Foundations of Computer Algebra: Introduction

Marc Moreno Maza
University of Western Ontario
AM 475 / AM 563 and CS424 / CS556 – Winter 2007

January 7, 2008

1 Computer Algebra

1.1 Objectives

Computer algebra is the set of the techniques for performing algebraic calculations on computers. This includes developing

- algorithms and data structures for these calculations,
- programming languages and software systems implementing them,
- interfaces between computer algebra systems.

By *algebraic calculations* we mean symbolic or exact computations.

- Most of the calculations that we have learned at school were of that kind. For instance we have learned
  - to solve equations of the form $ax^2 + bx + c = 0$,
  - to factor such polynomials,
  - to decide whether a number is prime,
  - to solve systems of linear equations such as $\begin{cases} x + y &= 1 \\ x - y &= 2 \end{cases}$.

- Most of the calculations that scientific calculators perform are not of that kind. For instance, these calculators
  - give approximate solutions of $x^2 + x - 1 = 0$,
  - sometimes factor polynomials of low degree,
  - rarely do primality test,
  - give approximate solutions of linear equations of small order.

Actually, these high-school problems are not that easy:

- solving equations of the form $ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$,
- factoring univariate polynomials, such as $x^4 + 2x^3 + 4x^2 + 2x + 3$,
- deciding whether a number is prime,
- solving systems of linear equations of high order.

Several kinds of difficulties appear there:

- for some problems, we do not know a *practical* algorithm, or the algorithms that we know are too greedy in time or space,
- for other problems, we may not even know an algorithm to compute the solution.

Computer algebra aims at remedying these difficulties, as much as possible.

Let us illustrate the question of exactness with an example. Consider the following system of 3 **non-linear** equations in 3 variable:

$$\begin{cases} x^2 + y + z &= 1 \\ x + y^2 + z &= 1 \\ x + y + z^2 &= 1 \end{cases}$$

The output with `phc` the *symb.-num.* software of J. Verschelde (Univ. Chicago):

```
solution 1 :    start residual :  3.968E-12   #iterations : 1   success
 x :  9.99999695984909E-01   4.13938269379988E-07
 y :  3.04015091103714E-07  -4.13938269379988E-07
 z :  3.04015090976779E-07  -4.13938269379988E-07
== err :  2.154E-06 = rco :  1.197E-07 = res :  9.920E-13 = complex regular ==
solution 2 :    start residual :  1.388E-16   #iterations : 1   success
 x :  4.14213562373095E-01   2.35098870164458E-38
 y :  4.14213562373095E-01  -1.67507944992176E-37
```

```
 z :  4.14213562373095E-01   1.29304378590452E-37
== err :  7.517E-16 = rco :  6.017E-02 = res :  5.551E-17 = real regular ==
solution 3 :    start residual :  2.400E-12   #iterations : 1   success
 x :  1.80048038888678E-08   4.29782537417684E-07
 y :  9.99999981995196E-01  -4.29782537417684E-07
 z :  1.80048038262633E-08   4.29782537417684E-07
== err :  1.344E-06 = rco :  7.463E-08 = res :  5.995E-13 = complex regular ==
solution 4 :    start residual :  9.614E-13   #iterations : 1   success
 x :  1.00000024904061E+00  -3.93267692590196E-08
 y : -2.49040612161639E-07   3.93267692590197E-08
 z : -2.49040612108234E-07   3.93267692590197E-08
== err :  8.657E-07 = rco :  4.806E-08 = res :  2.400E-13 = complex regular ==
solution 5 :    start residual :  2.745E-12   #iterations : 1   success
 x :  3.58839953269127E-07   1.89357516639334E-07
 y :  3.58839953269127E-07   1.89357516639334E-07
 z :  9.99999641160047E-01  -1.89357516639334E-07
== err :  1.645E-06 = rco :  7.071E-08 = res :  6.863E-13 = complex regular ==
solution 6 :    start residual :  1.744E-34   #iterations : 1   success
 x : -2.41421356237309E+00   0.00000000000000E+00
 y : -2.41421356237309E+00   0.00000000000000E+00
 z : -2.41421356237309E+00  -1.00577224408752E-106
== err :  3.611E-35 = rco :  4.142E-01 = res :  6.868E-106 = real regular ==
solution 7 :    start residual :  1.112E-12   #iterations : 1   success
 x : -2.64786238552867E-07  -4.67724648385200E-08
 y : -2.64786238552867E-07  -4.67724648385200E-08
 z :  1.00000026478624E+00   4.67724648385200E-08
== err :  9.341E-07 = rco :  4.530E-08 = res :  2.779E-13 = complex regular ==
solution 8 :    start residual :  2.045E-12   #iterations : 1   success
 x :  1.42636460554469E-07  -3.16738323586431E-07
 y :  9.99999857363539E-01   3.16738323586431E-07
 z :  1.42636460467758E-07  -3.16738323586431E-07
== err :  1.378E-06 = rco :  7.656E-08 = res :  5.117E-13 = complex regular ==
============================================================================
A list of 8 solutions has been refined :
Number of regular solutions   : 8.
Number of singular solutions  : 0.
Number of real solutions      : 2.
Number of complex solutions   : 6.
Number of clustered solutions : 0.
Number of failures            : 0.
============================================================================
```

The solution provided by `triade` my symbolic solver in ALDOR is

```
{z^2 +2*z -1,
y -z,
x -z}

{z,
y -1,
x}

{z,
```

```
y,
x -1}

{z -1,
y,
x}

Dimensions: [0,0,0,0]
Degrees: [2,1,1,1]
```

How to choose between symbolic and numeric methods?

- If the input is an **approximate** system, then only *numeric* (or *symbolic-numeric*) methods make sense.
- If the input is an **exact** system but the user only cares about **approximate** solutions, then *symbolic-numeric* methods are more appropriate.
- If both the input and the desired answer are *exact*, then only symbolic methods make sense. Examples:
  - Computation of canonical forms (in order to recognize certain properties),
  - Looking for real solutions (and discarding solutions with imaginary part)
  - Computing modulo primes (cryptography, auto-correcting codes).

1.2 A breif history

Algorithms for algebraic calculations have been developed since the Antique times

- The Euclidean Algorithm for computing GCDs (Euclide, around 300 B.C.)
- The method of Erastothene for testing primality of numbers (around 200 B.C.)
- Chinese Remaindering Algorithm (Sun-Tsu, 100 A.C)
- etc.

An intense algorithmic activity for solving equations since the Renaissance

- Descates Rule for isolating real roots of univariate polynomials (Descartes 1596 - 1650)
- The complexity analysis of the Euclideean Algorithm by Lamé (1845)
- Newton iteration for solving equations (1669)
- Galois Theory for deciding when univariate equations can be solved by radicals [Esc01]
- First algorithm for factoring univariate polynomials by Kronecker [Kro97b, Kro97a]
- The method of Gauss for multiplying complex numbers, see also Karatsuba's trick [KO63].
- etc.

The discovery of fast algorithms since the introduction of computers

- Karatsuba's trick for univariate polynomial multiplication [KO63],
- FFT-based univariate polynomial multiplication (Cooley and Tukey, 1965),
- First algorithm and its implementation of polynomial system solver [Buc65]
- Strassen's method for matrix multiplication [Str69]
- First efficient algorithm for factoring polynomials [Ber67, Zas69, LHWLL82]
- Fast algorithm for division (Cook, 1966) (Sieveking, 1972)
- Fast Chinese Remaindering Algorithm (Borodin & Moenck)
- Fast algorithms for GCD computations (Knuth, 1970), (Schönage, 1971) and others

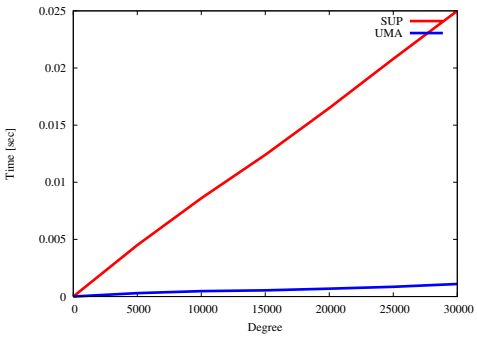

Figure 1: Polynomial addition over $\mathbb{Z}/p\mathbb{Z}$.

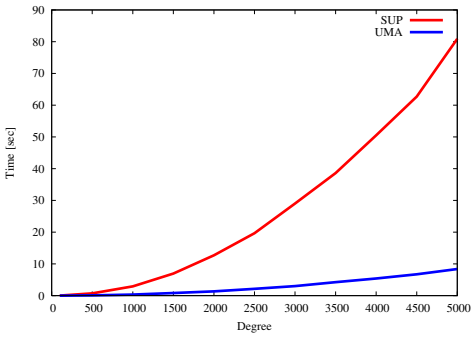

Figure 2: Polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$.

1.3 Challenges and Achievements

The main challenges of symbolic/exact computations:

- intermediate expression swell,
- complexity of symbolic computations
- implementing fast algorithms.

We iluustrate below the gap between naive and optimized implementations of the same algorithm.

Next, we show benchmarks between classical and FFT-based univariate polynomial multiplication.

The maple work sheet `Bareiss.mw` iluustrates the problem of intermediate expression swell. In many situations, this problem is solved by means

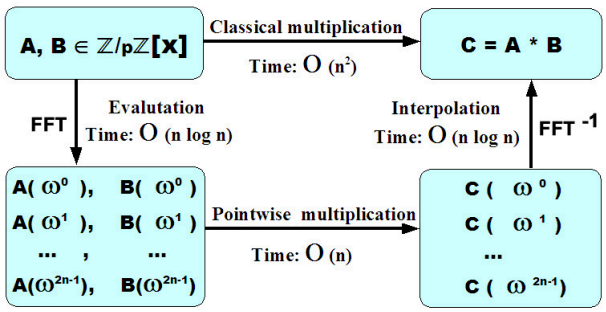

Figure 3: FFT-based univariate polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$

of *modular methods*. Here's an example of such method that will study in detail later

Input: $A = (a_{ij})$ a square matrix over $\mathbb{Z}$ of order n with $\mid a_{ij} \mid \leq C$ for $1 \leq i, j \leq n$.

Output: $\det(A) \in \mathbb{Z}$.

$B := n^{n/2}\, C^n$
$r := \lceil \log_2(2\,B + 1) \rceil$
choose r distinct prime numbers $m_0, \ldots, m_{r-1} \in \mathbb{N}$
for $i = 0 \cdots r - 1$ **repeat**
  $d_i := \det(A \mod m_i)$
$d := \mathit{interpolate}([m_0, \ldots, m_{r-1}], [d_0, \ldots, d_{r-1}])$
$d := d$ rem m
if $d \geq \frac{m}{2}$ **then**
  $d := d - m$
return(d)

Another example of intermediate expression swell is given below. Consider the polynomials

$$r_0 = x^{200} + x^{10} + 1 \quad \text{and} \quad r_1 = x^{80} + x^8 + 1 \tag{1}$$

in the ring $\mathbb{Q}[x]$. The successive remainders of the Euclidean Algorithm applied to r_0 and r_1 are

- $r_2 = x^{56} + 2\ x^{48} + x^{40} + x^{10} + 1$

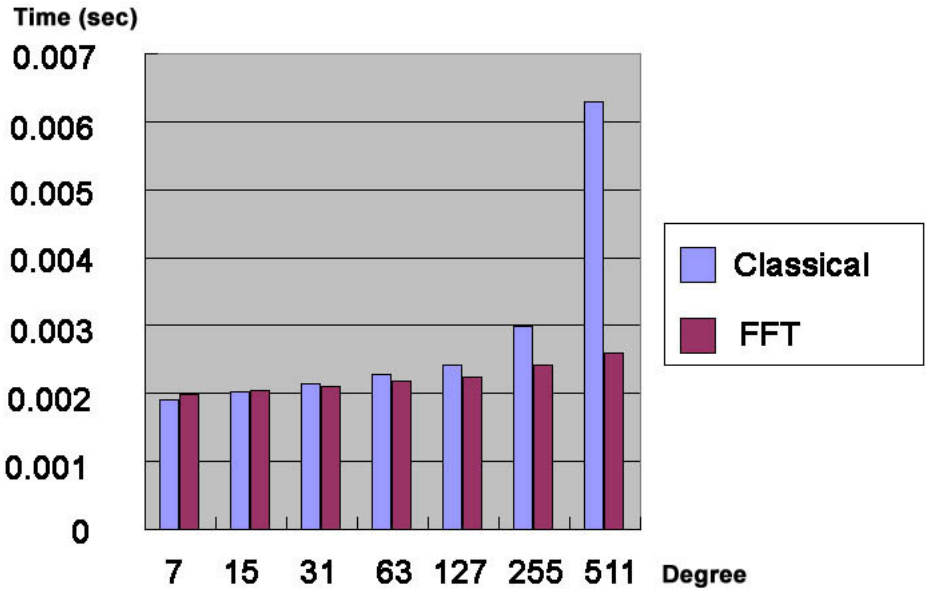

Figure 4: Classical vs FFT-based univariate polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$

