parseTran, 517
 - calls postError, 517
 - calls qcar, 517
 - calls qcdr, 517
 - defun, 517
- parseTranList, 95
 - calledby parseAnd, 97
 - calledby parseCategory, 99
 - calledby parseConstruct, 95
 - calledby parseDEF, 101
 - calledby parseJoin, 120
 - calledby parseMDEF, 123
 - calledby parseOr, 126
 - calledby parseTranList, 95
 - calledby parseTran, 93
 - calledby parseVCONS, 129
 - calls parseTranList, 95
 - calls parseTran, 95

- defun, 95
- parseTransform, 93
 - calledby s-process, 550
 - calls msubst, 93
 - calls parseTran, 93
 - uses \$defOp, 93
 - defun, 93
- parseType, 98
 - calledby parseAtSign, 98
 - calledby parseCoerce, 100
 - calledby parseColon, 100
 - calledby parseHas, 109
 - calledby parseLETD, 123
 - calledby parsePretend, 126
 - calls msubst, 98
 - calls parseTran, 98
 - defun, 98
- parseVCONS, 129
 - calls parseTranList, 129
 - defun, 129
- parseWhere, 129
 - calls mapInto, 129
 - defun, 129
- pathname[5]
 - called by compileSpad2Cmd, 536
 - called by compileSpadLispCmd, 596
 - called by compiler, 533
- pathnameDirectory[5]
 - called by compileSpadLispCmd, 596
- pathnameName[5]
 - called by compileSpadLispCmd, 596
- pathnameType[5]
 - called by compileSpad2Cmd, 536
 - called by compileSpadLispCmd, 596
 - called by compiler, 533
- pathnameTypeId
 - calledby initializeLisplib, 175
- pmatch
 - calledby coerceable, 350
- pmatchWithSl
 - calledby compApplyModemap, 239
- pname
 - calledby checkTransformFirsts, 488
 - calledby compDefineFunctor1, 183
 - calledby encodeItem, 150
 - calledby isCategoryPackageName, 190
 - calledby mkCategoryPackage, 139
 - calledby recordAttributeDocumentation, 463
- pname[5]
 - called by comp3, 560
 - called by floatexpid, 459
 - called by getScriptName, 399
- Pop-Reduction, 524
 - calledby pop-stack-1, 522
 - calledby pop-stack-2, 523
 - calledby pop-stack-3, 523
 - calledby pop-stack-4, 523
 - calls stack-pop, 524
 - defun, 524
- pop-stack-1, 522
 - calledby PARSE-Application, 426
 - calledby PARSE-Category, 418
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Data, 436
 - calledby PARSE-Enclosure, 432
 - calledby PARSE-Exit, 444
 - calledby PARSE-Expression, 419
 - calledby PARSE-Expr, 420
 - calledby PARSE-FloatTok, 447
 - calledby PARSE-Float, 429
 - calledby PARSE-Form, 426
 - calledby PARSE-Import, 419
 - calledby PARSE-InfixWith, 417
 - calledby PARSE-Infix, 423
 - calledby PARSE-Iterator, 441
 - calledby PARSE-LabelExpr, 446
 - calledby PARSE-Leave, 444
 - calledby PARSE-LedPart, 420
 - calledby PARSE-Loop, 446
 - calledby PARSE-Name, 435
 - calledby PARSE-NudPart, 421
 - calledby PARSE-Prefix, 423
 - calledby PARSE-Primary1, 428
 - calledby PARSE-Qualification, 424
 - calledby PARSE-Reduction, 425
 - calledby PARSE-Return, 443
 - calledby PARSE-ScriptItem, 435
 - calledby PARSE-Seg, 445
 - calledby PARSE-Selector, 427
 - calledby PARSE-SemiColon, 443

- calledby PARSE-Sequence1, 439
- calledby PARSE-Sequence, 439
- calledby PARSE-Sexpr1, 437
- calledby PARSE-SpecialCommand, 413
- calledby PARSE-Statement, 416
- calledby PARSE-Suffix, 442
- calledby PARSE-TokenCommandTail, 414
- calledby PARSE-VarForm, 434
- calledby PARSE-With, 417
- calledby spad, 549
- calledby star, 461
- calls Pop-Reduction, 522
- calls reduction-value, 522
- defmacro, 522
- pop-stack-2, 523
 - calledby PARSE-Application, 426
 - calledby PARSE-Category, 417
 - calledby PARSE-CommandTail, 415
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Float, 429
 - calledby PARSE-Import, 419
 - calledby PARSE-InfixWith, 417
 - calledby PARSE-Infix, 423
 - calledby PARSE-Iterator, 441
 - calledby PARSE-LabelExpr, 446
 - calledby PARSE-Loop, 446
 - calledby PARSE-Prefix, 423
 - calledby PARSE-Primary1, 428
 - calledby PARSE-Reduction, 425
 - calledby PARSE-ScriptItem, 435
 - calledby PARSE-Seg, 445
 - calledby PARSE-Selector, 427
 - calledby PARSE-SemiColon, 443
 - calledby PARSE-Sequence1, 439
 - calledby PARSE-Sexpr1, 436
 - calledby PARSE-Statement, 416
 - calledby PARSE-TokenCommandTail, 414
 - calledby PARSE-VarForm, 434
 - calls Pop-Reduction, 523
 - calls reduction-value, 523
 - calls stack-push, 523
 - defmacro, 523
- pop-stack-3, 523
 - calledby PARSE-Category, 417
 - calledby PARSE-Conditional, 445
 - calledby PARSE-Float, 429
- calledby PARSE-Iterator, 441
- calls Pop-Reduction, 523
- calls reduction-value, 523
- calls stack-push, 523
- defmacro, 523
- pop-stack-4, 523
 - calledby PARSE-Float, 429
 - calls Pop-Reduction, 523
 - calls reduction-value, 523
 - calls stack-push, 523
 - defmacro, 523
- postAdd, 366
 - calls postCapsule, 366
 - calls postTran, 366
 - defun, 366
- postAtom, 361
 - calledby postTran, 360
 - uses \$boot, 361
 - defun, 361
- postAtSign, 369
 - calls postTran, 369
 - calls postType, 369
 - defun, 369
- postBigFloat, 370
 - calls postTran, 370
 - uses \$InteractiveMode, 370
 - uses \$boot, 370
 - defun, 370
- postBlock, 370
 - calledby postSemiColon, 389
 - calls postBlockItemList, 370
 - calls postTran, 370
 - defun, 370
- postBlockItem, 368
 - calledby postBlockItemList, 367
 - calledby postCapsule, 367
 - calls postTran, 368
 - defun, 368
- postBlockItemList, 367
 - calledby postBlock, 370
 - calledby postCapsule, 367
 - calls postBlockItem, 367
 - defun, 367
- postCapsule, 367
 - calledby postAdd, 366
 - calls checkWarning, 367

