

\$SPAD/src/input hilbert.as

The Axiom Team

July 30, 2014

**Abstract**

## Contents

<b>1</b>	<b>Hilbert input tests</b>	<b>3</b>
<b>2</b>	<b>License</b>	<b>3</b>

# 1 Hilbert input tests

— input —

```
)compile hilbert.as
mon1 := monom(4,0,0,0)
mon2:= monom(3,3,0,0)
mon3 := monom(3,2,1,0)
mon4 := monom(3,1,2,0)
mon5 := monom(0,2,0,1)
mon6 := monom(0,1,0,5)
l := [mon1, mon2, mon3, mon4, mon5, mon6]
Hilbert l
idA := varMonomsPower(6,5);
#idA
Hilbert idA
idB := varMonomsPower(6,6);
#idB
Hilbert idB
idC := varMonomsPower(12,3);
#idC
Hilbert idC
idD:=[monom(2,0,0,0),monom(1,1,0,0),monom(1,0,1,0),monom(1,0,0,1),_
      monom(0,3,0,0),monom(0,2,1,0)]^4;
#idD
Hilbert idD
```

—————

# 2 License

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

— \* —

```
\getchunk{input}

#include "axiom.as"
#pile

-- This file computes hilbert functions for monomial ideals
-- ref: "On the Computation of Hilbert-Poincare Series",
-- Bigatti, Caboara, Robbiano,
-- AAEC vol 2 #1 (1991) pp 21-33

macro
  Monom == Monomial
  L == List
  SI == SingleInteger
  B == Boolean
  POLY == SparseUnivariatePolynomial Integer
  Array == Vector

import from NonNegativeInteger
import from SingleInteger
import from Segment SI
import from Integer

Monomial : OrderedSet with
  totalDegree: % -> SI
  divides?: (% , %) -> B
  homogLess: (% , %) -> B
  quo: (% , %) -> %
  quo: (% , SI) -> %
  *: (% , %) -> %
  varMonom: (i:SI,n:SI, deg:SI) -> %
```

```

variables: L % -> L SI
apply: (% , SI) -> SI
#: % -> SI
monom: Tuple SI -> %
== Array(Integer) add
Rep ==> Array(Integer)
import from Rep

monom(t:Tuple SI):% == per [ t ]

totalDegree(m:%):SI ==
sum:SI := 0
for e in rep m repeat sum := sum + e
sum

divides?(m1:%, m2:%):B ==
for e1 in rep m1 for e2 in rep m2 repeat
  if e1 > e2 then return false
true

(m1:%) < (m2:%):B ==
for e1 in rep m1 for e2 in rep m2 repeat
  if e1 < e2 then return true
  if e1 > e2 then return false
false

--      (m1:%) > (m2:%):B == m2 < m1

homogLess(m1:%, m2:%):B ==
(d1:=totalDegree(m1)) < (d2:=totalDegree(m2)) => true
d1 > d2 => false
( m1 < m2)

(m:%) quo (v:SI):% == --remove vth variable
--      per [(if i=v then 0 else (rep m).i) for i in 1..#rep m]
      m2:= copy rep m
      m2.v := 0
      per m2

(m1:%) quo (m2:%):% ==
      per [(max(a1-a2,0) for a1 in rep m1 for a2 in rep m2)]

(m1:%) * (m2:%):% == per [(a1+a2 for a1 in rep m1 for a2 in rep m2)]

varMonom(i:SI,n:SI, deg:SI):% ==
--      per [(if j=i then deg else 0$SI) for j in 1..n]
      m:Rep := new(n, 0)
      m.i := deg
      per m

```

```

        variables(I:L %) :L SI ==
empty? I => nil
n:SI:=# rep first I
ans : L SI := nil
        v:SI:=0
        while (v:=v+1)<=n repeat
-- for v in 1..n repeat
    for m in I repeat
        (rep m).v ~= 0 =>
        ans := cons(v, ans)
    break
ans

HilbertFunctionPackage: with
        Hilbert: L Monom -> POLY
        adjoin: (Monom, L Monom) -> L Monom
    == add

        adjoin(m:Monom, lm:L Monom):L Monom ==
empty?(lm) => cons(m, nil)
ris1:L Monom:= empty()
ris2:L Monom:= empty()
while not empty? lm repeat
    m1:Monom := first lm
    lm := rest lm
    if m <= m1 then
        if not divides?(m,m1) then (ris1 := cons(m1, ris1))
        iterate
    ris2 := cons(m1, ris2)
    if divides?(m1, m) then
        return concat!(reverse!(ris1), concat!(reverse! ris2, lm))
concat!(reverse!(ris1), cons(m, reverse! ris2))

        reduce(lm:L Monom):L Monom ==
lm := sortHomogRemDup(lm)
empty? lm => lm
ris :L Monom := nil
risd:L Monom := list first lm
d := totalDegree first lm
for m in rest lm repeat
    if totalDegree(m)=d then risd := cons(m, risd)
    else
        ris := mergeDiv(ris, risd)
        d := totalDegree m
        risd := [m]
mergeDiv(ris, risd)

        mergeDiv( small:L Monom, big:L Monom): L Monom ==
ans : L Monom := small