- $r_3 = 5\ x^{48} + 4\ x^{40} - x^{34} + 2\ x^{26} - x^{24} - 3\ x^{18} + 2\ x^{16} + 4\ x^{10} - 2\ x^8 + 5$
- $r_4 = \frac{1}{5}\ x^{42} + \frac{1}{25}\ x^{40} - \frac{4}{25}\ x^{34} + \frac{1}{5}\ x^{32} + \frac{3}{25}\ x^{26} - \frac{4}{25}\ x^{24} - \frac{2}{25}\ x^{18} - \frac{2}{25}\ x^{16} + \frac{1}{25}\ x^{10} - \frac{13}{25}\ x^8 - \frac{1}{5}$
- $r_5 = \frac{1001}{625}\ x^{40} - \frac{29}{25}\ x^{38} + \frac{29}{125}\ x^{36} - \frac{154}{625}\ x^{34} - \frac{74}{125}\ x^{32} + \frac{23}{25}\ x^{30} - \frac{23}{125}\ x^{28} + \frac{273}{625}\ x^{26} + \frac{121}{625}\ x^{24} + \frac{8}{25}\ x^{22} - \frac{8}{125}\ x^{20} - \frac{367}{625}\ x^{18} + \frac{123}{625}\ x^{16} + \frac{66}{25}\ x^{14} - \frac{66}{125}\ x^{12} + \frac{566}{625}\ x^{10} - \frac{263}{625}\ x^8 + x^6 - \frac{1}{5}\ x^4 + \frac{1}{25}\ x^2 + \frac{124}{125}$
- $r_6 = \frac{525625}{1002001}\ x^{38} + \frac{20000}{1002001}\ x^{36} - \frac{3750}{13013}\ x^{34} + \frac{768750}{1002001}\ x^{32} - \frac{416875}{1002001}\ x^{30} - \frac{166875}{1002001}\ x^{28} + \frac{2500}{11011}\ x^{26} - \frac{101250}{91091}\ x^{24} - \frac{145000}{1002001}\ x^{22} + \frac{404375}{1002001}\ x^{20} - \frac{184375}{1002001}\ x^{18} - \frac{2166250}{1002001}\ x^{16} - \frac{108750}{91091}\ x^{14} - \frac{23750}{91091}\ x^{12} - \frac{60000}{1002001}\ x^{10} - \frac{2987500}{1002001}\ x^8 - \frac{453125}{1002001}\ x^6 + \frac{90625}{1002001}\ x^4 - \frac{643750}{1002001}\ x^2 - \frac{1575625}{1002001}$
- $r_7 = \frac{511511}{707281}\ x^{36} - \frac{1439438}{707281}\ x^{34} + \frac{1088087}{707281}\ x^{32} + \frac{7007}{24389}\ x^{30} - \frac{558558}{707281}\ x^{28} + \frac{1924923}{707281}\ x^{26} - \frac{861861}{707281}\ x^{24} - \frac{19019}{24389}\ x^{22} + \frac{634634}{707281}\ x^{20} + \frac{2466464}{707281}\ x^{18} - \frac{525525}{707281}\ x^{16} + \frac{10010}{24389}\ x^{14} - \frac{420420}{707281}\ x^{12} + \frac{4358354}{707281}\ x^{10} - \frac{2640638}{707281}\ x^8 - \frac{5005}{24389}\ x^6 + \frac{30030}{24389}\ x^4 + \frac{1477476}{707281}\ x^2 - \frac{1178177}{707281}$

- $r_8 = \frac{1396901}{261121}\ x^{34} - \frac{1306073}{261121}\ x^{32} - \frac{31117}{37303}\ x^{30} - \frac{252300}{261121}\ x^{28} - \frac{2249675}{261121}\ x^{26} + \frac{140447}{37303}\ x^{24} + \frac{407044}{261121}\ x^{22} - \frac{1982237}{261121}\ x^{20} - \frac{487780}{37303}\ x^{18} - \frac{65598}{37303}\ x^{16} - \frac{802314}{261121}\ x^{14} - \frac{248936}{37303}\ x^{12} - \frac{102602}{5329}\ x^{10} + \frac{2434695}{261121}\ x^8 - \frac{458345}{261121}\ x^6 - \frac{1977191}{261121}\ x^4 - \frac{1869543}{261121}\ x^2 + \frac{932669}{261121}$

- $r_9 = \frac{1451751}{2758921}\ x^{32} + \frac{785918}{2758921}\ x^{30} + \frac{494137}{2758921}\ x^{28} + \frac{91469}{2758921}\ x^{26} - \frac{1803830}{2758921}\ x^{24} + \frac{222796}{250811}\ x^{22} + \frac{2809989}{2758921}\ x^{20} + \frac{1538110}{2758921}\ x^{18} - \frac{2954091}{2758921}\ x^{16} + \frac{2029692}{2758921}\ x^{14} + \frac{1194718}{2758921}\ x^{12} + \frac{40369}{2758921}\ x^{10} - \frac{4301598}{2758921}\ x^8 + \frac{1421091}{2758921}\ x^6 + \frac{1051638}{2758921}\ x^4 - \frac{811468}{2758921}\ x^2 - \frac{2893282}{2758921}$

- $r_{10} = \frac{2444992}{8071281}\ x^{30} + \frac{2089538}{8071281}\ x^{28} - \frac{2219096}{8071281}\ x^{26} - \frac{22750717}{8071281}\ x^{24} + \frac{6844981}{8071281}\ x^{22} + \frac{1019854}{2690427}\ x^{20} + \frac{9319871}{8071281}\ x^{18} - \frac{12728243}{2690427}\ x^{16} + \frac{597960}{896809}\ x^{14} - \frac{486673}{8071281}\ x^{12} - \frac{4801951}{8071281}\ x^{10} - \frac{9713528}{2690427}\ x^8 + \frac{352132}{896809}\ x^6 + \frac{191015}{896809}\ x^4 - \frac{1376969}{8071281}\ x^2 - \frac{18359033}{8071281}$

- $r_{11} = \frac{821049}{541696}\ x^{28} + \frac{1230153}{135424}\ x^{26} - \frac{7537173}{1083392}\ x^{24} + \frac{1423341}{1083392}\ x^{22} - \frac{804003}{541696}\ x^{20} + \frac{19361415}{1083392}\ x^{18} - \frac{9889521}{1083392}\ x^{16} + \frac{309669}{135424}\ x^{14} + \frac{2951799}{1083392}\ x^{12} + \frac{12275961}{1083392}\ x^{10} - \frac{1082421}{135424}\ x^8 + \frac{184665}{270848}\ x^6 + \frac{71025}{47104}\ x^4 + \frac{7338303}{1083392}\ x^2 - \frac{4707537}{1083392}$

- $r_{12} = \frac{2879600}{83521}\ x^{26} - \frac{2819432}{83521}\ x^{24} + \frac{687608}{83521}\ x^{22} - \frac{1300512}{83521}\ x^{20} + \frac{5881560}{83521}\ x^{18} - \frac{4015064}{83521}\ x^{16} + \frac{681168}{83521}\ x^{14} + \frac{130456}{83521}\ x^{12} + \frac{3484776}{83521}\ x^{10} - \frac{3296176}{83521}\ x^8 + \frac{218224}{83521}\ x^6 + \frac{112424}{83521}\ x^4 + \frac{2110296}{83521}\ x^2 - \frac{1857480}{83521}$

- $r_{13} = \frac{489294629}{244922500}\ x^{24} - \frac{84853001}{244922500}\ x^{22} + \frac{15566407}{122461250}\ x^{20} - \frac{51711059}{48984500}\ x^{18} + \frac{848011833}{244922500}\ x^{16} - \frac{11383999}{61230625}\ x^{14} + \frac{66509593}{244922500}\ x^{12} + \frac{44817253}{244922500}\ x^{10} + \frac{322278061}{122461250}\ x^8 - \frac{7199857}{61230625}\ x^6 - \frac{2507653}{244922500}\ x^4 + \frac{1074213}{244922500}\ x^2 + \frac{79874687}{48984500}$

- $r_{14} = \frac{101577545700}{2866455549721}\ x^{22} + \frac{367076384000}{2866455549721}\ x^{20} - \frac{333608859000}{2866455549721}\ x^{18} + \frac{272633704600}{2866455549721}\ x^{16} + \frac{73493401600}{2866455549721}\ x^{14} + \frac{181230098700}{2866455549721}\ x^{12} - \frac{95624598700}{2866455549721}\ x^{10} - \frac{70116632400}{2866455549721}\ x^8 + \frac{959856442\,00}{2866455549721}\ x^6 + \frac{93781623400}{2866455549721}\ x^4 - \frac{2339329448\,00}{2866455549721}\ x^2 + \frac{36036190800}{2866455549721}$

- $r_{15} = \frac{179398565293636}{10531900693521}\ x^{20} - \frac{54943352709941}{3510633564507}\ x^{18} + \frac{117687363770051}{10531900693521}\ x^{16} + \frac{9087736866857}{10531900693521}\ x^{14} + \frac{27503113958149}{3510633564507}\ x^{12} - \frac{29314013696810}{10531900693521}\ x^{10} - \frac{7870248236452}{3510633564507}\ x^8 + \frac{27346896912740}{10531900693521}\ x^6 + \frac{61025918153225}{10531900693521}\ x^4 - \frac{95570962105003}{10531900693521}\ x^2 + \frac{2527399767739}{1170211188169}$

- $r_{16} = \frac{2517066778702173}{11227749627077776}\ x^{18} - \frac{38181122126302\,17}{11227749627077776}\ x^{16} + \frac{381679883443605}{11227749627077776}\ x^{14} - \frac{1538892758851425}{11227749627077776}\ x^{12} - \frac{388249516683645}{5613874813538888}\ x^{10} - \frac{691018345032231}{2806937406769444}\ x^8 - \frac{60451575413454}{701734351692361}\ x^6 - \frac{963937271272515}{11227749627077776}\ x^4 - \frac{170313672155631}{11227749627077776}\ x^2 - \frac{2469351761856789}{11227749627077776}$

- $r_{17} = \frac{849182055193466800}{601565221018811449}\ x^{16} + \frac{343702214828429548}{601565221018811449}\ x^{14} + \frac{682224379091263268}{601565221018811449}\ x^{12} + \frac{673295990111439292}{601565221018811449}\ x^{10} + \frac{547091169006122260}{601565221018811449}\ x^8 + \frac{460333296636481632}{601565221018811449}\ x^6 + \frac{383125215937085648}{601565221018811449}\ x^4 + \frac{294034834904863988}{601565221018811449}\ x^2 + \frac{429248529314696936}{601565221018811449}$

- $r_{18} = \frac{8093646079436159529}{64225707449292490000}\ x^{14} + \frac{8963514770014081139}{64225707449292490000}\ x^{12} + \frac{36664367930801793141}{64225707449292490000}\ x^{10} - \frac{5166279318582404449}{128451414898584980000}\ x^8 + \frac{1655921845497232367}{8028213431161561250}\ x^6 + \frac{553008257063777544}{4014106715580780625}\ x^4 + \frac{5923554723478841499}{64225707449292490000}\ x^2 - \frac{311063035823698661}{32112853724646245000}$

- $r_{19} = -\frac{321061300213427783600}{1088944383257885412 09}\ x^{12} + \frac{260178685392579511400}{36298146108596180403}\ x^{10} - \frac{50329696078066373200}{15556348332255505887}\ x^8 + \frac{6523576485821115 4700}{1088944383257885412 09}\ x^6 + \frac{17696479989822163900}{36298146108596180403}\ x^4 + \frac{34027149119020814900}{36298146108596180403}\ x^2 + \frac{4916242201393557 2500}{10889443832578854 1209}$

- $r_{20} = \frac{4829115100553781731631}{4012425965728787627 6}\ x^{10} - \frac{2757078265767537479145}{4012425965728787627 6}\ x^8 + \frac{2023158681969000162489}{8024851931457575245 52}\ x^6 + \frac{1601998453967540114703}{8024851931457575245 52}\ x^4 + \frac{1613072222486808321189}{8024851931457575245 52}\ x^2 + \frac{3731389845047462473 89}{8024851931457575245 52}$

- $r_{21} = -\frac{37302462364041459452987}{21415558969712598439 3929}\ x^8 + \frac{2941869220852112384353}{14277039313141732292 9286}\ x^6 - \frac{10204521350404552579523}{42831117939425196878 7858}\ x^4 - \frac{19661683201604262591215}{42831117939425196878 7858}\ x^2 - \frac{34803628802314892079373}{42831117939425196878 7858}$

- $r_{22} = \frac{10625161139814314894463 99}{55486579353465726911265604}\ x^6 - \frac{19844309975589758372477 07}{55486579353465726911265604}\ x^4 - \frac{99988178500516085183449 35}{55486579353465726911265604}\ x^2 + \frac{13859935515210895384665945}{55486579353465726911265604}$

- $r_{23} = \frac{67553029541578289076276832}{52715901278450400631509 69}\ x^4 + \frac{64692822808128420125817 80}{17571967092816800210503 23}\ x^2 - \frac{11783640725899377196972 8352}{52715901278450400631509 69}$

- $r_{24} = -\frac{14489328729366219403529 3991}{20560880907683469568857 664}\ x^2 + \frac{23864545606303620039559371}{25701101134604336961072 08}$

- $r_{25} = \frac{14749541926460186164326 40872}{39824918466007220836990 67449}$

- $r_{26} = 0$

We will study modular methods for the Euclidean Algorithm too.

2 A First Session with AXIOM

We start by some illustrative examples of Computer Algebra capabilities using the AXIOM system.

- AXIOM was designed at IBM during the 70's and 80's.
- Then AXIOM was maintained at NAG during the 90's.
- AXIOM became in the early 90's the most power computer algebra system, especially thanks to its programming language.
- Then, AXIOM *retired* in 2001 by lack of support, although no other Computer Algebra systems have such powerful programming facilities.

- Today, AXIOM is back as a free software.
- Moreover, AXIOM has a son, ALDOR, born in the early 90's at IBM and growing today at UWO!