- calls postBlockItemList, 367
- calls postBlockItem, 367
- calls postFlatten, 367
- defun, 367
- postCategory, 371
 - calls nreverse0, 371
 - calls postTran, 371
 - uses \$insidePostCategoryIfTrue, 371
 - defun, 371
- postcheck, 363
 - calledby postTransformCheck, 363
 - calledby postcheck, 363
 - calls postcheck, 363
 - calls setDefOp, 363
 - defun, 363
- postCollect, 373
 - calledby postCollect, 373
 - calledby postTupleCollect, 393
 - calls postCollect,finish, 373
 - calls postCollect, 373
 - calls postIteratorList, 373
 - calls postTran, 373
 - defun, 373
- postCollect,finish, 372
 - calledby postCollect, 373
 - calls postMakeCons, 372
 - calls postTranList, 372
 - calls qcar, 372
 - calls qcdr, 372
 - calls tuple2List, 372
 - defun, 372
- postColon, 375
 - calls postTran, 375
 - calls postType, 375
 - defun, 375
- postColonColon, 375
 - calls postForm, 375
 - uses \$boot, 375
 - defun, 375
- postComma, 376
 - calls comma2Tuple, 376
 - calls postTuple, 376
 - defun, 376
- postConstruct, 377
 - calls comma2Tuple, 377
 - calls postMakeCons, 377
 - calls postTranList, 377
 - calls postTranSegment, 377
 - calls postTran, 377
 - calls tuple2List, 377
 - defun, 377
- postDef, 378
 - calls nequal, 379
 - calls nreverse0, 379
 - calls postDefArgs, 379
 - calls postMDef, 378
 - calls postTran, 379
 - calls recordHeaderDocumentation, 379
 - uses \$InteractiveMode, 379
 - uses \$boot, 379
 - uses \$docList, 379
 - uses \$headerDocumentation, 379
 - uses \$maxSignatureLineNumber, 379
 - defun, 378
- postDefArgs, 380
 - calledby postDefArgs, 380
 - calledby postDef, 379
 - calls postDefArgs, 380
 - calls postError, 380
 - defun, 380
- postError, 364
 - calledby checkWarning, 521
 - calledby getScriptName, 399
 - calledby parseInBy, 118
 - calledby parseIn, 117
 - calledby parseSeq, 128
 - calledby parseTranCheckForRecord, 517
 - calledby postDefArgs, 380
 - calledby postForm, 364
 - calls bumperrorcount, 364
 - calls nequal, 364
 - uses \$InteractiveMode, 364
 - uses \$defOp, 364
 - uses \$postStack, 364
 - defun, 364
- postExit, 381
 - calls postTran, 381
 - defun, 381
- postFlatten, 376
 - calledby comma2Tuple, 376
 - calledby postCapsule, 367
 - calledby postFlatten, 376

- calls postFlatten, 376
- defun, 376
- postFlattenLeft, 389
 - calledby postFlattenLeft, 389
 - calledby postSemiColon, 389
 - calls postFlattenLeft, 389
 - defun, 389
- postForm, 364
 - calledby postColonColon, 375
 - calledby postTran, 360
 - calls bright, 364
 - calls intl, 364
 - calls postError, 364
 - calls postTranList, 364
 - calls postTran, 364
 - uses \$boot, 364
 - defun, 364
- postIf, 381
 - calls nreverse0, 381
 - calls postTran, 381
 - uses \$boot, 381
 - defun, 381
- postIn, 383
 - calls postInSeq, 383
 - calls postTran, 383
 - calls systemErrorHere, 383
 - defun, 383
- postin, 382
 - calls postInSeq, 382
 - calls postTran, 382
 - calls systemErrorHere, 382
 - defun, 382
- postInSeq, 382
 - calledby postIn, 383
 - calledby postIteratorList, 374
 - calledby postin, 382
 - calls postTranSegment, 382
 - calls postTran, 382
 - calls tuple2List, 382
 - defun, 382
- postIteratorList, 374
 - calledby postCollect, 373
 - calledby postIteratorList, 374
 - calledby postRepeat, 388
 - calls postInSeq, 374
 - calls postIteratorList, 374
 - calls postTran, 374
 - defun, 374
- postJoin, 384
 - calls postTranList, 384
 - calls postTran, 384
 - defun, 384
- postMakeCons, 372
 - calledby postCollect, finish, 372
 - calledby postConstruct, 377
 - calledby postMakeCons, 372
 - calls postMakeCons, 372
 - calls postTran, 372
 - defun, 372
- postMapping, 384
 - calls postTran, 384
 - calls unTuple, 384
 - defun, 384
- postMDef, 385
 - calledby postDef, 378
 - calls nreverse0, 385
 - calls postTran, 385
 - calls throwkeyedmsg, 385
 - uses \$InteractiveMode, 385
 - uses \$boot, 385
 - defun, 385
- postOp, 361
 - calledby postTran, 360
 - defun, 361
- postPretend, 386
 - calls postTran, 386
 - calls postType, 386
 - defun, 386
- postQUOTE, 387
 - defun, 387
- postReduce, 387
 - calledby postReduce, 387
 - calls postReduce, 387
 - calls postTran, 387
 - uses \$InteractiveMode, 387
 - defun, 387
- postRepeat, 388
 - calls postIteratorList, 388
 - calls postTran, 388
 - defun, 388
- postScripts, 389
 - calls getScriptName, 389

- calls postTranScripts, 389
- defun, 389
- postScriptsForm, 362
 - calledby postTran, 360
 - calls getScriptName, 362
 - calls length, 362
 - calls postTranScripts, 362
 - defun, 362
- postSemiColon, 389
 - calls postBlock, 389
 - calls postFlattenLeft, 389
 - defun, 389
- postSignature, 390
 - calls killColons, 390
 - calls postType, 390
 - calls removeSuperfluousMapping, 390
 - defun, 390
- postSlash, 391
 - calls postTran, 391
 - defun, 391
- postTran, 360
 - calledby postAdd, 366
 - calledby postAtSign, 369
 - calledby postBigFloat, 370
 - calledby postBlockItem, 368
 - calledby postBlock, 370
 - calledby postCategory, 371
 - calledby postCollect, 373
 - calledby postColon, 375
 - calledby postConstruct, 377
 - calledby postDef, 379
 - calledby postExit, 381
 - calledby postForm, 364
 - calledby postIf, 381
 - calledby postInSeq, 382
 - calledby postIn, 383
 - calledby postIteratorList, 374
 - calledby postJoin, 384
 - calledby postMDef, 385
 - calledby postMakeCons, 372
 - calledby postMapping, 384
 - calledby postPretend, 386
 - calledby postReduce, 387
 - calledby postRepeat, 388
 - calledby postSlash, 391
 - calledby postTranList, 362
 - calledby postTranScripts, 362
 - calledby postTranSegment, 378
 - calledby postTransform, 359
 - calledby postType, 369
 - calledby postWhere, 393
 - calledby postWith, 394
 - calledby postin, 382
 - calledby tuple2List, 522
 - calls postAtom, 360
 - calls postForm, 360
 - calls postOp, 360
 - calls postScriptsForm, 360
 - calls postTranList, 360
 - calls postTran, 360
 - calls qcar, 360
 - calls qcdr, 360
 - calls unTuple, 360
 - defun, 360
- postTranList, 362
 - calledby postCollect,finish, 372
 - calledby postConstruct, 377
 - calledby postForm, 364
 - calledby postJoin, 384
 - calledby postTran, 360
 - calledby postTuple, 392
 - calledby postWhere, 393
 - calls postTran, 362
 - defun, 362
- postTranScripts, 362
 - calledby postScriptsForm, 362
 - calledby postScripts, 389
 - calledby postTranScripts, 362
 - calls postTranScripts, 362
 - calls postTran, 362
 - defun, 362
- postTranSegment, 378
 - calledby postConstruct, 377
 - calledby postInSeq, 382
 - calledby tuple2List, 521
 - calls postTran, 378
 - defun, 378
- postTransform, 359
 - calledby new2OldLisp, 518
 - calledby recordAttributeDocumentation, 463