```

```

for m in big repeat
  if not contained?(m,small) then ans := cons(m, ans)
ans

  contained?(m:Monom, id: L Monom) : B ==
for mm in id repeat
  divides?(mm, m) => return true
false

  (I:L Monom) quo (m:Monom):L Monom ==
reduce [mm quo m for mm in I]

  sort(I:L Monom, v:SI):L Monom ==
sort((a:Monom,b:Monom):B+-(a.v < b.v), I)

  sortHomogRemDup(l:L Monom):L Monom ==
l:=sort(homogLess, l)
empty? l => l
ans:L Monom := list first l
for m in rest l repeat
  if m ~= first(ans) then ans:=cons(m, ans)
reverse! ans

  decompose(I:L Monom, v:SI):Record(size:SI, ideals:L L Monom, degs:L SI) ==
I := sort(I, v)
idlist: L L Monom := nil
deglist : L SI := nil
size : SI := 0
J: L Monom := nil
while not empty? I repeat
  d := first(I).v
  tj : L Monom := nil
  local m:Monom
  while not empty? I and d=(m:=first I).v repeat
    tj := cons(m quo v, tj)
    I := rest I
  J := mergeDiv(tj, J)
  idlist := cons(J, idlist)
  deglist := cons(d, deglist)
  size := size + ((#J)::Integer::SI)
[size, idlist, deglist]

  var(n:SI) : SparseUnivariatePolynomial Integer ==
    monomial(1$Integer, n::Integer::NonNegativeInteger)

  Hilbert(I:L Monom):POLY ==
empty? I => 1 -- no non-zero generators = 0 ideal
empty? rest I => var(0) - var(totalDegree first I)

```

```

lvar :L SI := variables I
import from Record(size:SI, ideals:L L Monom, degs:L SI)
Jbest := decompose(I, first lvar)
for v in rest lvar while (#I)::Integer::SI < Jbest.size repeat
  JJ := decompose(I, v)
  JJ.size < Jbest.size => Jbest := JJ
import from L L Monom
import from L SI
Jold:List Monom := first(Jbest.ideals)
dold:SI := first(Jbest.degs)
f: SparseUnivariatePolynomial Integer:=var(dold)*Hilbert(Jold)
for J:List Monom in rest Jbest.ideals for d:SI in rest Jbest.degs repeat
  f := f + (var(d) - var(dold)) * Hilbert(J)
  dold := d
var(0) - var(dold) + f

MonomialIdealPackage: with
  varMonomsPower: (SI, SI) -> L Monom
  *: (L Monom, L Monom) -> L Monom
  ^: (L Monom, SI) -> L Monom
  == add

  varMonoms(n:SI):L Monom ==
-- [varMonom(i,n,1) for i in 1..n]
  i:SI:=0
  [varMonom(i,n,1) while {free i; (i:=i+1)<=n}]

  varMonomsPower(n:SI, deg:SI):L Monom ==
  n = 1 => [ monom(deg)]
  ans : L Monom := nil
-- for j in 0..deg repeat
  j:SI:=-1
  while (j:=j+1)<=deg repeat
    ans := concat(varMonomMult(j,varMonomsPower(n-1,deg-j)), ans)
  ans

  varMonomMult(i:SI, mlist : L Monom) : L Monom ==
[varMonomMult(i, m) for m in mlist]

  varMonomMult(i:SI, m:Monom) : Monom ==
nm:Array SI := new(#m + 1, i)
-- for k in 1..#m repeat nm.k :=m.k
  k:SI:=0
while (k:=k+1)<=#m repeat nm.k :=m.k
nm pretend Monom

  (i1:L Monom) * (i2:L Monom):L Monom ==
  import from HilbertFunctionPackage
  ans : L Monom := nil
  for m1 in i1 repeat for m2 in i2 repeat

```



```

      ans := adjoin(m1*m2, ans)
ans

      (i:L Monom) ^ (n:SI) : L Monom ==
n = 1 => i
odd? n => i * (i*i)^shift(n, -1)
(i*i)^shift(n,-1)

```

---

## References

- [1] nothing