For more information about AXIOM please visit:

`http://www.nag.co.uk/symbolic_software.asp`

`http://www.nongnu.org/axiom/`

AXIOM SESSION

```
Starts dribbling to session1.out (Mon Sep 15 12:01:24 2003)


-----------------------------------------
-- There are different types of integers  --
-----------------------------------------

123

   (1)  123
                                                    Type: PositiveInteger
                                                              Time: 0 sec

0

   (2)  0
                                                 Type: NonNegativeInteger
                                                              Time: 0 sec

-45

   (3)  - 45
                                                            Type: Integer
                                                              Time: 0 sec
```

```
----------------------------------------
-- There is a type inference mechanism  --
----------------------------------------

123 + (-45)


   (4)  78
                                                        Type: PositiveInteger
                                                                  Time: 0 sec

123 + (-456)


   (5)  - 333
                                                              Type: Integer
                                                                Time: 0 sec

355/113


        355
   (6)  ---
        113
                                                     Type: Fraction Integer
                                                                  Time: 0 sec

355/113.0


   (7)  3.1415929203 539823009
                                                                 Type: Float
                            Time: 0.01 (IN) + 0.01 (OT) + 0.01 (GC) = 0.03 sec

--------------------------------
-- Coercions are also needed --
--------------------------------

(355/113) :: Float
```

```
   (8)  3.1415929203 539823009
                                                                Type: Float
                                                                Time: 0 sec

(355/113) :: DoubleFloat


   (9)  3.1415929203539825
                                                          Type: DoubleFloat
                                                                Time: 0 sec

-- Type inference again

n: Integer := 12


   (10)  12
                                                             Type: Integer
                                                               Time: 0 sec

a: Integer := 3


   (11)  3
                                                             Type: Integer
                                                               Time: 0 sec

a^n


   (12)  531441
                                                   Type: Fraction Integer
                                                                Time: 0 sec

n := -n


   (13)  - 12
                                                             Type: Integer
```

```
                                                    Time: 0.01 (OT) = 0.01 sec

a^n


            1
   (14)  ------
         531441
                                                Type: Fraction Integer
                                                           Time: 0 sec

inv(a^n)


   (15)  531441
                                                Type: Fraction Integer
                                                           Time: 0 sec

f := n * 1.0


   (16)  - 12.0
                                                             Type: Float
                                                    Time: 0.02 (OT) = 0.02 sec

a^f


   (17)  0.0000018816 7642315892 07457
                                                             Type: Float
                                          Time: 0.01 (IN) + 0.01 (OT) = 0.02 sec

----------------------------------------
-- Types have an influence on the result --
----------------------------------------

gcd (123,45)


   (18)  3
```

```
                                                          Type: PositiveInteger
                                                                    Time: 0 sec

gcd (123,45/1)


   (19)  1
                                                          Type: Fraction Integer
                                 Time: 0.03 (IN) + 0.01 (EV) + 0.01 (OT) = 0.05 sec

x


   (20)  x
                                                             Type: Variable x
                                                                    Time: 0 sec

x + 1


   (21)  x + 1
                                                       Type: Polynomial Integer
                                       Time: 0.01 (IN) + 0.01 (GC) = 0.02 sec

x + 1/2


             1
   (22)  x + -
             2
                                            Type: Polynomial Fraction Integer
                                       Time: 0.04 (IN) + 0.01 (OT) = 0.05 sec

p := (2*x + 2) * (x + 2)


           2
   (23)  2x  + 6x + 4
                                                       Type: Polynomial Integer
                                                  Time: 0.01 (IN) = 0.01 sec
```

```
q := (x + 1) * (x + 3) * 2
```

```
          2
   (24)  2x  + 8x + 6
                                                    Type: Polynomial Integer
                                                  Time: 0.01 (OT) = 0.01 sec
```

```
gcd(p,q)
```

```
   (25)  2x + 2
                                                    Type: Polynomial Integer
                                               Time: 0.01 (OT) + 0.03 (GC) = 0.04 sec
```

```
q := (x + 1) * (x + 3) * 2/1
```

```
          2
   (26)  2x  + 8x + 6
                                          Type: Polynomial Fraction Integer
                 Time: 0.01 (IN) + 0.01 (EV) + 0.01 (OT) + 0.01 (GC) = 0.04 sec
```

```
gcd (p,q)
```

```
   (27)  x + 1
                                          Type: Polynomial Fraction Integer
                                        Time: 0.02 (IN) + 0.01 (EV) = 0.03 sec
```

```
----------------------
-- Specifying types --
----------------------
```

```
Z := Integer
```

```
   (28)  Integer
                                                                 Type: Domain
```

```
                                                                  Time: 0 sec

a: Z := 12

   (29)  12
                                                               Type: Integer
                                                                  Time: 0 sec

b: Z := 45

   (30)  45
                                                               Type: Integer
                                                                  Time: 0 sec

gcd(a,b)

   (31)  3
                                                            Type: PositiveInteger
                                                                  Time: 0 sec

Q := Fraction Integer

   (32)  Fraction Integer
                                                                Type: Domain
                                                                  Time: 0 sec

r: Q := a

   (33)  12
                                                          Type: Fraction Integer
                                                                  Time: 0 sec

s: Q := b
```

```
   (34)  45
                                                        Type: Fraction Integer
                                                                   Time: 0 sec

gcd(r,s)

   (35)  1
                                                        Type: Fraction Integer
                                                                   Time: 0 sec

gcd(12,45)$Q

   (36)  1
                                                        Type: Fraction Integer
                                                                   Time: 0 sec

gcd(12,45)$Z

   (37)  3
                                                                Type: Integer
                                                                   Time: 0 sec

a := inv(a)

   Cannot convert right-hand side of assignment
    1
   --
   12

      to an object of the type Integer of the left-hand side.

--------------------
-- Browsing types --
--------------------

)sh Z
```

```
Integer is a domain constructor.
Abbreviation for Integer is INT
This constructor is exposed in this frame.
Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/integ

------------------------------ Operations ------------------------------

?*? : (Integer,%) -> %                  ?*? : (PositiveInteger,%) -> %
?*? : (%,%) -> %                        ?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %                        ?-? : (%,%) -> %
-? : % -> %                             ?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean                 ?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean                  ?>=? : (%,%) -> Boolean
D : (%,NonNegativeInteger) -> %         D : % -> %
OMwrite : (%,Boolean) -> String         OMwrite : % -> String
1 : () -> %                             0 : () -> %
?^? : (%,PositiveInteger) -> %          abs : % -> %
addmod : (%,%,%) -> %                   associates? : (%,%) -> Boolean
base : () -> %                          binomial : (%,%) -> %
bit? : (%,%) -> Boolean                 coerce : % -> OutputForm
coerce : Integer -> %                   coerce : % -> %
convert : % -> DoubleFloat              convert : % -> Float
convert : % -> InputForm                convert : % -> Integer
convert : % -> Pattern Integer          convert : % -> String
copy : % -> %                           dec : % -> %
differentiate : % -> %                  even? : % -> Boolean
factor : % -> Factored %                factorial : % -> %
gcd : List % -> %                       gcd : (%,%) -> %
hash : % -> SingleInteger               hash : % -> %
inc : % -> %                            init : () -> %
invmod : (%,%) -> %                     latex : % -> String
lcm : List % -> %                       lcm : (%,%) -> %
length : % -> %                         mask : % -> %
max : (%,%) -> %                        min : (%,%) -> %
mulmod : (%,%,%) -> %                   negative? : % -> Boolean
odd? : % -> Boolean                     one? : % -> Boolean
permutation : (%,%) -> %                positive? : % -> Boolean
positiveRemainder : (%,%) -> %          powmod : (%,%,%) -> %
prime? : % -> Boolean                   ?quo? : (%,%) -> %
```

```
 random : % -> %                          random : () -> %
 rational : % -> Fraction Integer         rational? : % -> Boolean
 recip : % -> Union(%,"failed")           ?rem? : (%,%) -> %
 retract : % -> Integer                   sample : () -> %
 shift : (%,%) -> %                       sign : % -> Integer
 sizeLess? : (%,%) -> Boolean             squareFree : % -> Factored %
 squareFreePart : % -> %                  submod : (%,%,%) -> %
 symmetricRemainder : (%,%) -> %          unit? : % -> Boolean
 unitCanonical : % -> %                   zero? : % -> Boolean
 ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?**? : (%,NonNegativeInteger) -> %
 OMwrite : (OpenMathDevice,%,Boolean) -> Void
 OMwrite : (OpenMathDevice,%) -> Void
 ?^? : (%,NonNegativeInteger) -> %
 characteristic : () -> NonNegativeInteger
 differentiate : (%,NonNegativeInteger) -> %
 divide : (%,%) -> Record(quotient: %,remainder: %)
 euclideanSize : % -> NonNegativeInteger
 expressIdealMember : (List %,%) -> Union(List %,"failed")
 exquo : (%,%) -> Union(%,"failed")
 extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
 extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
 gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUn
 multiEuclidean : (List %,%) -> Union(List %,"failed")
 nextItem : % -> Union(%,"failed")
 patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(
 principalIdeal : List % -> Record(coef: List %,generator: %)
 rationalIfCan : % -> Union(Fraction Integer,"failed")
 reducedSystem : Matrix % -> Matrix Integer
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
 retractIfCan : % -> Union(Integer,"failed")
 subtractIfCan : (%,%) -> Union(%,"failed")
 unitNormal : % -> Record(unit: %,canonical: %,associate: %)



)sh Q

 Fraction Integer is a domain constructor.
```

```
Abbreviation for Fraction is FRAC
This constructor is exposed in this frame.
Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/fract

------------------------------- Operations --------------------------------

?*? : (Fraction Integer,%) -> %          ?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %           ?*? : (%,Fraction Integer) -> %
?*? : (%,Integer) -> %                   ?*? : (%,%) -> %
?**? : (%,Integer) -> %                  ?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %                         ?-? : (%,%) -> %
-? : % -> %                              ?/? : (Integer,Integer) -> %
?/? : (%,%) -> %                         ?<? : (%,%) -> Boolean
?<=? : (%,%) -> Boolean                  ?=? : (%,%) -> Boolean
?>? : (%,%) -> Boolean                   ?>=? : (%,%) -> Boolean
D : (%,List Symbol) -> %                 D : (%,NonNegativeInteger) -> %
D : (%,Symbol) -> %                      D : % -> %
OMwrite : (%,Boolean) -> String          OMwrite : % -> String
1 : () -> %                              0 : () -> %
?^? : (%,Integer) -> %                   ?^? : (%,PositiveInteger) -> %
abs : % -> %                             associates? : (%,%) -> Boolean
ceiling : % -> Integer                   coerce : % -> OutputForm
coerce : Fraction Integer -> %           coerce : Integer -> %
coerce : Symbol -> %                     coerce : % -> %
convert : % -> DoubleFloat               convert : % -> Float
convert : % -> InputForm                 convert : % -> Pattern Float
convert : % -> Pattern Integer           denom : % -> Integer
denominator : % -> %                     differentiate : (%,Symbol) -> %
differentiate : % -> %                   ?.? : (%,Integer) -> %
eval : (%,Equation Integer) -> %         eval : (%,Integer,Integer) -> %
eval : (%,Symbol,Integer) -> %           factor : % -> Factored %
floor : % -> Integer                     fractionPart : % -> %
gcd : List % -> %                        gcd : (%,%) -> %
hash : % -> SingleInteger                init : () -> %
inv : % -> %                             latex : % -> String
lcm : List % -> %                        lcm : (%,%) -> %
max : (%,%) -> %                         min : (%,%) -> %
negative? : % -> Boolean                 numer : % -> Integer
numerator : % -> %                       one? : % -> Boolean
positive? : % -> Boolean                 prime? : % -> Boolean
```

```
?quo? : (%,%) -> %                      random : () -> %
recip : % -> Union(%,"failed")          ?rem? : (%,%) -> %
retract : % -> Fraction Integer         retract : % -> Integer
retract : % -> Symbol                   sample : () -> %
sign : % -> Integer                     sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %            squareFreePart : % -> %
unit? : % -> Boolean                    unitCanonical : % -> %
wholePart : % -> Integer                zero? : % -> Boolean
?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
D : (%,List Symbol,List NonNegativeInteger) -> %
D : (%,(Integer -> Integer),NonNegativeInteger) -> %
D : (%,(Integer -> Integer)) -> %
D : (%,Symbol,NonNegativeInteger) -> %
OMwrite : (OpenMathDevice,%,Boolean) -> Void
OMwrite : (OpenMathDevice,%) -> Void
?^? : (%,NonNegativeInteger) -> %
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed")
conditionP : Matrix % -> Union(Vector %,"failed")
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
differentiate : (%,List Symbol) -> %
differentiate : (%,(Integer -> Integer),NonNegativeInteger) -> %
differentiate : (%,(Integer -> Integer)) -> %
differentiate : (%,NonNegativeInteger) -> %
differentiate : (%,Symbol,NonNegativeInteger) -> %
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
eval : (%,List Equation Integer) -> %
eval : (%,List Integer,List Integer) -> %
eval : (%,List Symbol,List Integer) -> %
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial
factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariateP
gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUn
map : ((Integer -> Integer),%) -> %
```

```
 multiEuclidean : (List %,%) -> Union(List %,"failed")
 nextItem : % -> Union(%,"failed")
 patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Floa
 patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult
 principalIdeal : List % -> Record(coef: List %,generator: %)
 reducedSystem : Matrix % -> Matrix Integer
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
 retractIfCan : % -> Union(Fraction Integer,"failed")
 retractIfCan : % -> Union(Integer,"failed")
 retractIfCan : % -> Union(Symbol,"failed")
 solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePoly
 squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynor
 subtractIfCan : (%,%) -> Union(%,"failed")
 unitNormal : % -> Record(unit: %,canonical: %,associate: %)



Q has Field


   (38)  true
                                                                    Type: Boolean
                                                                      Time: 0 sec

Z has Field


   (39)  false
                                                                    Type: Boolean
                                                                      Time: 0 sec

)sh Field

 Field  is a category constructor
 Abbreviation for Field is FIELD
 This constructor is exposed in this frame.
 Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/catde

------------------------------- Operations --------------------------------
 ?*? : (Fraction Integer,%) -> %       ?*? : (%,Fraction Integer) -> %
```

```
?*? : (%,%) -> %                          ?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %            ?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %           ?+? : (%,%) -> %
?-? : (%,%) -> %                          -? : % -> %
?/? : (%,%) -> %                          ?=? : (%,%) -> Boolean
1 : () -> %                               0 : () -> %
?^? : (%,Integer) -> %                    ?^? : (%,PositiveInteger) -> %
associates? : (%,%) -> Boolean            coerce : Fraction Integer -> %
coerce : % -> %                           coerce : Integer -> %
coerce : % -> OutputForm                  factor : % -> Factored %
gcd : List % -> %                         gcd : (%,%) -> %
hash : % -> SingleInteger                 inv : % -> %
latex : % -> String                       lcm : List % -> %
lcm : (%,%) -> %                          one? : % -> Boolean
prime? : % -> Boolean                     ?quo? : (%,%) -> %
recip : % -> Union(%,"failed")            ?rem? : (%,%) -> %
sample : () -> %                          sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %              squareFreePart : % -> %
unit? : % -> Boolean                      unitCanonical : % -> %
zero? : % -> Boolean                      ?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
characteristic : () -> NonNegativeInteger
divide : (%,%) -> Record(quotient: %,remainder: %)
euclideanSize : % -> NonNegativeInteger
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUn
multiEuclidean : (List %,%) -> Union(List %,"failed")
principalIdeal : List % -> Record(coef: List %,generator: %)
subtractIfCan : (%,%) -> Union(%,"failed")
unitNormal : % -> Record(unit: %,canonical: %,associate: %)
```