- calledby recordSignatureDocumentation, 463
- calledby s-process, 550
- calls aplTran, 359
- calls identp[5], 359
- calls postTransformCheck, 359
- calls postTran, 359
- defun, 359
- postTransformCheck, 363
 - calledby postTransform, 359
 - calls postcheck, 363
 - uses \$defOp, 363
 - defun, 363
- postTuple, 392
 - calledby postComma, 376
 - calls postTranList, 392
 - defun, 392
- postTupleCollect, 393
 - calls postCollect, 393
 - defun, 393
- postType, 369
 - calledby postAtSign, 369
 - calledby postColon, 375
 - calledby postPretend, 386
 - calledby postSignature, 390
 - calls postTran, 369
 - calls unTuple, 369
 - defun, 369
- postWhere, 393
 - calls postTranList, 393
 - calls postTran, 393
 - defun, 393
- postWith, 394
 - calls postTran, 394
 - uses \$insidePostCategoryIfTrue, 394
 - defun, 394
- pp
 - calledby checkComments, 481
 - calledby compDefineFunctor1, 183
 - calledby optimizeFunctionDef, 208
- PredImplies
 - calledby compForm2, 579
- preparse, 71, 76
 - calledby preparse, 76
 - calledby spad, 549
 - calls ifcar, 76
 - calls parseprint, 76
 - calls preparse1, 76
 - calls preparse, 76
 - uses \$comblocklist, 77
 - uses \$constructorLineNumber, 77
 - uses \$docList, 77
 - uses \$headerDocumentation, 77
 - uses \$index, 77
 - uses \$maxSignatureLineNumber, 77
 - uses \$preparse-last-line, 77
 - uses \$preparseReportIfTrue, 77
 - uses \$skipme, 77
 - defun, 76
- preparse-echo, 88
 - calledby fincomblock, 526
 - calledby preparse1, 81
 - uses Echo-Meta, 88
 - uses \$EchoLineStack, 88
 - defun, 88
- preparse1, 81
 - calledby preparse, 76
 - calls doSystemCommand[5], 81
 - calls escaped, 81
 - calls fincomblock, 81
 - calls indent-pos, 81
 - calls is-console, 81
 - calls make-full-cvec, 81
 - calls maxindex, 81
 - calls parsepiles, 81
 - calls preparse-echo, 81
 - calls preparseReadLine, 81
 - calls strposl[5], 81
 - local def \$index, 81
 - local def \$preparse-last-line, 81
 - local def \$skipme, 81
 - local ref \$byConstructors, 81
 - local ref \$constructorsSeen, 81
 - local ref \$echolinestack, 81
 - local ref \$in-stream, 81
 - local ref \$index, 81
 - local ref \$linelist, 81
 - local ref \$preparse-last-line, 81
 - catches, 81
 - defun, 81
- preparseReadLine, 85
 - calledby preparse1, 81

- calledby `preparseReadLine`, 85
- calledby `skip-to-endif`, 528
- calls `dcq`, 85
- calls `initial-substring`, 85
- calls `preparseReadLine1`, 85
- calls `preparseReadLine`, 85
- calls `skip-to-endif`, 85
- calls `storeblanks`, 85
- calls `string2BootTree`, 85
- local ref `$*eof*`, 85
- defun, 85
- `preparseReadLine1`, 87
 - calledby `preparseReadLine1`, 87
 - calledby `preparseReadLine`, 85
 - calledby `skip-ifblock`, 86
 - calledby `skip-to-endif`, 528
 - calls `expand-tabs`, 87
 - calls `get-a-line`, 87
 - calls `maxindex`, 87
 - calls `preparseReadLine1`, 87
 - calls `strconc`, 87
 - uses `$EchoLineStack`, 87
 - uses `$index`, 87
 - uses `$linelist`, 87
 - uses `$preparse-last-line`, 87
 - defun, 87
- `pretend`, 126, 318, 386
 - `defplist`, 126, 318, 386
- `prettyprint`
 - calledby `optimize`, 209
 - calledby `s-process`, 550
- `PrimitivteArray`
 - calledby `optCallEval`, 218
- `primitiveType`, 568
 - calledby `compAtom`, 565, 566
 - uses `$DoubleFloat`, 568
 - uses `$EmptyMode`, 568
 - uses `$NegativeInteger`, 568
 - uses `$NonNegativeInteger`, 568
 - uses `$PositiveInteger`, 568
 - uses `$String`, 568
 - defun, 568
- `print-defun`, 553
 - calls `is-console`, 553
 - calls `print-full`, 553
 - uses `$PrettyPrint`, 553
 - uses `vmisp::optionlist`, 553
 - defun, 553
- `print-full`
 - calledby `print-defun`, 553
- `print-package`, 521
 - local ref `$out-stream`, 521
 - defun, 521
- `printSignature`
 - calledby `getSignature`, 296
- `printStats`
 - calledby `compile`, 146
- `prior-token`, 90
 - usedby `PARSE-Expression`, 419
 - uses `$token`, 90
 - defvar, 90
- `processFuncor`, 259
 - calledby `compCapsuleInner`, 259
 - calls `buildFuncor`, 259
 - calls `error`, 259
 - defun, 259
- `processInteractive[5]`
 - called by `s-process`, 551
- `processSynonyms[5]`
 - called by `PARSE-NewExpr`, 411
- `profileRecord`
 - calledby `compDefineCapsuleFunction`, 288
 - calledby `setqSingle`, 334
- `profileWrite`
 - calledby `finalizeLisplib`, 176
- `push-reduction`, 462
 - calledby `PARSE-AnyId`, 438
 - calledby `PARSE-Application`, 426
 - calledby `PARSE-Category`, 417
 - calledby `PARSE-CommandTail`, 415
 - calledby `PARSE-Command`, 412
 - calledby `PARSE-Conditional`, 445
 - calledby `PARSE-Data`, 436
 - calledby `PARSE-Enclosure`, 432
 - calledby `PARSE-Exit`, 443
 - calledby `PARSE-Expression`, 419
 - calledby `PARSE-Expr`, 420
 - calledby `PARSE-FloatBasePart`, 431
 - calledby `PARSE-FloatBase`, 430
 - calledby `PARSE-FloatExponent`, 431
 - calledby `PARSE-FloatTok`, 447
 - calledby `PARSE-Float`, 429

- calledby PARSE-Form, 425
- calledby PARSE-Import, 419
- calledby PARSE-InfixWith, 417
- calledby PARSE-Infix, 423
- calledby PARSE-LabelExpr, 446
- calledby PARSE-Leave, 444
- calledby PARSE-LedPart, 420
- calledby PARSE-Loop, 446
- calledby PARSE-Name, 435
- calledby PARSE-NudPart, 420
- calledby PARSE-OpenBrace, 440
- calledby PARSE-OpenBracket, 440
- calledby PARSE-Prefix, 422, 423
- calledby PARSE-Primary1, 428
- calledby PARSE-PrimaryOrQM, 415
- calledby PARSE-Quad, 433
- calledby PARSE-Qualification, 424
- calledby PARSE-Reduction, 425
- calledby PARSE-Return, 443
- calledby PARSE-ScriptItem, 435
- calledby PARSE-Seg, 444
- calledby PARSE-Selector, 427
- calledby PARSE-SemiColon, 443
- calledby PARSE-Sequence1, 439
- calledby PARSE-Sequence, 439
- calledby PARSE-Sexpr1, 436
- calledby PARSE-SpecialCommand, 413
- calledby PARSE-Statement, 416
- calledby PARSE-Suffix, 442
- calledby PARSE-TokenCommandTail, 414
- calledby PARSE-TokenList, 414
- calledby PARSE-VarForm, 434
- calledby PARSE-With, 417
- calledby parse-argument-designator, 520
- calledby parse-identifier, 519
- calledby parse-keyword, 520
- calledby parse-number, 520
- calledby parse-spadstring, 518
- calledby parse-string, 519
- calledby star, 461
- calls make-reduction, 462
- calls stack-push, 462
- uses reduce-stack, 462
- defun, 462
- put
 - calledby checkAndDeclare, 298
 - calledby compColon, 275
 - calledby compDefineCapsuleFunction, 288
 - calledby compMacro, 317
 - calledby compSubsetCategory, 342
 - calledby compSuchthat, 343
 - calledby compTypeOf, 564
 - calledby doIt, 261
 - calledby evalAndSub, 249
 - calledby getInverseEnvironment, 310
 - calledby getSuccessEnvironment, 308
 - calledby giveFormalParametersValues, 135
 - calledby putDomainsInScope, 235
 - calledby setqMultiple, 331
 - calledby updateCategoryFrameForCategory, 113
 - calledby updateCategoryFrameForConstructor, 112
- putDomainsInScope, 235
 - calledby addConstructorModemaps, 238
 - calledby augModemapsFromCategoryRep, 252
 - calledby augModemapsFromCategory, 245
- calls delete, 235
- calls getDomainsInScope, 235
- calls member, 235
- calls put, 235
- calls say, 235
- local def \$CapsuleDomainsInScope, 235
- local ref \$insideCapsuleFunctionIfTrue, 235
- defun, 235
- putInLocalDomainReferences, 154
 - calledby compile, 145
 - calls NRTputInTail, 154
 - local def \$elt, 154
 - local ref \$QuickCode, 154
 - defun, 154
- qcar
 - calledby TruthP, 249
 - calledby addArgumentConditions, 293
 - calledby addConstructorModemaps, 238
 - calledby addEltModemap, 245
 - calledby addModemap0, 254
 - calledby autoCoerceByModemap, 354
 - calledby canReturn, 306