Z has Ring

```
   (40)  true
                                                                  Type: Boolean
                                                                    Time: 0 sec

)sh Ring

 Ring  is a category constructor
 Abbreviation for Ring is RING
 This constructor is exposed in this frame.
 Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/catde

------------------------------ Operations --------------------------------
 ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %        ?**? : (%,PositiveInteger) -> %
 ?+? : (%,%) -> %                      ?-? : (%,%) -> %
 -? : % -> %                           ?=? : (%,%) -> Boolean
 1 : () -> %                           0 : () -> %
 ?^? : (%,PositiveInteger) -> %        coerce : Integer -> %
 coerce : % -> OutputForm              hash : % -> SingleInteger
 latex : % -> String                   one? : % -> Boolean
 recip : % -> Union(%,"failed")        sample : () -> %
 zero? : % -> Boolean                  ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 characteristic : () -> NonNegativeInteger
 subtractIfCan : (%,%) -> Union(%,"failed")


Z has OrderedRing


   (41)  true
                                                                  Type: Boolean
                                                                    Time: 0 sec

)sh OrderedRing

 OrderedRing  is a category constructor
 Abbreviation for OrderedRing is ORDRING
```

```
 This constructor is exposed in this frame.
 Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/catde

------------------------------- Operations --------------------------------
 ?*? : (%,%) -> %                        ?*? : (Integer,%) -> %
 ?*? : (PositiveInteger,%) -> %          ?**? : (%,PositiveInteger) -> %
 ?+? : (%,%) -> %                        -? : % -> %
 ?-? : (%,%) -> %                        ?<? : (%,%) -> Boolean
 ?<=? : (%,%) -> Boolean                 ?=? : (%,%) -> Boolean
 ?>? : (%,%) -> Boolean                  ?>=? : (%,%) -> Boolean
 1 : () -> %                             0 : () -> %
 ?^? : (%,PositiveInteger) -> %          abs : % -> %
 coerce : Integer -> %                   coerce : % -> OutputForm
 hash : % -> SingleInteger               latex : % -> String
 max : (%,%) -> %                        min : (%,%) -> %
 negative? : % -> Boolean                one? : % -> Boolean
 positive? : % -> Boolean                recip : % -> Union(%,"failed")
 sample : () -> %                        sign : % -> Integer
 zero? : % -> Boolean                    ?~=? : (%,%) -> Boolean
 ?*? : (NonNegativeInteger,%) -> %
 ?**? : (%,NonNegativeInteger) -> %
 ?^? : (%,NonNegativeInteger) -> %
 characteristic : () -> NonNegativeInteger
 subtractIfCan : (%,%) -> Union(%,"failed")


Q has OrderedRing


   (42)  true
                                                              Type: Boolean
                                                                Time: 0 sec

Z has GcdDomain


   (43)  true
                                                              Type: Boolean
                                                  Time: 0.01 (OT) = 0.01 sec
```

```
Q has GcdDomain

   (44)  true
                                                                Type: Boolean
                                                                  Time: 0 sec

Z has EuclideanDomain

   (45)  true
                                                                Type: Boolean
                                                      Time: 0.01 (OT) = 0.01 sec

Q has EuclideanDomain

   (46)  true
                                                                Type: Boolean
                                                                  Time: 0 sec

--------------------------
-- Programming in AXIOM --
--------------------------

f(u,v) == gcd(u,v)

                                                                 Type: Void
                                                                Time: 0 sec

f(a,b)

   Compiling function f with type (Integer,Integer) -> Integer

   (48)  3
                                                          Type: PositiveInteger
                                                                  Time: 0 sec

f(r,s)
```

```
   Compiling function f with type (Fraction Integer,Fraction Integer)
       -> Fraction Integer

   (49)  1
                                                          Type: Fraction Integer
                                                                     Time: 0 sec

g(u,v) == while u ~= v repeat if u < v then v := v - u else u := u - v

                                                                      Type: Void
                                                                     Time: 0 sec

h(u,v) ==
    g(u,v)
    u

                                                                      Type: Void
                                                                     Time: 0 sec

h(12,45)

   Compiling function g with type (PositiveInteger,PositiveInteger) ->
      Void
   Compiling function h with type (PositiveInteger,PositiveInteger) ->
      PositiveInteger

   (52)  12
                                                       Type: PositiveInteger
                                                 Time: 0.03 (IN) = 0.03 sec

h(u,v) ==
   while u ~= v repeat if u < v then v := v - u else u := u - v
   u

   Compiled code for h has been cleared.
   1 old definition(s) deleted for function or rule h
                                                                      Type: Void
                                                                     Time: 0 sec

h(12,45)
```

```
   Compiling function h with type (PositiveInteger,PositiveInteger) ->
      Integer

+++ |*2;h;1;Session1| redefined

   (54)  3
                                                        Type: PositiveInteger
                                                   Time: 0.01 (OT) = 0.01 sec


-------------------------------
-- Functions returning types --
-------------------------------

3/4


         3
   (55)  -
         4
                                                     Type: Fraction Integer
                                                               Time: 0 sec

(x + 1)/(x + 2)


         x + 1
   (56)  -----
         x + 2
                                              Type: Fraction Polynomial Integer
                                                   Time: 0.01 (OT) = 0.01 sec

)sh Fraction

 Fraction S: IntegralDomain  is a domain constructor
 Abbreviation for Fraction is FRAC
 This constructor is exposed in this frame.
 Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/fract
```

```
------------------------------- Operations --------------------------------
?*? : (%,S) -> %                        ?*? : (S,%) -> %
?*? : (Fraction Integer,%) -> %         ?*? : (%,Fraction Integer) -> %
?*? : (%,%) -> %                        ?*? : (Integer,%) -> %
?*? : (PositiveInteger,%) -> %          ?**? : (%,Integer) -> %
?**? : (%,PositiveInteger) -> %         ?+? : (%,%) -> %
?-? : (%,%) -> %                        -? : % -> %
?/? : (S,S) -> %                        ?/? : (%,%) -> %
?=? : (%,%) -> Boolean                  D : (%,(S -> S)) -> %
D : % -> % if S has DIFRING             1 : () -> %
0 : () -> %                             ?^? : (%,Integer) -> %
?^? : (%,PositiveInteger) -> %          associates? : (%,%) -> Boolean
ceiling : % -> S if S has INS           coerce : S -> %
coerce : Fraction Integer -> %          coerce : % -> %
coerce : Integer -> %                   coerce : % -> OutputForm
denom : % -> S                          denominator : % -> %
factor : % -> Factored %                floor : % -> S if S has INS
gcd : List % -> %                       gcd : (%,%) -> %
hash : % -> SingleInteger               init : () -> % if S has STEP
inv : % -> %                            latex : % -> String
lcm : List % -> %                       lcm : (%,%) -> %
map : ((S -> S),%) -> %                 numer : % -> S
numerator : % -> %                      one? : % -> Boolean
prime? : % -> Boolean                   ?quo? : (%,%) -> %
random : () -> % if S has INS           recip : % -> Union(%,"failed")
?rem? : (%,%) -> %                      retract : % -> S
sample : () -> %                        sizeLess? : (%,%) -> Boolean
squareFree : % -> Factored %            squareFreePart : % -> %
unit? : % -> Boolean                    unitCanonical : % -> %
zero? : % -> Boolean                    ?~=? : (%,%) -> Boolean
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?<? : (%,%) -> Boolean if S has ORDSET
?<=? : (%,%) -> Boolean if S has ORDSET
?>? : (%,%) -> Boolean if S has ORDSET
?>=? : (%,%) -> Boolean if S has ORDSET
D : (%,(S -> S),NonNegativeInteger) -> %
D : (%,List Symbol,List NonNegativeInteger) -> % if S has PDRING SYMBOL
D : (%,Symbol,NonNegativeInteger) -> % if S has PDRING SYMBOL
D : (%,List Symbol) -> % if S has PDRING SYMBOL
```

```
D : (%,Symbol) -> % if S has PDRING SYMBOL
D : (%,NonNegativeInteger) -> % if S has DIFRING
OMwrite : (OpenMathDevice,%,Boolean) -> Void if S has INS and S has OM
OMwrite : (OpenMathDevice,%) -> Void if S has INS and S has OM
OMwrite : (%,Boolean) -> String if S has INS and S has OM
OMwrite : % -> String if S has INS and S has OM
?^? : (%,NonNegativeInteger) -> %
abs : % -> % if S has OINTDOM
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if $ has CHARNZ and S has PFECAT or S has CHARNZ
coerce : Symbol -> % if S has RETRACT SYMBOL
conditionP : Matrix % -> Union(Vector %,"failed") if $ has CHARNZ and S has PFECAT
convert : % -> DoubleFloat if S has REAL
convert : % -> Float if S has REAL
convert : % -> InputForm if S has KONVERT INFORM
convert : % -> Pattern Float if S has KONVERT PATTERN FLOAT
convert : % -> Pattern Integer if S has KONVERT PATTERN INT
differentiate : (%,(S -> S)) -> %
differentiate : (%,(S -> S),NonNegativeInteger) -> %
differentiate : (%,List Symbol,List NonNegativeInteger) -> % if S has PDRING SYMBOL
differentiate : (%,Symbol,NonNegativeInteger) -> % if S has PDRING SYMBOL
differentiate : (%,List Symbol) -> % if S has PDRING SYMBOL
differentiate : (%,Symbol) -> % if S has PDRING SYMBOL
differentiate : (%,NonNegativeInteger) -> % if S has DIFRING
differentiate : % -> % if S has DIFRING
divide : (%,%) -> Record(quotient: %,remainder: %)
?.? : (%,S) -> % if S has ELTAB(S,S)
euclideanSize : % -> NonNegativeInteger
eval : (%,Symbol,S) -> % if S has IEVALAB(SYMBOL,S)
eval : (%,List Symbol,List S) -> % if S has IEVALAB(SYMBOL,S)
eval : (%,List Equation S) -> % if S has EVALAB S
eval : (%,Equation S) -> % if S has EVALAB S
eval : (%,S,S) -> % if S has EVALAB S
eval : (%,List S,List S) -> % if S has EVALAB S
expressIdealMember : (List %,%) -> Union(List %,"failed")
exquo : (%,%) -> Union(%,"failed")
extendedEuclidean : (%,%,%) -> Union(Record(coef1: %,coef2: %),"failed")
extendedEuclidean : (%,%) -> Record(coef1: %,coef2: %,generator: %)
factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial
factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariateP
```

```
 fractionPart : % -> % if S has EUCDOM
 gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUn
 max : (%,%) -> % if S has ORDSET
 min : (%,%) -> % if S has ORDSET
 multiEuclidean : (List %,%) -> Union(List %,"failed")
 negative? : % -> Boolean if S has OINTDOM
 nextItem : % -> Union(%,"failed") if S has STEP
 patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Floa
 patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult(
 positive? : % -> Boolean if S has OINTDOM
 principalIdeal : List % -> Record(coef: List %,generator: %)
 reducedSystem : Matrix % -> Matrix S
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix S,vec: Vector S)
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
 reducedSystem : Matrix % -> Matrix Integer if S has LINEXP INT
 retract : % -> Integer if S has RETRACT INT
 retract : % -> Fraction Integer if S has RETRACT INT
 retract : % -> Symbol if S has RETRACT SYMBOL
 retractIfCan : % -> Union(Integer,"failed") if S has RETRACT INT
 retractIfCan : % -> Union(Fraction Integer,"failed") if S has RETRACT INT
 retractIfCan : % -> Union(Symbol,"failed") if S has RETRACT SYMBOL
 retractIfCan : % -> Union(S,"failed")
 sign : % -> Integer if S has OINTDOM
 solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePoly
 squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynom
 subtractIfCan : (%,%) -> Union(%,"failed")
 unitNormal : % -> Record(unit: %,canonical: %,associate: %)
 wholePart : % -> S if S has EUCDOM


)sh Polynomial

 Polynomial R: Ring  is a domain constructor
 Abbreviation for Polynomial is POLY
 This constructor is exposed in this frame.
 Issue )edit /usr/local/share/axiom/axiom2.3//mnt/linuxglibc2.1//../../src/algebra/multp

------------------------------ Operations --------------------------------
 ?*? : (%,R) -> %                      ?*? : (R,%) -> %
 ?*? : (%,%) -> %                      ?*? : (Integer,%) -> %
```

```
?*? : (PositiveInteger,%) -> %          ?**? : (%,PositiveInteger) -> %
?+? : (%,%) -> %                        ?-? : (%,%) -> %
-? : % -> %                             ?=? : (%,%) -> Boolean
D : (%,List Symbol) -> %                D : (%,Symbol) -> %
1 : () -> %                             0 : () -> %
?^? : (%,PositiveInteger) -> %          coefficients : % -> List R
coerce : Symbol -> %                    coerce : R -> %
coerce : Integer -> %                   coerce : % -> OutputForm
differentiate : (%,Symbol) -> %         eval : (%,Symbol,%) -> %
eval : (%,Symbol,R) -> %                eval : (%,List %,List %) -> %
eval : (%,%,%) -> %                     eval : (%,Equation %) -> %
eval : (%,List Equation %) -> %         ground : % -> R
ground? : % -> Boolean                  hash : % -> SingleInteger
latex : % -> String                     leadingCoefficient : % -> R
leadingMonomial : % -> %                map : ((R -> R),%) -> %
monomial? : % -> Boolean                monomials : % -> List %
one? : % -> Boolean                     primitiveMonomials : % -> List %
recip : % -> Union(%,"failed")          reductum : % -> %
retract : % -> Symbol                   retract : % -> R
sample : () -> %                        variables : % -> List Symbol
zero? : % -> Boolean                    ?~=? : (%,%) -> Boolean
?*? : (Fraction Integer,%) -> % if R has ALGEBRA FRAC INT
?*? : (%,Fraction Integer) -> % if R has ALGEBRA FRAC INT
?*? : (NonNegativeInteger,%) -> %
?**? : (%,NonNegativeInteger) -> %
?/? : (%,R) -> % if R has FIELD
?<? : (%,%) -> Boolean if R has ORDSET
?<=? : (%,%) -> Boolean if R has ORDSET
?>? : (%,%) -> Boolean if R has ORDSET
?>=? : (%,%) -> Boolean if R has ORDSET
D : (%,List Symbol,List NonNegativeInteger) -> %
D : (%,Symbol,NonNegativeInteger) -> %
?^? : (%,NonNegativeInteger) -> %
associates? : (%,%) -> Boolean if R has INTDOM
binomThmExpt : (%,%,NonNegativeInteger) -> % if R has COMRING
characteristic : () -> NonNegativeInteger
charthRoot : % -> Union(%,"failed") if $ has CHARNZ and R has PFECAT or R has CHARNZ
coefficient : (%,List Symbol,List NonNegativeInteger) -> %
coefficient : (%,Symbol,NonNegativeInteger) -> %
coefficient : (%,IndexedExponents Symbol) -> R
```

```
coerce : % -> % if R has INTDOM
coerce : Fraction Integer -> % if R has ALGEBRA FRAC INT or R has RETRACT FRAC INT
conditionP : Matrix % -> Union(Vector %,"failed") if $ has CHARNZ and R has PFECAT
content : (%,Symbol) -> % if R has GCDDOM
content : % -> R if R has GCDDOM
convert : % -> InputForm if Symbol has KONVERT INFORM and R has KONVERT INFORM
convert : % -> Pattern Integer if Symbol has KONVERT PATTERN INT and R has KONVERT PATT
convert : % -> Pattern Float if Symbol has KONVERT PATTERN FLOAT and R has KONVERT PATT
degree : (%,List Symbol) -> List NonNegativeInteger
degree : (%,Symbol) -> NonNegativeInteger
degree : % -> IndexedExponents Symbol
differentiate : (%,List Symbol,List NonNegativeInteger) -> %
differentiate : (%,Symbol,NonNegativeInteger) -> %
differentiate : (%,List Symbol) -> %
discriminant : (%,Symbol) -> % if R has COMRING
eval : (%,List Symbol,List %) -> %
eval : (%,List Symbol,List R) -> %
exquo : (%,%) -> Union(%,"failed") if R has INTDOM
exquo : (%,R) -> Union(%,"failed") if R has INTDOM
factor : % -> Factored % if R has PFECAT
factorPolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynomial
factorSquareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariateP
gcd : (%,%) -> % if R has GCDDOM
gcd : List % -> % if R has GCDDOM
gcdPolynomial : (SparseUnivariatePolynomial %,SparseUnivariatePolynomial %) -> SparseUn
integrate : (%,Symbol) -> % if R has ALGEBRA FRAC INT
isExpt : % -> Union(Record(var: Symbol,exponent: NonNegativeInteger),"failed")
isPlus : % -> Union(List %,"failed")
isTimes : % -> Union(List %,"failed")
lcm : (%,%) -> % if R has GCDDOM
lcm : List % -> % if R has GCDDOM
mainVariable : % -> Union(Symbol,"failed")
mapExponents : ((IndexedExponents Symbol -> IndexedExponents Symbol),%) -> %
max : (%,%) -> % if R has ORDSET
min : (%,%) -> % if R has ORDSET
minimumDegree : (%,List Symbol) -> List NonNegativeInteger
minimumDegree : (%,Symbol) -> NonNegativeInteger
minimumDegree : % -> IndexedExponents Symbol
monicDivide : (%,%,Symbol) -> Record(quotient: %,remainder: %)
monomial : (%,List Symbol,List NonNegativeInteger) -> %
```

```
 monomial : (%,Symbol,NonNegativeInteger) -> %
 monomial : (R,IndexedExponents Symbol) -> %
 multivariate : (SparseUnivariatePolynomial %,Symbol) -> %
 multivariate : (SparseUnivariatePolynomial R,Symbol) -> %
 numberOfMonomials : % -> NonNegativeInteger
 patternMatch : (%,Pattern Integer,PatternMatchResult(Integer,%)) -> PatternMatchResult
 patternMatch : (%,Pattern Float,PatternMatchResult(Float,%)) -> PatternMatchResult(Floa
 pomopo! : (%,R,IndexedExponents Symbol,%) -> %
 prime? : % -> Boolean if R has PFECAT
 primitivePart : (%,Symbol) -> % if R has GCDDOM
 primitivePart : % -> % if R has GCDDOM
 reducedSystem : Matrix % -> Matrix R
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix R,vec: Vector R)
 reducedSystem : (Matrix %,Vector %) -> Record(mat: Matrix Integer,vec: Vector Integer)
 reducedSystem : Matrix % -> Matrix Integer if R has LINEXP INT
 resultant : (%,%,Symbol) -> % if R has COMRING
 retract : % -> Integer if R has RETRACT INT
 retract : % -> Fraction Integer if R has RETRACT FRAC INT
 retractIfCan : % -> Union(Symbol,"failed")
 retractIfCan : % -> Union(Integer,"failed") if R has RETRACT INT
 retractIfCan : % -> Union(Fraction Integer,"failed") if R has RETRACT FRAC INT
 retractIfCan : % -> Union(R,"failed")
 solveLinearPolynomialEquation : (List SparseUnivariatePolynomial %,SparseUnivariatePoly
 squareFree : % -> Factored % if R has GCDDOM
 squareFreePart : % -> % if R has GCDDOM
 squareFreePolynomial : SparseUnivariatePolynomial % -> Factored SparseUnivariatePolynor
 subtractIfCan : (%,%) -> Union(%,"failed")
 totalDegree : (%,List Symbol) -> NonNegativeInteger
 totalDegree : % -> NonNegativeInteger
 unit? : % -> Boolean if R has INTDOM
 unitCanonical : % -> % if R has INTDOM
 unitNormal : % -> Record(unit: %,canonical: %,associate: %) if R has INTDOM
 univariate : % -> SparseUnivariatePolynomial R
 univariate : (%,Symbol) -> SparseUnivariatePolynomial %


P := Polynomial Q


   (57)  Polynomial Fraction Integer
```

```
                                                                  Type: Domain
                                                                   Time: 0 sec

P has EuclideanDomain


   (58)  false
                                                                  Type: Boolean
                                                        Time: 0.01 (IN) = 0.01 sec

(x + 1)/(y + 2)


         x + 1
   (59)  -----
         y + 2
                                                Type: Fraction Polynomial Integer
                                                       Time: 0.01 (IN) = 0.01 sec

U := UnivariatePolynomial(x,Q)


   (60)  UnivariatePolynomial(x,Fraction Integer)
                                                                  Type: Domain
                                                                   Time: 0 sec

U has EuclideanDomain


   (61)  true
                                                                  Type: Boolean
                                                                   Time: 0 sec


p1: P := 6 * (x + 1) * (x + 2)


           2
   (62)  6x  + 18x + 12
                                                Type: Polynomial Fraction Integer
```

```
                                                                Time: 0 sec

p2: P := 9 * (x + 1) * ( x + 3)


          2
   (63)  9x  + 36x + 27
                                                Type: Polynomial Fraction Integer
                                                       Time: 0.01 (IN) = 0.01 sec

gcd(p1,p2)


   (64)  x + 1
                                                Type: Polynomial Fraction Integer
                                                       Time: 0.01 (EV) = 0.01 sec

u1: U := p1


          2
   (65)  6x  + 18x + 12
                                   Type: UnivariatePolynomial(x,Fraction Integer)
                                                       Time: 0.03 (IN) = 0.03 sec

u2: U := p2


          2
   (66)  9x  + 36x + 27
                                   Type: UnivariatePolynomial(x,Fraction Integer)
                                                                Time: 0 sec

gcd(u1,u2)


   (67)  x + 1
                                   Type: UnivariatePolynomial(x,Fraction Integer)
                                                       Time: 0.01 (IN) = 0.01 sec
```

```
gcd( (y^3-1) * p1, (y^2 -1) * p2)


   (68)  (x + 1)y - x - 1
                                                    Type: Polynomial Fraction Integer
                                                           Time: 0.02 (EV) = 0.02 sec
Session1 (69) -> )spool

Finished dribbling to session1.out.
```