- calledby coerceByModemap, 354
- calledby coerceExtraHard, 349
- calledby compAdd, 256
- calledby compCategoryItem, 271
- calledby compCategory, 270
- calledby compCons1, 279
- calledby compDefWhereClause, 204
- calledby compDefineFunctor1, 184
- calledby compHasFormat, 303
- calledby compJoin, 313
- calledby compLambda, 315
- calledby compMacro, 317
- calledby compSetq1, 329
- calledby compWithMappingMode1, 586
- calledby compileCases, 291
- calledby compile, 145
- calledby decodeScripts, 399
- calledby doIt, 261
- calledby eltModemapFilter, 577
- calledby encodeItem, 150
- calledby fixUpPredicate, 161
- calledby flattenSignatureList, 159
- calledby genDomainView, 201
- calledby getFormModemaps, 575
- calledby getInverseEnvironment, 310
- calledby getModemapList, 244
- calledby getSignatureFromMode, 287
- calledby getSignature, 296
- calledby getSuccessEnvironment, 308
- calledby interactiveModemapForm, 160
- calledby isDomainConstructorForm, 339
- calledby isDomainForm, 338
- calledby isMacro, 267
- calledby makeFunctorArgumentParameters, 198
- calledby mkAlistOfExplicitCategoryOps, 158
- calledby mkEvalableCategoryForm, 140
- calledby mkNewModemapList, 246
- calledby mkOpVec, 203
- calledby mkRepititionAssoc, 149
- calledby mkUnion, 356
- calledby modemapPattern, 169
- calledby mustInstantiate, 274
- calledby optCallEval, 218
- calledby optCatch, 225
- calledby optCond, 227
- calledby optXLAMCond, 210
- calledby optimizeFunctionDef, 208
- calledby optimize, 209
- calledby orderPredTran, 163
- calledby orderPredicateItems, 162
- calledby outputComp, 337
- calledby parseHasRhs, 110
- calledby parseHas, 108
- calledby parseTranCheckForRecord, 517
- calledby postCollect,finish, 372
- calledby postTran, 360
- calledby replaceExitEtc, 327
- calledby setqMultiple, 331
- calledby spadCompileOrSetq, 151
- calledby stripOffArgumentConditions, 296
- calledby stripOffSubdomainConditions, 295
- calledby transIs1, 102
- calledby unknownTypeError, 233
- qcdr
 - calledby addArgumentConditions, 294
 - calledby addConstructorModemaps, 238
 - calledby addEltModemap, 245
 - calledby autoCoerceByModemap, 354
 - calledby canReturn, 306
 - calledby coerceByModemap, 354
 - calledby coerceExtraHard, 349
 - calledby compAdd, 256
 - calledby compCategoryItem, 271
 - calledby compCategory, 270
 - calledby compCons1, 279
 - calledby compDefWhereClause, 204
 - calledby compDefineFunctor1, 184
 - calledby compHasFormat, 303
 - calledby compJoin, 313
 - calledby compLambda, 315
 - calledby compSetq1, 330
 - calledby compWithMappingMode1, 586
 - calledby compileCases, 292
 - calledby compile, 145
 - calledby decodeScripts, 399
 - calledby doIt, 261
 - calledby eltModemapFilter, 577
 - calledby fixUpPredicate, 161
 - calledby flattenSignatureList, 159
 - calledby genDomainViewList, 201

- calledby genDomainView, 201
- calledby getFormModemaps, 576
- calledby getInverseEnvironment, 310
- calledby getModemapList, 244
- calledby getSignatureFromMode, 287
- calledby getSignature, 296
- calledby getSuccessEnvironment, 308
- calledby interactiveModemapForm, 160
- calledby isDomainConstructorForm, 339
- calledby isDomainForm, 338
- calledby isMacro, 267
- calledby makeFunctorArgumentParameters, 198
- calledby mkAListOfExplicitCategoryOps, 158
- calledby mkEvalableCategoryForm, 140
- calledby mkNewModemapList, 246
- calledby mkOpVec, 203
- calledby mkRepetitionAssoc, 149
- calledby mkUnion, 356
- calledby modemapPattern, 169
- calledby optCatch, 225
- calledby optCond, 227
- calledby optXLAMCond, 210
- calledby optimizeFunctionDef, 208
- calledby optimize, 209
- calledby orderPredTran, 163
- calledby orderPredicateItems, 162
- calledby outputComp, 337
- calledby parseHasRhs, 110
- calledby parseHas, 108
- calledby parseTranCheckForRecord, 517
- calledby postCollect,finish, 372
- calledby postTran, 360
- calledby replaceExitEtc, 327
- calledby setqMultiple, 331
- calledby spadCompileOrSetq, 151
- calledby stripOffArgumentConditions, 296
- calledby stripOffSubdomainConditions, 295
- calledby transIs1, 102
- qslessp
 - calledby addDomain, 232
 - calledby getUniqueModemap, 243
- qsminus, 221
 - defplist, 221
- quote, 319, 387
 - defplist, 319, 387
- quote-if-string, 450
 - calledby match-advance-string, 449
 - calledby unget-tokens, 452
 - calls escape-keywords, 450
 - calls pack, 450
 - calls strconc, 450
 - calls token-nonblank, 450
 - calls token-symbol, 450
 - calls token-type, 450
 - calls underscore, 450
 - uses \$boot, 450
 - uses \$spad, 450
 - defun, 450
- quotify
 - calledby compIf, 305
- rassoc
 - calledby checkPrenAList, 483
 - calledby modemapPattern, 169
- rbp
 - usedby PARSE-NudPart, 421
 - usedby PARSE-Operation, 421
- rdefiostream
 - calledby compileDocumentation, 174
 - calledby writeLib1, 176
- read-a-line, 601
 - calledby get-a-line, 608
 - calledby read-a-line, 601
 - calls Line-New-Line, 601
 - calls read-a-line, 601
 - calls subseq, 601
 - uses *eof*, 601
 - uses File-Closed, 601
 - defun, 601
- recompile-lib-file-if-necessary, 597
 - calledby compileSpadLispCmd, 596
 - calls compile-lib-file, 597
 - uses *lisp-bin-filetype*, 597
 - defun, 597
- Record, 269
 - defplist, 269
- recordAttributeDocumentation, 463
 - calledby PARSE-Category, 418
 - calls ifcdr, 463
 - calls opOf, 463