2.1 Conclusions

- Every AXIOM expression has a unique and precise type
- AXIOM has an inference mechanism to help the user in the desk-calculator mode
- This mechanism uses coercions between types.
- A given mathematical object may have several implementations in AXIOM which one with is different type (for instance multivariate polynomials can be viewed recursively univariate or distributed).
- Types control internal representations and available methods.
- The AXIOM programming language offers *types of types* which are extremely convenient for implementing algebraic structures.
- Types are organized in a hierarchical manner.
- Types of types are called *categories*. Other types are called *domains*.
- An AXIOM function can return a type (or admit a type among its arguments). Such a function is called a *constructor*.

3 A First Session with MAPLE

We continue with some illustrative examples of Computer Algebra capabilities using the MAPLE system.

- MAPLE was designed at the Univ. of Waterloo (Ontario) in the early 80's.
- Today, MAPLE is a successful commercial product with about one million licences around the world.
- MAPLE is a product of the Maplesoft company based at Waterloo (Ontario).
- Several institutions worlwide, in particular the Ontario Research Center for Computer Algebra, contribute significantly to the libraries of MAPLE.
- MAPLE is probably the most quoted computer algebra systems in research articles.

For more information about MAPLE and ORCCA please visit:

`Maplesoft`

`ORCCA`

AXIOM SESSION

```
> restart;
Elementary calculations
> 1/3+2;
                                7
                                -
                                3
> 1/3+2.0;
                           2.333333333
> 5.0^(1/3);
                           1.709975947
> 2345/34570;
                               469
                               ----
                               6914
> 3*x^3-4*x^2+x-7;
                          3      2
                        3 x  - 4 x  + x - 7
> x^2/25+y^2/36;
```

```
                        1   2   1   2
                        -- x  + -- y
                        25      36
> sum('i^2', 'i'=1..100);
                             338350
> sum('i^2', 'i'=1..n);
                1        3   1        2   1     1
                - (n + 1)  - - (n + 1)  + - n + -
                3            2            6     6
> sum('1/i^2', 'i'=1..infinity);
                             1   2
                             - Pi
                             6
> evalf(Pi, 25);
                   3.1415926535897932384626433
> diff(x^3*y^2, y);
                                3
                             2 x  y
> limit((-x^2+x+1)/(x+4), x=infinity);
                            -infinity
> limit(tan(x), x=Pi/2, left);
                             infinity
> int(1/(exp(x^2)+x), x);
                          /     1        \
                       int|-----------, x|
                          |   / 2\       |
                          \exp\x / + x   /
> int(1/x, x=2..4);
                             ln(2)
> int(int(int(x^2*y^2*z^2, x=1..2), y=1..2), z=1..2);
                              343
                              ---
                              27
> with(linalg):
Warning, the protected names norm and trace have been redefined and unprotected
> A := array(1..3,1..3,[[1,x,-x],[-x,0,x],[0,1,x]]);
                             [1   x  -x]
                             [         ]
                        A := [-x  0  x ]
                             [         ]
```

```
                              [0   1  x ]
> det(A);
                                  3    2
                             -x + x  + x
> inverse(A);
        [       1              x + 1            x     ]
        [- -----------    - -----------    ----------]
        [        2               2             2    ]
        [  -1 + x  + x      -1 + x  + x    -1 + x  + x]
        [                                           ]
        [      x               1            x - 1   ]
        [ -----------      -----------     ----------]
        [       2               2             2    ]
        [ -1 + x  + x      -1 + x  + x    -1 + x  + x]
        [                                           ]
        [       1               1             x     ]
        [- -----------  - ---------------  ----------]
        [        2          /      2    \        2    ]
        [  -1 + x  + x    x \-1 + x  + x/  -1 + x  + x]
> det(array(1..2, 1..2, [[a,b],[c,d]]));
                             a d - b c
> solve(cos(x) + y = 9, x);
                        Pi - arccos(y - 9)
> solve(x^3 - 6*x^2 + 11*x - 6, x);
                              1, 2, 3
> solve({x+y=1, 2*x+y=3}, {x,y});
                          {x = 2, y = -1}
> solve(z^4+z^3-2*z-1,z);
                 /  4     3                       \
           RootOf\_Z  + _Z  - 2 _Z - 1, index = 1/,

                   /  4     3                       \
             RootOf\_Z  + _Z  - 2 _Z - 1, index = 2/,

                   /  4     3                       \
             RootOf\_Z  + _Z  - 2 _Z - 1, index = 3/,

                   /  4     3                       \
             RootOf\_Z  + _Z  - 2 _Z - 1, index = 4/
> solve(z^3+z-1,z);
```

```
                   (1/3)
1 /           (1/2)\                   2
- \108 + 12 93     /      - -----------------------,
6                                          (1/3)
                            /           (1/2)\
                            \108 + 12 93     /
                      (1/3)
  1  /           (1/2)\                   1
- -- \108 + 12 93     /      + -----------------------
  12                                          (1/3)
                              /           (1/2)\
                              \108 + 12 93     /

                /                    (1/3)                          \
     1    (1/2) |1 /           (1/2)\                   2           |
   + - I 3      |- \108 + 12 93     /      + -----------------------|,
     2          |6                                             (1/3)|
                |                              /           (1/2)\     |
                \                              \108 + 12 93     /     /
                       (1/3)
  1  /           (1/2)\                   1
- -- \108 + 12 93     /      + -----------------------
  12                                          (1/3)
                              /           (1/2)\
                              \108 + 12 93     /

                /                    (1/3)                          \
     1    (1/2) |1 /           (1/2)\                   2           |
   - - I 3      |- \108 + 12 93     /      + -----------------------|
     2          |6                                             (1/3)|
                |                              /           (1/2)\     |
                \                              \108 + 12 93     /     /
> solve({x^3+y-1,x+y^3-1},{x,y});
                                  /          /  3                      \
{y = 0, x = 1}, {x = 0, y = 1}, { x = RootOf\_Z  + _Z - 1, label = _L3/,
                                  \

                                        \    /
           /  3                      \ }  {
  y = RootOf\_Z  + _Z - 1, label = _L3// ,  \
```

```
           /  4     3                          \
  y = RootOf\_Z  + _Z  - 2 _Z - 1, label = _L4/,

                                                 3    \
            /  4     3                          \      }
  x = -RootOf\_Z  + _Z  - 2 _Z - 1, label = _L4/  + 1/
```