- calls pname, 463
- calls postTransform, 463
- calls recordDocumentation, 463
- calls upper-case-p, 463
- defun, 463
- RecordCategory, 281
 - defplist, 281
- recordcopy, 231
 - defplist, 231
- recordDocumentation, 464
 - calledby recordAttributeDocumentation, 463
 - calledby recordSignatureDocumentation, 463
 - calls collectComBlock, 464
 - calls recordHeaderDocumentation, 464
 - local def \$docList, 464
 - local def \$maxSignatureLineNumber, 464
 - defun, 464
- recordelt, 230
 - defplist, 230
- recordHeaderDocumentation, 464
 - calledby postDef, 379
 - calledby recordDocumentation, 464
 - calls assocright, 464
 - local def \$comblocklist, 464
 - local def \$headerDocumentation, 464
 - local ref \$comblocklist, 464
 - local ref \$headerDocumentation, 464
 - local ref \$maxSignatureLineNumber, 464
 - defun, 464
- recordSignatureDocumentation, 463
 - calledby PARSE-Category, 418
 - calls postTransform, 463
 - calls recordDocumentation, 463
 - defun, 463
- reduce, 320, 387
 - defplist, 320, 387
- reduce-stack, 462
 - usedby push-reduction, 462
 - uses \$stack, 462
 - defvar, 462
- reduce-stack-clear, 462
 - defmacro, 462
- reduction, 92
 - defstruct, 92
- reduction-value
 - calledby nth-stack, 524
 - calledby pop-stack-1, 522
 - calledby pop-stack-2, 523
 - calledby pop-stack-3, 523
 - calledby pop-stack-4, 523
- refvecp
 - calledby translabe1, 515
- remdup
 - calledby compDefineFunctor1, 183
 - calledby displayPreCompilationErrors, 516
 - calledby finalizeDocumentation, 466
 - calledby getSignature, 296
 - calledby hasSigInTargetCategory, 298
 - calledby mkAListOfExplicitCategoryOps, 158
 - calledby mkExplicitCategoryFunction, 273
 - calledby orderByDependency, 207
- removeBackslashes, 476
 - calledby checkGetParse, 475
 - calledby removeBackslashes, 476
 - calls charPosition, 476
 - calls length, 476
 - calls removeBackslashes, 476
 - calls strconc, 476
 - local ref \$charBack, 476
 - defun, 476
- removeEnv
 - calledby compApply, 563
 - calledby getSuccessEnvironment, 309
 - calledby setqSingle, 334
- removeSuperfluousMapping, 391
 - calledby postSignature, 390
 - defun, 391
- removeZeroOne
 - calledby compDefineCategory2, 142
 - calledby compDefineFunctor1, 184
 - calledby finalizeLisplib, 176
- remprop
 - calledby unloadOneConstructor, 173
- repeat, 322, 388
 - defplist, 322, 388
- replaceExitEsc
 - calledby coerceExit, 351
- replaceExitEtc, 327
 - calledby compDefineCapsuleFunction, 288

- calledby compSeq1, 326
- calledby replaceExitEtc, 327
- calls convertOrCroak, 327
- calls intersectionEnvironment, 327
- calls qcar, 327
- calls qcdr, 327
- calls replaceExitEtc, 327
- calls rplac, 327
- local def \$finalEnv, 327
- local ref \$finalEnv, 327
- defun, 327
- replaceFile
 - calledby compileDocumentation, 174
 - calledby lisplibDoRename, 174
- replaceVars, 161
 - calledby interactiveModemapForm, 160
 - calls msubst, 161
 - defun, 161
- reportOnFunctorCompilation, 197
 - calledby compDefineFunctor1, 184
 - calls addStats, 197
 - calls displayMissingFunctions, 197
 - calls displaySemanticErrors, 197
 - calls displayWarnings, 197
 - calls normalizeStatAndStringify, 197
 - calls sayBrightly, 197
 - uses \$functionStats, 197
 - uses \$functorStats, 197
 - uses \$op, 197
 - uses \$semanticErrorStack, 197
 - uses \$warningStack, 197
 - defun, 197
- resolve, 356
 - calledby coerceExit, 351
 - calledby compApplication, 574
 - calledby compApply, 563
 - calledby compCategory, 270
 - calledby compCoerce1, 353
 - calledby compConstructorCategory, 282
 - calledby compDefineCapsuleFunction, 288
 - calledby compIf, 305
 - calledby compReturn, 325
 - calledby compString, 339
 - calledby convert, 568
 - calledby modifyModeStack, 593
 - calls mkUnion, 356
 - calls modeEqual, 356
 - calls nequal, 356
 - local ref \$EmptyMode, 356
 - local ref \$NoValueMode, 356
 - local ref \$String, 356
 - defun, 356
- return, 127, 325
 - defplist, 127, 325
- rpackfile
 - calledby compDefineLisplib, 171
 - calledby compileDocumentation, 174
- rplac
 - calledby coerce, 346
 - calledby getAbbreviation, 285
 - calledby optCall, 214
 - calledby optCatch, 225
 - calledby optCond, 227
 - calledby optSpecialCall, 216
 - calledby optXLAMCond, 210
 - calledby optimizeFunctionDef, 208
 - calledby optimize, 209
 - calledby replaceExitEtc, 327
- rplaca
 - calledby NRTgetLocalIndex, 192
 - calledby NRTputInTail, 155
 - calledby doItIf, 265
 - calledby optPackageCall, 215
 - calledby optSpecialCall, 216
- rplacd
 - calledby doItIf, 265
 - calledby optCond, 227
 - calledby optPackageCall, 215
- rplacw
 - calledby optSpecialCall, 216
- rshut
 - calledby compDefineLisplib, 171
 - calledby compileDocumentation, 174
- rwrite
 - calledby compilerDoitWithScreenedLisplib, 358
- rwrite128
 - calledby lisplibWrite, 182
- rwriteLispForm, 170
 - calledby evalAndRwriteLispForm, 169
 - local ref \$libFile, 170
 - local ref \$lisplib, 170

- defun, 170
- s-process, 550
 - calledby spad, 549
 - calls compTopLevel, 551
 - calls curstrm, 550
 - calls def-process, 551
 - calls def-rename, 550
 - calls displayPreCompilationErrors, 550
 - calls displaySemanticErrors, 551
 - calls get-internal-run-time, 551
 - calls new2OldLisp, 550
 - calls parseTransform, 550
 - calls postTransform, 550
 - calls prettyprint, 550
 - calls processInteractive[5], 551
 - calls terpri, 551
 - uses \$DomainFrame, 551
 - uses \$EmptyEnvironment, 551
 - uses \$EmptyMode, 551
 - uses \$Index, 551
 - uses \$InteractiveFrame, 551
 - uses \$LocalFrame, 551
 - uses \$PolyMode, 551
 - uses \$PrintOnly, 551
 - uses \$TranslateOnly, 551
 - uses \$Translation, 551
 - uses \$VariableCount, 551
 - uses \$compUniquelyIfTrue, 551
 - uses \$currentFunction, 551
 - uses \$currentLine, 551
 - uses \$exitModeStack, 551
 - uses \$exitMode, 551
 - uses \$e, 551
 - uses \$form, 551
 - uses \$genFVar, 551
 - uses \$genSDVar, 551
 - uses \$insideCapsuleFunctionIfTrue, 551
 - uses \$insideCategoryIfTrue, 551
 - uses \$insideCoerceInteractiveHardIfTrue, 551
 - uses \$insideExpressionIfTrue, 551
 - uses \$insideFunctorIfTrue, 551
 - uses \$insideWhereIfTrue, 551
 - uses \$leaveLevelStack, 551
 - uses \$leaveMode, 551
 - uses \$macroassoc, 551
 - uses \$newspad, 551
 - uses \$postStack, 551
 - uses \$previousTime, 551
 - uses \$returnMode, 551
 - uses \$semanticErrorStack, 551
 - uses \$stop-level, 551
 - uses \$stopOp, 551
 - uses \$warningStack, 551
 - uses curoutstream, 551
 - defun, 550
- say
 - calledby canReturn, 306
 - calledby compOrCroak1, 557
 - calledby getSignature, 296
 - calledby modifyModeStack, 593
 - calledby optimize, 209
 - calledby orderByDependency, 207
 - calledby putDomainsInScope, 235
- sayBrightly
 - calledby NRTgetLookupFunction, 191
 - calledby checkAndDeclare, 298
 - calledby checkDocError, 479
 - calledby compDefineCapsuleFunction, 288
 - calledby compDefineFunctor1, 183
 - calledby compMacro, 317
 - calledby compilerDoit, 538
 - calledby compile, 145
 - calledby displayMissingFunctions, 197
 - calledby displayPreCompilationErrors, 516
 - calledby doIt, 261
 - calledby reportOnFunctorCompilation, 197
 - calledby spadCompileOrSetq, 152
 - calledby transDocList, 468
 - calledby transformAndRecheckComments, 470
- saybrightly1
 - calledby checkDocError, 478
- sayBrightlyI
 - calledby optimizeFunctionDef, 208
- sayBrightlyNT
 - calledby NRTgetLookupFunction, 191
- sayKeyedMsg
 - calledby finalizeDocumentation, 466
- sayKeyedMsg[5]
 - called by compileSpad2Cmd, 536

- called by compileSpadLispCmd, 596
- sayMath
 - calledby displayPreCompilationErrors, 516
- sayMSG
 - calledby compDefineLisplib, 171
 - calledby finalizeDocumentation, 466
 - calledby finalizeLisplib, 176
- ScanOrPairVec[5]
 - called by hasFormalMapVariable, 592
- Scripts, 388
 - defplist, 388
- segment, 127, 128
 - defplist, 127, 128
- selectOptionLC[5]
 - called by compileSpad2Cmd, 536
 - called by compileSpadLispCmd, 596
 - called by compiler, 533
- seq, 218, 326
 - defplist, 218, 326
- setDefOp, 394
 - calledby parseDEF, 101
 - calledby postcheck, 363
 - uses \$defOp, 394
 - uses \$topOp, 394
 - defun, 394
- setdifference
 - calledby orderPredTran, 163
- setelt
 - calledby checkAddPeriod, 483
 - calledby modifyModeStack, 593
- seteltModemapFilter, 577
 - calls isConstantId, 577
 - calls stackMessage, 577
 - defun, 577
- setq, 329
 - defplist, 329
- setqMultiple, 330
 - calledby compSetq1, 330
 - calls addBinding, 331
 - calls compSetq1, 331
 - calls convert, 331
 - calls genSomeVariable, 331
 - calls genVariable, 331
 - calls length, 331
 - calls mkprogn, 331
 - calls nreverse0, 330
 - calls put, 331
 - calls qcar, 331
 - calls qcdr, 331
 - calls setqMultipleExplicit, 331
 - calls stackMessage, 331
 - local ref \$EmptyMode, 331
 - local ref \$NoValueMode, 331
 - local ref \$noEnv, 331
 - defun, 330
- setqMultipleExplicit, 333
 - calledby setqMultiple, 331
 - calls compSetq1, 333
 - calls genVariable, 333
 - calls last, 333
 - calls nequal, 333
 - calls stackMessage, 333
 - local ref \$EmptyMode, 333
 - local ref \$NoValueMode, 333
 - defun, 333
- setqSetelt, 334
 - calledby compSetq1, 330
 - calls comp, 334
 - defun, 334
- setqSingle, 334
 - calledby compSetq1, 329
 - calls NRTassocIndex, 334
 - calls addBinding[5], 334
 - calls assignError, 334
 - calls augModemapsFromDomain1, 334
 - calls comp, 334
 - calls consProplistOf, 334
 - calls convert, 334
 - calls getProplist[5], 334
 - calls getmode, 334
 - calls get, 334
 - calls identp[5], 334
 - calls isDomainForm, 334
 - calls isDomainInScope, 334
 - calls maxSuperType, 334
 - calls nequal, 334
 - calls outputComp, 334
 - calls profileRecord, 334
 - calls removeEnv, 334
 - calls stackWarning, 334
 - uses \$EmptyMode, 335
 - uses \$NoValueMode, 335

- uses \$QuickLet, 335
 - uses \$form, 335
 - uses \$insideSetqSingleIfTrue, 335
 - uses \$profileCompiler, 335
 - defun, 334
- setrecordelt, 230
 - defplist, 230
- shut
 - calledby whoOwns, 480
- shut[5]
 - called by spad, 549
- Signature, 390
 - defplist, 390
- signatureTran, 163
 - calledby orderPredicateItems, 162
 - calledby signatureTran, 163
 - calls isCategoryForm, 163
 - calls signatureTran, 163
 - local ref \$e, 163
 - defun, 163
- simpBool
 - calledby compDefineFunctor1, 184
- skip-blanks, 448
 - calledby match-string, 448
 - calls advance-char, 448
 - calls current-char, 448
 - calls token-lookahead-type, 448
 - defun, 448
- skip-ifblock, 86
 - calledby skip-ifblock, 86
 - calls initial-substring, 86
 - calls preparseReadLine1, 86
 - calls skip-ifblock, 86
 - calls storeblanks, 86
 - calls string2BootTree, 86
 - defun, 86
- skip-to-endif, 528
 - calledby preparseReadLine, 85
 - calledby skip-to-endif, 528
 - calls initial-substring, 528
 - calls preparseReadLine1, 528
 - calls preparseReadLine, 528
 - calls skip-to-endif, 528
 - defun, 528
- SourceLevelSubsume
 - calledby getSignature, 296
- spad, 549
 - calls PARSE-NewExpr, 549
 - calls addBinding[5], 549
 - calls init-boot/spad-reader[5], 549
 - calls initialize-preparse, 549
 - calls ioclear, 549
 - calls makeInitialModemapFrame[5], 549
 - calls pop-stack-1, 549
 - calls preparse, 549
 - calls s-process, 549
 - calls shut[5], 549
 - uses *comp370-apply*, 549
 - uses *eof*, 549
 - uses *fileactq-apply*, 549
 - uses /editfile, 549
 - uses \$InitialDomainsInScope, 549
 - uses \$InteractiveFrame, 549
 - uses \$InteractiveMode, 549
 - uses \$boot, 549
 - uses \$noSubsumption, 549
 - uses \$spad, 549
 - uses boot-line-stack, 549
 - uses curoutstream, 549
 - uses echo-meta, 549
 - uses file-closed, 549
 - uses line, 549
 - uses optionlist, 549
 - catches, 549
 - defun, 549
- spad-fixed-arg, 598
 - defun, 598
- spad2AsTranslatorAutoloadOnceTrigger
 - calledby compileSpad2Cmd, 536
- spad[5]
 - called by /rf-1, 540
- spadcall, 223
 - defplist, 223
- spadCompileOrSetq, 151
 - calledby compile, 146
 - calls LAM,EVALANDFILEACTQ, 152
 - calls bright, 152
 - calls compileConstructor, 152
 - calls comp, 152
 - calls contained, 152
 - calls mkq, 152
 - calls qcar, 151

- calls qcdr, 151
- calls sayBrightly, 152
- local ref \$insideCapsuleFunctionIfTrue, 152
- defun, 151
- SpadInterpretStream[5]
 - called by ncINTERPFILE, 595
- spadPrompt
 - calledby compileSpad2Cmd, 536
 - calledby compileSpadLispCmd, 596
- spadreduce
 - calledby floatexpid, 459
- spadSysChoose
 - calledby checkRecordHash, 473
- splitEncodedFunctionName, 149
 - calledby compile, 145
 - calls stringimage, 149
 - calls strpos, 149
 - defun, 149
- stack, 88
 - defstruct, 88
- stack-/empty, 89
 - uses \$stack, 89
 - defmacro, 89
- stack-clear, 89
 - uses \$stack, 89
 - defun, 89
- stack-load, 89
 - uses \$stack, 89
 - defun, 89
- stack-pop, 90
 - calledby Pop-Reduction, 524
 - uses \$stack, 90
 - defun, 90
- stack-push, 89
 - calledby pop-stack-2, 523
 - calledby pop-stack-3, 523
 - calledby pop-stack-4, 523
 - calledby push-reduction, 462
 - uses \$stack, 89
 - defun, 89
- stack-size
 - calledby star, 461
- stack-store
 - calledby nth-stack, 524
- stackAndThrow
 - calledby compDefine1, 283
 - calledby compInternalFunction, 287
 - calledby compLambda, 315
 - calledby compWithMappingMode1, 586
 - calledby getSignatureFromMode, 287
- stackMessage
 - calledby assignError, 336
 - calledby augModemapsFromDomain1, 237
 - calledby autoCoerceByModemap, 354
 - calledby coerce, 346
 - calledby compElt, 300
 - calledby compMapCond", 241
 - calledby compMapCond', 241
 - calledby compRepeatOrCollect, 323
 - calledby compSymbol, 569
 - calledby eltModemapFilter, 577
 - calledby getFormModemaps, 576
 - calledby getOperationAlist, 250
 - calledby seteltModemapFilter, 577
 - calledby setqMultipleExplicit, 333
 - calledby setqMultiple, 331
- stackMessageIfNone
 - calledby compExit, 302
 - calledby compForm, 571
- stackSemanticError
 - calledby compColonInside, 565
 - calledby compJoin, 313
 - calledby compOrCroak1, 557
 - calledby compPretend, 319
 - calledby compReturn, 325
 - calledby compSubDomain1, 341
 - calledby constructMacro, 151
 - calledby doIt, 261
 - calledby getArgumentModeOrMoan, 156
 - calledby getSignature, 296
 - calledby getTargetFromRhs, 135
 - calledby unknownTypeError, 233
- stackWarning
 - calledby compColonInside, 565
 - calledby compElt, 300
 - calledby compPretend, 319
 - calledby doIt, 261
 - calledby getUniqueModemap, 243
 - calledby hasSigInTargetCategory, 298
 - calledby setqSingle, 334
- star, 461