Sequences, sets and lists

```
> restart;
> 1,2,3,4,5;
                          1, 2, 3, 4, 5
> a+b,b+c,c+d,e+f,f+g;
                a + b, b + c, c + d, e + f, f + g
> seq(sin(Pi*i/6), i = 0..3);
                           1  1  (1/2)
                        0, -, - 3     , 1
                           2  2
> i;
                                i
> seq(i!/i^2, i=1..7);
                        1  2  3  24      720
                     1, -, -, -, --, 20, ---
                        2  3  2  5        7
> {1, 1, 2, 3, 2};
                            {1, 2, 3}
> {red, white, blue};
                        {white, blue, red}
> L := [seq(i, i = 0..6)];
                    L := [0, 1, 2, 3, 4, 5, 6]
> {seq(i^2 mod 7, i = L)};
                            {0, 1, 2, 4}
> {a, b, c, d} union {d, e, f};
                        {a, e, b, c, f, d}
> {1, 2, 3, 4, 5} intersect {2, 4, 6, 8, 10};
                              {2, 4}
```

Assignments and equations

```
> restart;
```

```
> i :=1;
>
                              i := 1
> i;
                                1
> unassign('i');
> i;
                                i
> x = y + 3;
                            x = y + 3
> x - y = 1 ;
                            x - y = 1
> x; y;
                                x
                                y
> sols := solve({x+y=3,x-y=1}, {x,y});
                     sols := {y = 1, x = 2}
> x; y;
                                x
                                y
> assign(sols);
> x; y;
                                2
                                1
> x := x + 3 ;
                              x := 5
> unassign('x');
> x := x + 3 ;
Error, recursive assignment
> x;
                                x


Types and representations
> restart;
> whattype(34/57);
                             fraction
> whattype([1,2,3,4,5]);
                               list
> (x+3)*(y-4); whattype((x+3)*(y-4));
                         (x + 3) (y - 4)
```

```
                                *
> whattype(x = y+1);
                                =
> greetings := "hello there";
                   greetings := "hello there"
> type(greetings, integer);
                               false
> type(greetings, string);
                                true
> hastype((x+1/2)*exp(3), fraction);
                                true
> hastype(x^2+3*x+5, `*`);
                                true
> int(exp(-x^2),x); hastype(int(exp(-x^2),x), fraction);
                         1   (1/2)
                         - Pi      erf(x)
                         2
                                true
> has(x^2+3*x+5, 3);
                                true
> has(x^2+3*x+5, 2*x);
                               false
> object := 3*x^2 + 2*x - 3;
                                  2
                    object := 3 x  + 2 x - 3
> nops(object);
                                 3
> op(object);
                             2
                          3 x , 2 x, -3
> op(1, object);
                                  2
                                3 x
> op(1, op(1, object));
                                 3
> whattype(op(2, x^2+exp(1)-3));
                             function
```

Evaluation and substitution

```
> restart;
```

```
> subs(y=ln(x),exp(y));
>
                              exp(ln(x))
> simplify(subs(y=ln(x),exp(y)));
                                 x
> 4^(1/2)+3; simplify(4^(1/2)+3);
                               (1/2)
                              4      + 3
                                 5
```

The computations below show that one can replace a variable by a value
and,,also, a term by a value.

```
> f := x^2 + 3*x + 1; simplify(f, {x^2=1}); simplify(f, {x=1});
                                2
                        f := x  + 3 x + 1
                             3 x + 2
                                5
> f := x^2 + x*y + y^2; simplify(f, {x^2=1, y^2=6});
                                2          2
                        f := x  + x y + y
                             x y + 7
```

More generally, one can simplify an expression by working modulo a relation.
To make clear sense, the precise specifications of "simplify" requires mathematical
notions that are outisde the scope of this introduction.

```
> f := x^2 + 2*x*y + y^2; simplify(f, {x+y=4});
                               2            2
                       f := x  + 2 x y + y
                                16
> restart; simplify(x^2*y, {x^2=x*y});
                                  2
                                 x y
```

In particular, when working modulo several relations, things become more tricky.
Indeed, one has to decide which simplication rule to use first. On the following
example it is easy to see using the first rules twice and then the second once
leads to the returned result.

```
> g := simplify(x^5-x^2-x, {x^3 = x*y, y^2 = x+1});
                               g := 0
```

Solving equations

```
> restart;
>  solve({x^2 - 3 * x + 2 = 0},{x}) ;
                        {x = 2}, {x = 1}
```

```
>  solve(a*x^2 + b*x+ c,x);
                        (1/2)                    (1/2)
            / 2        \             / 2        \
        -b + \b  - 4 a c/         b + \b  - 4 a c/
        ----------------------, - --------------------
                 2 a                       2 a
> solve(a*x^3 + b*x^2 + c*x + d,x):
> solve(a*x^4 + b*x^3 + c*x^2 + b*x + e,x);
                  /    4       3       2           \
            RootOf\a _Z  + b _Z  + c _Z  + b _Z + e/
> d := (m -1)*x -2*(m+1)*y-m;
                d := (m - 1) x - 2 (m + 1) y - m
>  solve({d = 0},{x});
                     /    2 y m + 2 y + m\
                    { x = --------------- }
                     \         m - 1     /
>  solve({abs(x^2 -4*x -3) = 3},{x});
    /          (1/2)\    /          (1/2)\
   { x = 2 + 10      }, { x = 2 - 10      }, {x = 0}, {x = 4}
    \               /    \               /
>  solve({-2*x-3*y=2,4*x+6*y=5},{x,y});
> solve({x -2*y -3,-2*x+4*y+6},{x,y});
                      {x = 3 + 2 y, y = y}
> solve({x-y+z-1,x+y-z-2,x+y+z-3},{x,y,z});
                     /           1      3\
                    { y = 1, z = -, x = - }
                     \           2      2/
>  restart; solve({a*x + b*y - e, c*x + d * y - f},{x,y}) ;
               /    -c e + f a        -d e + b f\
              { y = ----------, x = - ---------- }
               \    d a - b c         d a - b c /
```

Programming

```
> f := proc(x,y) x + y; end;
                f := proc(x, y) x + y end proc;
> f(2,3);
                              5
> print(f);
proc(x, y) x + y end proc;
> f;
```

```
                                f
Please, refer to the Maple documentation.
Representations of polynomials: expression trees
The default representation for polynomials in Maple is by expression trees.
This has the advantage that arithmetic operations have O(1) complexity.
However, a quatity which is null may be represented by something else than 0.
In addition,  the complexity of the equality test is not linear in the size of the input
polynomials. Also finding the degree or the leading term requires to traverse the
representation of the input polynomials.
> restart;
> p1 := x * (x + 1) * (x + 2);
                    p1 := x (x + 1) (x + 2)
> op(p1);
                        x, x + 1, x + 2
> p2 := x^3 + 3 * x^2 + 2 * x;
                             3      2
                     p2 := x  + 3 x  + 2 x
> op(p2);
                          3     2
                         x , 3 x , 2 x
> p3 := ((x + 3) * x + 2) * x;
                    p3 := ((x + 3) x + 2) x
> op(p3);
                        (x + 3) x + 2, x
> p1 - p2;
                                    3      2
              x (x + 1) (x + 2) - x  - 3 x  - 2 x
> expand(p1 - p2);
                                0

Representations of polynomials: canonical representations I
> restart; p := 4*x^3 *y + 5*x*y^2 -7 *x^2 -13 *y + 1;
                      3          2      2
             p := 4 x  y + 5 x y  - 7 x  - 13 y + 1
> vars := [x,y]; p := expand(p,[x,y]);
                         vars := [x, y]
                      3          2      2
             p := 4 x  y + 5 x y  - 7 x  - 13 y + 1
> with(Groebner);
[MulMatrix, SetBasis, ToricIdealBasis, fglm_algo, gbasis, gsolve, hilbertdim,
```

```
  hilbertpoly, hilbertseries, inter_reduce, is_finite, is_solvable, leadcoeff,

  leadmon, leadterm, normalf, pretend_gbasis, reduce, spoly, termorder,

  testorder, univpoly]
> ord := plex(op(vars)); lmp :=  leadterm(p,ord); lcp := leadcoeff(p,ord);
                      ord := plex(x, y)
                                   3
                         lmp := x  y
                           lcp := 4
> [[degree(lmp,vars[1]), degree(lmp,vars[2])], lcp];
                          [[3, 1], 4]
> p := p - lmp  * lcp;
                           2      2
                p := 5 x y  - 7 x  - 13 y + 1
Representations of polynomials: canonical representations II
> restart; read(‘polynomials.input‘):
> vars := [x,y];
                         vars := [x, y]
> p := x^2 + 2*x*y + 3*x + 4*y^2 + 5*y +6;
                    2                    2
             p := x  + 2 x y + 3 x + 4 y  + 5 y + 6
> q := x^2 -2*x*y +3*x -4*y^2 +5*y;
                     2                    2
               q := x  - 2 x y + 3 x - 4 y  + 5 y
> mapleToPol(1, vars); mapleToPol(x, vars); mapleToPol(y, vars);
                         [[[0, 0], 1]]
                         [[[1, 0], 1]]
                         [[[0, 1], 1]]
> tp := mapleToPol(p,vars);
tp := [[[2, 0], 1], [[1, 1], 2], [[1, 0], 3], [[0, 2], 4], [[0, 1], 5],

  [[0, 0], 6]]
> tq := mapleToPol(q, vars);
tq := [[[2, 0], 1], [[1, 1], -2], [[1, 0], 3], [[0, 2], -4], [[0, 1], 5]]
> pp := polToMaple(tp,vars); pp -p ;
                    2                    2
            pp := x  + 2 x y + 3 x + 4 y  + 5 y + 6
                                 0
```

```
> qq := polToMaple(tq,vars); qq -q ;
                      2                    2
              qq := x  - 2 x y + 3 x - 4 y  + 5 y
                                   0
> ts := addPol(tp, tq); polToMaple(ts, vars) - (p + q);
  ts := [[[2, 0], 2], [[1, 0], 6], [[0, 1], 10], [[0, 0], 6]]
                                   0
> tr := polMul(tp, tq); expand(polToMaple(tr, vars) - p * q);
tr := [[[4, 0], 1], [[3, 0], 6], [[2, 2], -4], [[2, 1], 10], [[2, 0], 15],

  [[1, 3], -16], [[1, 1], 18], [[1, 0], 18], [[0, 4], -16], [[0, 2], 1],

  [[0, 1], 30]]
                                   0
```

4 A Second Session with AXIOM

4.1 Operations on rational numbers

AXIOM SESSION

```
FermatNumber(n:PositiveInteger):PositiveInteger == 2**2**n + 1

                                                                  Type: Void
                                                                  Time: 0 sec

l := [FermatNumber(i) for i in 1..6]


   (2)  [5,17,257,65537,4294967297,18446744073709551617]
                                                 Type: List PositiveInteger
                                       Time: 0.01 (IN) + 0.01 (OT) = 0.02 sec
```

```
[prime?(n) for n in l]


   (3)  [true,true,true,true,false,false]
                                                          Type: List Boolean
                                        Time: 0.01 (EV) + 0.02 (OT) = 0.03 sec

[factor(n) for n in l]


   (4)  [5,17,257,65537,641 6700417,274177 67280421310721]
                                                  Type: List Factored Integer
                                        Time: 0.04 (EV) + 0.01 (OT) = 0.05 sec

-- factor FermatNumber(7)

[factorial(n) for n in 99..101]


   (5)
   [
    933262154439441526816992388562667004907159682643816214685929638952175999 93_
     229915608941463976156518286253697920827223758251185210916864000000000000 0_
     000000000
     ,

    933262154439441526816992388562667004907159682643816214685929638952175999 93_
     229915608941463976156518286253697920827223758251185210916864000000000000 0_
     00000000000
     ,

    942594775983835942085162312448293674956231279470254376832788935341697759 93_
     162214765030878615918083469116234900035495995833697063026032640000000000 0_
     0000000000000
     ]
                                                          Type: List Integer
                                        Time: 0.01 (EV) + 0.01 (OT) = 0.02 sec

S(n:PositiveInteger):Fraction Integer == reduce(+,[1/k**2 for k in 1..n])
```

```
                                                                  Type: Void
                                                                 Time: 0 sec
```

```
[S(n) for n in 1..20]
```

```
   (7)
      5  49  205  5269  5369  266681  1077749  9778141  1968329  239437889
   [1, -, --, ---, ----, ----, ------, -------, -------, -------, ---------,
      4  36  144  3600  3600  176400   705600  6350400  1270080  153679680
    240505109  40799043101  40931552621  205234915681  822968714749
    ---------, -----------, -----------, ------------, ------------,
    153679680  25971865920  25971865920  129859329600  519437318400
    238357395880861  238820721143261  86364397717734821  17299975731542641
    ---------------, ---------------, -----------------, -----------------]
    150117385017600  150117385017600  54192375991353600  10838475198270720
                                                 Type: List Fraction Integer
                                                  Time: 0.03 (OT) = 0.03 sec
```