- calledby PARSE-Application, 426
- calledby PARSE-Category, 418
- calledby PARSE-CommandTail, 415
- calledby PARSE-Expr, 420
- calledby PARSE-Import, 419
- calledby PARSE-IteratorTail, 441
- calledby PARSE-Loop, 446
- calledby PARSE-Option, 416
- calledby PARSE-ScriptItem, 435
- calledby PARSE-Sexpr1, 437
- calledby PARSE-SpecialCommand, 413
- calledby PARSE-Statement, 416
- calledby PARSE-TokenCommandTail, 414
- calledby PARSE-TokenList, 414
- calls pop-stack-1, 461
- calls push-reduction, 461
- calls stack-size, 461
- defmacro, 461
- step
 - calledby floatexpid, 459
- storeblanks, 607
 - calledby preparseReadLine, 85
 - calledby skip-ifblock, 86
 - defun, 607
- strconc
 - calledby checkAddBackSlashes, 511
 - calledby checkAddSpaceSegments, 505
 - calledby checkComments, 481
 - calledby checkIndentedLines, 502
 - calledby checkTransformFirsts, 488
 - calledby compApplication, 574
 - calledby compDefine1, 283
 - calledby compDefineFunctor1, 183
 - calledby compileSpad2Cmd, 536
 - calledby compile, 145
 - calledby decodeScripts, 399
 - calledby finalizeDocumentation, 466
 - calledby getCaps, 150
 - calledby mkCategoryPackage, 139
 - calledby preparseReadLine1, 87
 - calledby quote-if-string, 450
 - calledby removeBackslashes, 476
 - calledby unget-tokens, 452
 - calledby whoOwns, 480
- String, 339
 - defplist, 339
- string-not-greaterp
 - calledby initial-substring-p, 450
- string2BootTree
 - calledby preparseReadLine, 85
 - calledby skip-ifblock, 86
- string2id-n
 - calledby infixtok, 527
- stringimage
 - calledby encodeFunctionName, 148
 - calledby encodeItem, 150
 - calledby finalizeDocumentation, 466
 - calledby getCaps, 150
 - calledby splitEncodedFunctionName, 149
 - calledby substituteCategoryArguments, 237
- stringPrefix?
 - calledby checkGetArgs, 504
 - calledby comp3, 560
- stripOffArgumentConditions, 296
 - calledby compDefineCapsuleFunction, 288
 - calls msubst, 296
 - calls qcar, 296
 - calls qcdr, 296
 - local def \$argumentConditionList, 296
 - local ref \$argumentConditionList, 296
 - defun, 296
- stripOffSubdomainConditions, 295
 - calledby compDefineCapsuleFunction, 288
 - calls assoc, 295
 - calls mkpf, 295
 - calls qcar, 295
 - calls qcdr, 295
 - local def \$argumentConditionList, 295
 - local ref \$argumentConditionList, 295
 - defun, 295
- stripUnionTags
 - calledby augModemapsFromDomain, 236
- strpos
 - calledby splitEncodedFunctionName, 149
- strposl[5]
 - called by preparse1, 81
- SubDomain, 340
 - defplist, 340
- sublis
 - calledby NRTgetLookupFunction, 191
 - calledby applyMapping, 562

- calledby augLisplibModemapsFromCategory, 157
- calledby coerceable, 350
- calledby compApplyModemap, 239
- calledby compDefineCategory2, 142
- calledby compDefineFunctor1, 184
- calledby compForm2, 579
- calledby doIt, 261
- calledby formal2Pattern, 194
- calledby getModemap, 239
- calledby mkOpVec, 203
- calledby substituteCategoryArguments, 237
- calledby substituteIntoFunctorModemap, 583
- sublis
 - calledby compHasFormat, 303
 - calledby finalizeDocumentation, 466
 - calledby mkCategoryPackage, 139
- sublname, 212
 - calledby optimize, 209
 - calls bpiname, 212
 - calls compiled-function-p, 212
 - calls identp, 212
 - calls mbpip, 212
 - defun, 212
- subseq
 - calledby match-string, 448
 - calledby read-a-line, 601
- SubsetCategory, 342
 - defplist, 342
- substituteCategoryArguments, 237
 - calledby augModemapsFromDomain1, 237
 - calls internl, 237
 - calls msubst, 237
 - calls stringimage, 237
 - calls sublis, 237
 - defun, 237
- substituteIntoFunctorModemap, 583
 - calledby compFocompFormWithModemap, 581
 - calls compOrCroak, 583
 - calls eqsubstlist, 583
 - calls keyedSystemError, 583
 - calls nequal, 583
 - calls sublis, 583
 - defun, 583
- substNames, 251
 - calledby evalAndSub, 249
 - calledby genDomainOps, 202
 - calls eqsubstlist, 251
 - calls isCategoryPackageName, 251
 - calls nreverse0, 251
 - calls substq, 251
 - uses \$FormalMapVariableList, 251
 - defun, 251
- substq
 - calledby substNames, 251
- substring?
 - calledby checkBeginEnd, 484
 - calledby checkExtract, 507
- substVars, 168
 - calledby interactiveModemapForm, 160
 - calls contained, 168
 - calls msubst, 168
 - calls nsubst, 168
 - local ref \$FormalMapVariableList, 168
 - defun, 168
- suffix
 - calledby addclose, 524
- systemCommand[5]
 - called by PARSE-CommandTail, 415
 - called by PARSE-TokenCommandTail, 414
- systemError
 - calledby checkTrim, 506
 - calledby compReduce1, 320
 - calledby errhuh, 403
 - calledby getOperationAlist, 250
- systemErrorHere
 - calledby addArgumentConditions, 294
 - calledby addEltModemap, 245
 - calledby canReturn, 306
 - calledby compCategory, 270
 - calledby compColon, 275
 - calledby getSlotFromCategoryForm, 179
 - calledby getSlotFromFunctor, 181
 - calledby optCall, 214
 - calledby postIn, 383
 - calledby postin, 382
- take
 - calledby compColon, 275
 - calledby compDefineCategory2, 142