4.2 Operations on real numbers

AXIOM SESSION

```
Ran := RealClosure(Fraction Integer)
```

```
   (1)  RealClosure Fraction Integer
                                                                 Type: Domain
                                                                  Time: 0 sec
```

```
s2 := sqrt(2)$Ran
```

```
         +-+
   (2)  \|2
                                          Type: RealClosure Fraction Integer
                          Time: 0.01 (EV) + 0.02 (OT) + 0.03 (GC) = 0.06 sec
```

```
s5 := sqrt(5)$Ran
```

```
       +-+
   (3)  \|5
                                                Type: RealClosure Fraction Integer
                                                                       Time: 0 sec

s10 := sqrt(10)$Ran


       +--+
   (4)  \|10
                                                Type: RealClosure Fraction Integer
                                                                       Time: 0 sec

eq1:=sqrt(s10+3)*sqrt(s5+2) - sqrt(s10-3)*sqrt(s5-2) = sqrt(10*s2+10)


          +---------+ +--------+    +---------+ +--------+   +-----------+
          | +--+      | +-+         | +--+      | +-+        |   +-+
   (5)  - \|\|10  - 3 \|\|5  - 2  + \|\|10  + 3 \|\|5  + 2 = \|10\|2  + 10
                                 Type: Equation RealClosure Fraction Integer
                           Time: 0.06 (IN) + 0.03 (EV) + 0.01 (OT) = 0.10 sec

eq1::Boolean


   (6)  true
                                                                    Type: Boolean
                                                  Time: 0.19 (IN) = 0.19 sec

precision()$Float


   (7)  68
                                                         Type: PositiveInteger
                                                  Time: 0.01 (OT) = 0.01 sec

f2: Float := sqrt(2)
```

```
   (8)  1.4142135623 730950488
                                                                Type: Float
                                                                Time: 0 sec

f5: Float := sqrt(5)


   (9)  2.2360679774 997896964
                                                                Type: Float
                                                                Time: 0 sec

f10: Float := sqrt(10)


   (10)  3.1622776601 68379332
                                                                Type: Float
                                                                Time: 0 sec

eq1bis:=sqrt(f10+3)*sqrt(f5+2) - sqrt(f10-3)*sqrt(f5-2) = sqrt(10*f2+10)


   (11)  4.9134647270 262305698= 4.9134647270 262305698
                                                       Type: Equation Float
                                                 Time: 0.03 (IN) = 0.03 sec

eq1bis::Boolean


   (12)  true
                                                             Type: Boolean
                                                               Time: 0 sec


r3:Ran:=sqrt(3,5)


         5+-+
   (13)  \|3
                                          Type: RealClosure Fraction Integer
```

Time: 0 sec

```
r25:Ran:=sqrt(1/25,5)
```

$$(14)\quad \sqrt[5]{\frac{1}{25}}$$

Type: RealClosure Fraction Integer
Time: 0 sec

```
r32:Ran:=sqrt(32/5,5)
```

$$(15)\quad \sqrt[5]{\frac{32}{5}}$$

Type: RealClosure Fraction Integer
Time: 0 sec

```
r27:Ran:=sqrt(27/5,5)
```

$$(16)\quad \sqrt[5]{\frac{27}{5}}$$

Type: RealClosure Fraction Integer
Time: 0 sec

```
eq2:=sqrt((r32-r27,3)) = r25*(1+r3-r3**2)
```

$$(17)\quad \sqrt[3]{\sqrt[5]{\frac{27}{5}} + \sqrt[5]{\frac{32}{5}}} = (-\sqrt[3]{3} + \sqrt[3]{3} + 1)\sqrt[5]{\frac{1}{25}}$$

```
                                  Type: Equation RealClosure Fraction Integer
                                       Time: 0.01 (IN) + 0.05 (EV) = 0.06 sec

eq2::Boolean


   (18)  true
                                                                  Type: Boolean
                                       Time: 3.14 (IN) + 0.30 (GC) = 3.44 sec

d3:Float:=nthRoot(3,5)


   (19)  1.2457309396 15517326
                                                                    Type: Float
                                       Time: 0.01 (EV) + 0.01 (OT) = 0.02 sec

d25:Float:=nthRoot(1/25,5)


   (20)  0.5253055608 8075344729
                                                                    Type: Float
                                                  Time: 0.01 (IN) = 0.01 sec

d32:Float:=nthRoot(32/5,5)


   (21)  1.4495593273 553910629
                                                                    Type: Float
                                                  Time: 0.01 (EV) = 0.01 sec

d27:Float:=nthRoot(27/5,5)


   (22)  1.4011310323 534026984
                                                                    Type: Float
                                                                    Time: 0 sec

eq2bis:=nthRoot((d32-d27,3)) = d25*(1+d3-d3**2)
```

```
   (23)  0.3645018412 1606803517= 0.3645018412 1606803518
                                                        Type: Equation Float
                                                  Time: 0.01 (EV) = 0.01 sec

eq2bis::Boolean


   (24)  false
                                                                Type: Boolean
```

4.3 Operations on matrices

AXIOM SESSION

```
N := PositiveInteger


   (1)  PositiveInteger
                                                                Type: Domain
                                                                 Time: 0 sec

D := DoubleFloat


   (2)  DoubleFloat
                                                                Type: Domain
                                                                 Time: 0 sec

F := Float


   (3)  Float
                                                                Type: Domain
                                                                 Time: 0 sec
```

```
Q := Fraction Integer


   (4)  Fraction Integer
                                                                  Type: Domain
                                                                   Time: 0 sec


HilbertMatrixD(n:N): Matrix  D == _
  matrix([[1.0/(i + j) for i in 1..n] for j in 1..n])

                                                                    Type: Void
                                                      Time: 0.01 (IN) = 0.01 sec

HilbertMatrixF(n:N): Matrix  F == _
  matrix([[1.0/(i + j) for i in 1..n] for j in 1..n])

                                                                    Type: Void
                                                      Time: 0.01 (IN) = 0.01 sec

HilbertMatrixQ(n:N): Matrix  Q == _
  matrix([[1/(i + j) for i in 1..n] for j in 1..n])

                                                                    Type: Void
                                                                   Time: 0 sec


HilbertMatrixD(3)


        +       0.5         0.333333333333333        0.25       +
        |                                                       |
   (8)  |0.333333333333333        0.25                0.2       |
        |                                                       |
        +      0.25                0.2         0.166666666666667+
                                                  Type: Matrix DoubleFloat
                            Time: 0.04 (IN) + 0.03 (OT) + 0.01 (GC) = 0.08 sec

HilbertMatrixQ(3)
```

```
        +1  1  1+
        |-  -  -|
        |2  3  4|
        |       |
        |1  1  1|
   (9)  |-  -  -|
        |3  4  5|
        |       |
        |1  1  1|
        |-  -  -|
        +4  5  6+
                                                    Type: Matrix Fraction Integer
                                                       Time: 0.02 (OT) = 0.02 sec

HilbertMatrixF(3)


   (10)
   +          0.5            0.3333333333 3333333333          0.25          +
   |                                                                          |
   |0.3333333333 3333333333          0.25                      0.2          |
   |                                                                          |
   +         0.25                      0.2           0.1666666666 6666666667+
                                                                Type: Matrix Float
                                Time: 0.01 (IN) + 0.01 (OT) + 0.01 (GC) = 0.03 sec


[determinant(HilbertMatrixD(5*n)) for n in 1..6]


   (11)
   [1.4878155287776663e-14, 1.1708847390332544e-58, - 7.329551900980472e-127,
    1.8972182407450097e-202, - 1.7767073603611213e-280, 0.0]
                                                          Type: List DoubleFloat
                                Time: 0.43 (IN) + 0.14 (EV) + 0.06 (OT) = 0.63 sec

[determinant(HilbertMatrixF(5*n)) for n in 1..6]
```

```
   (12)
   [0.1487815528 7758289063 E -13, 0.1171371536 8024357091 E -57,
    0.8781685580 3800165504 E -132, 0.4005398450 3281293592 E -224,
    - 0.8672044362 0589913695 E -319, 0.1929750883 9978983829 E -417]
                                                          Type: List Float
                                        Time: 0.48 (EV) + 0.10 (GC) = 0.58 sec

[determinant(HilbertMatrixQ(5*n)) for n in 1..6]


   (13)
          1                                      1
   [--------------, ------------------------------------------------------------,
    67212633600000  8537000898240926708833515201784986712482596782080000000000

       1
    /
      1465387401836668774813339717252690679593841235023684570865040255471363 38_
       7281567173259185477862791557143607640064000000000000000000000000
     ,

       1
    /
      3277238801332787294766658902228320252045468165204856999173797109784066 89_
       5277922321448374865117026363142310985746482212493949016427083631251603 4_
       1523947219537233313370189682724077124853837717640184856576000000000000 0_
       0000000000000000000000
     ,

       1
    /
      9434434588484412289566936962797607303599034694579364997318510741116152 33_
       1425551441722523624677607868152261744813011187736957248479689761206593 3_
       5223768474732803020308467209016755660686512304691573355703502516536721 1_
       8116107520612044141135536448753640214618431419605770076705202431236268 0_
       2044269396292517037467147505741332480000000000000000000000000000000000 0_
       000000000000000
     ,

       1
```

```
    /
      34767809717363132770618111603748478866514045559675119423637442347693427 8_
       72336185915903600949727565688512425687551728790953230983654612947880686_
       64034084877513911919412368602956835751833985175217203858528427028478411_
       07420065708261775033449064671483311289619473162541454992338969245601369_
       24601181973306894033338133775362887249764402725843635016814711320344415_
       32672272163948431375908684874930724465894995883925403831087253290653960_
       68715359788388174845939833221273153240082092406289203200000000000000000 0_
       00000000000000000000000000000000000000000
     ]
                                                          Type: List Fraction Integer
                                                Time: 0.31 (EV) + 0.01 (OT) = 0.32 sec


h6F := HilbertMatrixF(6)


   (14)
   [
     [0.5, 0.3333333333 3333333333, 0.25, 0.2, 0.1666666666 6666666667,
      0.1428571428 5714285714]
     ,

     [0.3333333333 3333333333, 0.25, 0.2, 0.1666666666 6666666667,
      0.1428571428 5714285714, 0.125]
     ,

     [0.25, 0.2, 0.1666666666 6666666667, 0.1428571428 5714285714, 0.125,
      0.1111111111 1111111111]
     ,

     [0.2, 0.1666666666 6666666667, 0.1428571428 5714285714, 0.125,
      0.1111111111 1111111111, 0.1]
     ,

     [0.1666666666 6666666667, 0.1428571428 5714285714, 0.125,
      0.1111111111 1111111111, 0.1, 0.0909090909 0909090909 1]
     ,
```

```
     [0.1428571428 5714285714, 0.125, 0.1111111111 1111111111, 0.1,
      0.0909090909 0909090909 1, 0.0833333333 3333333333 3]
     ]
                                                          Type: Matrix Float
                                                  Time: 0.02 (OT) = 0.02 sec
h6F * inverse(h6F)


   (15)
   [[1.0,- 0.4 E -14,0.1 E -13,- 0.4 E -13,0.2 E -13,- 0.5 E -14],
    [0.6 E -16,0.9999999999 99999,0.2 E -14,- 0.2 E -13,0.0,- 0.7 E -14],
    [0.6 E -16,- 0.2 E -14,1.0,- 0.2 E -13,0.7 E -14,- 0.4 E -14],
    [0.6 E -16,- 0.2 E -14,0.5 E -14,0.9999999999 9998,0.0,- 0.5 E -14],
    [0.7 E -16,- 0.1 E -14,0.0,- 0.2 E -13,1.0,- 0.4 E -14],
    [0.3 E -16,- 0.9 E -15,0.4 E -14,- 0.1 E -13,0.7 E -14,1.0]]
                                                          Type: Matrix Float
                                        Time: 0.01 (EV) + 0.01 (OT) = 0.02 sec


h6Q := HilbertMatrixQ(6)


         +1  1  1  1   1   1 +
         |-  -  -  -   -   - |
         |2  3  4  5   6   7 |
         |                   |
         |1  1  1  1   1   1 |
         |-  -  -  -   -   - |
         |3  4  5  6   7   8 |
         |                   |
         |1  1  1  1   1   1 |
         |-  -  -  -   -   - |
         |4  5  6  7   8   9 |
   (16)  |                   |
         |1  1  1  1   1    1|
         |-  -  -  -   -   --|
         |5  6  7  8   9   10|
         |                   |
         |1  1  1  1    1   1|
         |-  -  -  -   --  --|
```

```
         |6  7  8  9   10  11|
         |                   |
         |1  1  1   1   1   1|
         |-  -  -  --  --  --|
         +7  8  9  10  11  12+
                                                    Type: Matrix Fraction Integer
                                                                  Time: 0 sec
h6Q * inverse(h6Q)


         +1  0  0  0  0  0+
         |                 |
         |0  1  0  0  0  0|
         |                 |
         |0  0  1  0  0  0|
   (17)  |                 |
         |0  0  0  1  0  0|
         |                 |
         |0  0  0  0  1  0|
         |                 |
         +0  0  0  0  0  1+
                                                    Type: Matrix Fraction Integer
                                                       Time: 0.01 (IN) = 0.01 sec


[determinant(HilbertMatrixF(5*n)) for n in 1..10];


                                                                 Type: List Float
                                                Time: 3.02 (EV) + 0.29 (GC) = 3.31 sec

[determinant(HilbertMatrixQ(5*n)) for n in 1..10];


                                                           Type: List Fraction Integer
                                                Time: 2.96 (EV) + 0.38 (GC) = 3.34 sec


[determinant(HilbertMatrixF(5*n)) for n in 1..15];
```

```
                                                              Type: List Float
                                Time: 0.01 (IN) + 14.09 (EV) + 1.59 (GC) = 15.69 sec

[determinant(HilbertMatrixQ(5*n)) for n in 1..15];


                                                    Type: List Fraction Integer
                                      Time: 19.41 (EV) + 2.75 (GC) = 22.16 sec



[determinant(HilbertMatrixF(5*n)) for n in 1..20];


                                                              Type: List Float
                                      Time: 42.99 (EV) + 5.07 (GC) = 48.06 sec

[determinant(HilbertMatrixQ(5*n)) for n in 1..20];


                                                    Type: List Fraction Integer
                                     Time: 75.55 (EV) + 11.16 (GC) = 86.71 sec
```