- calledby compForm2, 579
- calledby compHasFormat, 303
- calledby compWithMappingMode1, 586
- calledby drop, 525
- calledby getSignatureFromMode, 287
- calledby getSlotFromCategoryForm, 179
- terminateSystemCommand[5]
 - called by compileSpad2Cmd, 536
 - called by compileSpadLispCmd, 596
- terpri
 - calledby s-process, 551
- throwKeyedMsg
 - calledby compileSpad2Cmd, 536
 - calledby compileSpadLispCmd, 596
 - calledby compiler, 533
 - calledby loadLibIfNecessary, 111
- throwkeyedmsg
 - calledby postMDef, 385
- throws
 - compForm3, 580
- tmptok, 410
 - usedby PARSE-Operation, 421
 - defvar, 410
- tok, 410
 - usedby PARSE-GlyphTok, 438
 - usedby PARSE-NBGlyphTok, 437
 - defvar, 410
- token, 90
 - defstruct, 90
- token-install, 92
 - uses \$token, 92
 - defun, 92
- token-lookahead-type, 449
 - calledby skip-blanks, 448
 - uses Escape-Character, 449
 - defun, 449
- token-nonblank
 - calledby PARSE-FloatBasePart, 431
 - calledby quote-if-string, 450
 - calledby unget-tokens, 453
- token-print, 92
 - uses \$token, 92
 - defun, 92
- token-symbol
 - calledby PARSE-SpecialKeyWord, 412
 - calledby match-token, 454
 - calledby parse-argument-designator, 520
 - calledby parse-identifier, 519
 - calledby parse-keyword, 520
 - calledby parse-number, 520
 - calledby parse-spadstring, 518
 - calledby parse-string, 519
 - calledby quote-if-string, 450
- token-type
 - calledby match-token, 454
 - calledby quote-if-string, 450
- TPDHERE
 - See LocalAlgebra for an example call, 342
 - The use of and in spadreduce is undefined. rewrite this to loop, 459
 - test with BASTYPE, 134
- transDoc, 469
 - calledby transDocList, 469
 - calls checkDocError1, 469
 - calls checkExtract, 469
 - calls checkTrim, 469
 - calls nreverse, 469
 - calls transformAndRecheckComments, 469
 - local def \$arg1, 469
 - local def \$attribute?, 469
 - local def \$x, 469
 - local ref \$attribute?, 469
 - local ref \$x, 469
 - defun, 469
- transDocList, 468
 - calledby finalizeDocumentation, 466
 - calls checkDocError1, 469
 - calls checkDocError, 469
 - calls sayBrightly, 468
 - calls transDoc, 469
 - local ref \$constructorName, 469
 - defun, 468
- transformAndRecheckComments, 470
 - calledby transDoc, 469
 - calls sayBrightly, 470
 - local def \$checkingXmptex?, 470
 - local def \$exposeFlagHeading, 471
 - local def \$name, 470
 - local def \$origin, 471
 - local def \$recheckingFlag, 471
 - local def \$x, 470

- local ref \$exposeFlagHeading, 470
 - defun, 470
- transformOperationAlist, 179
 - calledby getCategoryOpsAndAtts, 178
 - calledby getFunctorOpsAndAtts, 181
 - calls assoc, 180
 - calls insertAlist, 180
 - calls keyedSystemError, 180
 - calls lassq, 180
 - calls member, 180
 - local ref \$functionLocations, 180
 - defun, 179
- transImplementation, 567
 - calledby compAtomWithModemap, 567
 - calls genDeltaEntry, 567
 - defun, 567
- transIs, 102
 - calledby parseIsnt, 120
 - calledby parseIs, 119
 - calledby parseLET, 122
 - calledby parseLhs, 102
 - calledby transIs1, 102
 - calls isListConstructor, 102
 - calls transIs1, 102
 - defun, 102
- transIs1, 102
 - calledby transIs1, 102
 - calledby transIs, 102
 - calls nreverse0, 102
 - calls qcar, 102
 - calls qcdr, 102
 - calls transIs1, 102
 - calls transIs, 102
 - defun, 102
- translabel, 515
 - calledby PARSE-Data, 436
 - calls translabel1, 515
 - defun, 515
- translabel1, 515
 - calledby translabel1, 515
 - calledby translabel, 515
 - calls lassoc, 515
 - calls maxindex, 515
 - calls refvecp, 515
 - calls translabel1, 515
 - defun, 515
- transSeq
 - calledby parseSeq, 128
- trimString
 - calledby checkGetArgs, 504
- TruthP, 249
 - calledby mergeModemap, 248
 - calledby optCond, 227
 - calls qcar, 249
 - defun, 249
- try-get-token, 455
 - calledby advance-token, 456
 - calledby current-token, 455
 - calledby next-token, 456
 - calls get-token, 455
 - uses valid-tokens, 455
 - defun, 455
- tuple2List, 521
 - calledby postCollect,finish, 372
 - calledby postConstruct, 377
 - calledby postInSeq, 382
 - calledby tuple2List, 521
 - calls postTranSegment, 521
 - calls postTran, 522
 - calls tuple2List, 521
 - uses \$InteractiveMode, 522
 - uses \$boot, 522
 - defun, 521
- TupleCollect, 392
 - defplist, 392
- unabbrevAndLoad
 - calledby parseHasRhs, 110
 - calledby parseHas, 108
- unAbbreviateKeyword[5]
 - called by PARSE-SpecialKeyWord, 412
- uncons, 330
 - calledby uncons, 330
 - calls uncons, 330
 - defun, 330
- Undef
 - usedby mkOpVec, 203
- underscore, 452
 - calledby quote-if-string, 450
 - calls vector-push, 452
 - defun, 452
- unembed

- calledby compilerDoitWithScreenedLisplib, 113
 - 358
- unErrorRef
 - calledby addModemap1, 254
- unget-tokens, 452
 - calledby match-string, 448
 - calls line-current-segment, 452
 - calls line-new-line, 453
 - calls line-number, 453
 - calls quote-if-string, 452
 - calls strconc, 452
 - calls token-nonblank, 453
 - uses valid-tokens, 453
 - defun, 452
- Union, 269
 - defplist, 269
- union
 - calledby compDefWhereClause, 204
 - calledby compJoin, 313
 - calledby makeFunctorArgumentParameters, 198
 - calledby mkAlistOfExplicitCategoryOps, 158
 - calledby mkExplicitCategoryFunction, 273
 - calledby mkUnion, 356
- UnionCategory, 281
 - defplist, 281
- unionq
 - calledby freelist, 594
 - calledby orderPredTran, 163
- unknownTypeError, 233
 - calledby addDomain, 232
 - calledby compColon, 275
 - calls qcar, 233
 - calls stackSemanticError, 233
 - defun, 233
- unloadOneConstructor, 173
 - calledby compDefineLisplib, 171
 - calls mkAutoLoad, 173
 - calls remprop, 173
 - defun, 173
- unTuple, 403
 - calledby postMapping, 384
 - calledby postTran, 360
 - calledby postType, 369
 - defun, 403
- calledby compDefineLisplib, 171
 - calledby isFunctor, 233
 - calledby loadLibIfNecessary, 111
 - calls addModemap, 113
 - calls getdatabase, 113
 - calls put, 113
 - local def \$CategoryFrame, 113
 - local ref \$CategoryFrame, 113
 - defun, 113
- updateCategoryFrameForConstructor, 112
 - calledby compDefineLisplib, 171
 - calledby isFunctor, 233
 - calledby loadLibIfNecessary, 111
 - calls addModemap, 112
 - calls convertOpAlist2compilerInfo, 112
 - calls getdatabase, 112
 - calls put, 112
 - local def \$CategoryFrame, 112
 - local ref \$CategoryFrame, 112
 - defun, 112
- updateSourceFiles[5]
 - called by compileSpad2Cmd, 536
- upper-case-p
 - calledby recordAttributeDocumentation, 463
- userError
 - calledby compDefWhereClause, 204
 - calledby compOrCroak1, 557
 - calledby compReturn, 325
 - calledby compile, 145
 - calledby convertOrCroak, 328
 - calledby doItIf, 265
 - calledby orderByDependency, 207
- valid-tokens, 91
 - usedby advance-token, 456
 - usedby current-token, 455
 - usedby next-token, 456
 - usedby try-get-token, 455
 - usedby unget-tokens, 453
 - uses \$token, 91
 - defvar, 91
- vcons, 129
 - defplist, 129
- Vector

- calledby optCallEval, 218
- vector, 343
 - defplist, 343
- vector-push
 - calledby underscore, 452
- VectorCategory, 281
 - defplist, 281
- vmisp::optionlist
 - usedby print-defun, 553
- where, 129, 344, 393
 - defplist, 129, 344, 393
- whoOwns, 480
 - calledby checkDocMessage, 479
 - calls awk, 480
 - calls getdatabase, 480
 - calls shut, 480
 - calls strconc, 480
 - local ref \$exposeFlag, 480
 - defun, 480
- with, 394
 - defplist, 394
- wrapDomainSub, 274
 - calledby compJoin, 313
 - calledby mkExplicitCategoryFunction, 273
 - defun, 274
- wrapSEQExit
 - calledby makeSimplePredicateOrNil, 518
- writeLib1, 176
 - calledby initializeLisplib, 175
 - calls rdefiostream, 176
 - defun, 176
- XTokenReader, 457
 - calledby get-token, 457
 - usedby get-token, 457
 - defvar, 457