4.4 Operations on univariate polynomials

AXIOM SESSION

```
N := NonNegativeInteger


   (1)  NonNegativeInteger
                                                                   Type: Domain
                                                                    Time: 0 sec

Z := Integer
```

```
   (2)  Integer
                                                                Type: Domain
                                                                 Time: 0 sec

UZ := UnivariatePolynomial(x,Z)


   (3)  UnivariatePolynomial(x,Integer)
                                                                Type: Domain
                                                                 Time: 0 sec


u: UZ  := (3*x-1)**2 * (2*x + 8)


          3      2
   (4)  18x  + 60x  - 46x + 8
                                        Type: UnivariatePolynomial(x,Integer)
                                                  Time: 0.01 (OT) = 0.01 sec

v: UZ := (1 - 6*x + 9*x**2)**2


          4       3      2
   (5)  81x  - 108x  + 54x  - 12x + 1
                                        Type: UnivariatePolynomial(x,Integer)
                                                  Time: 0.01 (OT) = 0.01 sec


u**2 + u*v


            7        6        5         4        3        2
   (6)  1458x  + 3240x  - 7074x  + 10584x  - 9282x  + 4120x  - 878x + 72
                                        Type: UnivariatePolynomial(x,Integer)
                                                  Time: 0.01 (OT) = 0.01 sec

leadingCoefficient u
```

```
   (7)  18
                                                        Type: PositiveInteger
                                                                  Time: 0 sec

degree u

   (8)  3
                                                        Type: PositiveInteger
                                                                  Time: 0 sec

reductum u

          2
   (9)  60x  - 46x + 8
                                          Type: UnivariatePolynomial(x,Integer)
                                                                  Time: 0 sec

gcd(u,v)

          2
   (10)  9x  - 6x + 1
                                          Type: UnivariatePolynomial(x,Integer)
                                                                  Time: 0 sec

factor(u)

                         2
   (11)  2(x + 4)(3x - 1)
                                Type: Factored UnivariatePolynomial(x,Integer)
                                                                  Time: 0 sec

factor(v)
```

```
                4
   (12)  (3x - 1)
                                Type: Factored UnivariatePolynomial(x,Integer)
                                                                Time: 0 sec


PZ := Polynomial Z


   (13)  Polynomial Integer
                                                                Type: Domain
                                                                 Time: 0 sec

pz: PZ := (4*x**3+2*y**2+1)*(12*x**5-x**3*y+12)


             3 3       5       2        6    3        8      5      3
   (14)  - 2x y  + (24x  + 24)y  + (- 4x  - x )y + 48x  + 12x  + 48x  + 12
                                                  Type: Polynomial Integer
                                                 Time: 0.01 (IN) = 0.01 sec

factor(pz)


              3       5         2     3
   (15)  - (x y - 12x  - 12)(2y  + 4x  + 1)
                                          Type: Factored Polynomial Integer
                                         Time: 0.04 (EV) = 0.04 sec



Q := Fraction Integer


   (16)  Fraction Integer
                                                                Type: Domain
                                                                 Time: 0 sec

PQ := Polynomial Q
```

```
   (17)  Polynomial Fraction Integer
                                                                  Type: Domain
                                                                   Time: 0 sec
```

```
pq: PQ := (4*x**3+(2/3)*x**2+1)*(12*x**5-(1/2)*x**3+12)
```

```
            8     7     6   35  5   95  3     2
   (18)  48x  + 8x  - 2x  + -- x  + -- x  + 8x  + 12
                            3       2
                                          Type: Polynomial Fraction Integer
                                    Time: 0.03 (IN) + 0.01 (OT) = 0.04 sec
```

```
factor(pq)
```

```
             3   1  2   1   5    1  3
   (19)  48(x  + - x  + -)(x  - -- x  + 1)
                 6      4       24
                                    Type: Factored Polynomial Fraction Integer
                                        Time: 0.03 (EV) + 0.01 (OT) = 0.04 sec
```

AXIOM SESSION

```
P := UnivariatePolynomial(x,Fraction Integer)
```

```
   (1)  UnivariatePolynomial(x,Fraction Integer)
                                                                  Type: Domain
                                                                   Time: 0 sec
```

```
euclideanGcd(a,b) ==
        l := [a];
        a:=unitCanonical a
        b:=unitCanonical b
```

```
        while not zero? b repeat
           l := cons(b,l)
           (a,b):= (b,a rem b)
           b:=unitCanonical b
        reverse l
```

```
                                                                  Type: Void
                                                                 Time: 0 sec
```

```
p1: P  := x^8 + x^6  -3*x^4 -3*x^3 +8*x^2 +2*x -5
```

```
         8    6     4     3     2
   (3)  x  + x  - 3x  - 3x  + 8x  + 2x - 5
                              Type: UnivariatePolynomial(x,Fraction Integer)
                                                  Time: 0.01 (OT) = 0.01 sec
q1: P  := 3*x^6 + 5*x^4 - 4*x^2 -9*x +21
```

```
          6     4     2
   (4)  3x  + 5x  - 4x  - 9x + 21
                              Type: UnivariatePolynomial(x,Fraction Integer)
                                      Time: 0.01 (IN) + 0.01 (OT) = 0.02 sec
```

```
euclideanGcd(p1,q1)
```

```
   (5)
     8    6     4     3     2            6   5  4   4  2
   [x  + x  - 3x  - 3x  + 8x  + 2x - 5, x  + - x  - - x  - 3x + 7,
                                          3      3
     4   1  2   3   2   25     49      6150
    x  - - x  + -, x  + -- x - --, x - ----, 1]
         5      5       13     13      4663
                        Type: List UnivariatePolynomial(x,Fraction Integer)
                                      Time: 0.01 (IN) + 0.01 (OT) = 0.02 sec
```

```
p2: P := x^30 + x^19 + 1
```

```
        30    19
   (6)  x   + x   + 1
                                Type: UnivariatePolynomial(x,Fraction Integer)
                                                              Time: 0 sec
q2: P := x^29 - x^17 + 3


        29    17
   (7)  x   - x   + 3
                                Type: UnivariatePolynomial(x,Fraction Integer)
                                                              Time: 0 sec

euclideanGcd(p2,q2)


   (8)
     30    19       29    17       19    18
   [x   + x   + 1, x   - x   + 3, x   + x   - 3x + 1,

        18    17     11     10     9     8     7     6     5     4     3     2
       x   + x   - 3x   + 4x   - 4x  + 4x  - 4x  + 4x  - 4x  + 4x  - 4x  + 4x
     +
       - 4x - 2
     ,

        12   4  11   4  10   4  9   4  8   4  7   4  6   4  5   4  4   4  3
       x   - - x   + - x   - - x  + - x  - - x  + - x  - - x  + - x  - - x
             3       3       3      3      3      3      3      3      3
     +
       4  2   1     1
       - x  - - x + -
       3      3     3
     ,

        11   8732  10   8732  9   8732  8   10919  7   5816  6   10433  5
       x   - ---- x   + ---- x  - ---- x  + ----- x  - ---- x  + ----- x
             6545       6545      6545       6545      6545       6545
     +
         676  4   9164  3   8588  2   1756      878
```

```
  - --- x  + ---- x  - ---- x  + ---- x + ----
    595      6545      6545      1309     1309
,

   10    9   19619  8   6553  7   15277  6       5       4      3      2
  x   - x  - ----- x  - ---- x  - ----- x  - 729x  - 242x  - 82x  - 26x
               16         4         16
+
    59065     19555
  - ----- x + -----
      16        16
,
 9    8   1  7   1  6     2        1
x  + x  + - x  + - x  + 3x  - 2x + -,
         3      3                 3

   8   78620  7   45799  6   34992  5   11616  4    4080  3    864   2
  x  + ----- x  + ----- x  + ----- x  + ----- x  + ----- x  + ----- x
       58777      58777      58777      58777      58777      58777
+
  177403     58697
  ------ x - -----
   58777     58777
,

   7   170782  6   2173617  5   1746351  4   5658021  3   8599611  2
  x  + ------ x  + ------- x  - ------- x  + ------- x  - ------- x
       923785      3695140      3695140      3695140      3695140
+
  2278869     1232437
  ------- x - -------
   739028     1847570
,

   6   2515753  5   15269857  4   12227447  3   2492039  2   2531080
  x  - ------- x  + -------- x  - -------- x  + ------- x  - ------- x
        142893       428679        428679        142893       428679
+
  4446728
  -------
```

```
      428679
    ,
     5   160644757  4   130375607  3   27115175  2   25927747     49124846
    x  - --------- x  + --------- x  - -------- x  + -------- x - --------,
          73785531       73785531      24595177      73785531     73785531
     4   4553589485  3   1129145127  2   163361143     2886544997
    x  - ---------- x  + ---------- x  + --------- x + ----------,
         7507014112      3753507056      234594191     7507014112
     3   112557717874  2   270014099776      15748937137
    x  - ------------ x  + ------------ x - ------------,
          43571774195      130715322585     130715322585
     2   619536582958      472414012299      19773994490288
    x  - ------------ x + -------------, x + --------------, 1]
         633789802471     2535159209884      34258486832515
                                                  Type: List UnivariatePolynomial(x,Fraction Integer)
                                                                                  Time: 0.04 (OT) = 0.04 sec
```

4.5 Algebraic systems

AXIOM SESSION

```
N := NonNegativeInteger
```

```
   (3)  NonNegativeInteger
                                                                                  Type: Domain
                                                                                   Time: 0 sec
```

```
P := Polynomial Integer
```

```
   (4)  Polynomial Integer
                                                                                  Type: Domain
                                                                                   Time: 0 sec
```

```
epsilon := 1/10^21 :: Float
```

```
   (5)  0.1 E -20
                                                                  Type: Float
                                                                  Time: 0 sec

syst(n:N): List P == [x^n+y+z-1, x+y^n+z-1, x+y+z^n-1]

                                                                   Type: Void
                                                                  Time: 0 sec


completeExactSolve(syst(1))


   (7)  [[z= - %D - %C + 1,y= %C,x= %D]]
                        Type: List List Equation Fraction Polynomial Integer
                                                 Time: 0.03 (IN) = 0.03 sec


completeExactSolve(syst(2))


   (8)
   [[z= 0,y= 0,x= 1], [z= 0,y= 1,x= 0], [z= 1,y= 0,x= 0],
                 2
    [z= x,y= x,x  + 2x - 1= 0]]
                        Type: List List Equation Fraction Polynomial Integer
                                                 Time: 0.09 (EV) = 0.09 sec

realApproximateSolve(syst(2),[x,y,z],epsilon)


   (9)
   [[- 2.4142135623 730950488,- 2.4142135623 730950488,- 2.4142135623 730950488]
     ,
    [0.4142135623 730950488,0.4142135623 730950488,0.4142135623 730950488],
    [1.0,0.0,0.0], [0.0,1.0,0.0], [0.0,0.0,1.0]]
                                                       Type: List List Float
                                    Time: 0.10 (EV) + 0.02 (OT) = 0.12 sec
```

```
completeExactSolve(syst(3))
```

```
   (10)
   [[z= - 1,y= 1,x= 1], [z= 0,y= 0,x= 1], [z= 1,y= - 1,x= 1], [z= 0,y= 1,x= 0],
    [z= 1,y= 0,x= 0], [z= 1,y= 1,x= - 1],
                 2          2         3
    [z= - y - x,y  + x y + x  - 1= 0,x  - x - 1= 0],
                                       3           3
                3                    - x  + 1    - x  + 1  4    2
    [z= x,y= x,x  + 2x - 1= 0], [z= --------,y= --------,x  - x  - 2x + 3= 0],
                                         2           2
                3          4    3     2
    [z= x,y= - x  - x + 1,x  + x  + 2x  + 1= 0],
          3               4    3     2
    [z= - x  - x + 1,y= x,x  + x  + 2x  + 1= 0]]
                          Type: List List Equation Fraction Polynomial Integer
               Time: 0.01 (IN) + 1.58 (EV) + 0.01 (OT) + 0.13 (GC) = 1.73 sec
```

```
realApproximateSolve(syst(3),[x,y,z],epsilon)
```

```
   (11)
   [[0.4533976515 1640376765,0.4533976515 1640376765,0.4533976515 1640376765],
    [1.0,1.0,- 1.0], [1.0,0.0,0.0], [0.0,1.0,0.0], [1.0,- 1.0,1.0],
    [0.0,0.0,1.0], [- 1.0,1.0,1.0]]
                                                          Type: List List Float
                          Time: 1.56 (EV) + 0.01 (OT) + 0.12 (GC) = 1.69 sec
```

```
completeExactSolve(syst(4))
```

```
   (12)
   [[z= 0,y= 0,x= 1], [z= 0,y= 1,x= 0], [z= 1,y= 0,x= 0],
                4
    [z= x,y= x,x  + 2x - 1= 0],
          4                11     8     7     5     4     3
    [z= - x  - x + 1,y= x,x   + 2x  - 3x  + 2x  - 4x  + 3x  - 2x + 2= 0],
```

```
              4          11     8     7     5     4     3
    [z= x,y= - x  - x + 1,x   + 2x  - 3x  + 2x  - 4x  + 3x  - 2x + 2= 0],

           4           4
       - x  + 1     - x  + 1
     [z= --------, y= --------,
            2           2
      11    10    9    8     7     6     4     3    2
     x   + x   + x  - x  - 4x  - 4x  + 4x  + 7x  - x  - 5x - 7= 0]
     ,

               4       2     4          8    5     4    2
     [z= - y - x  + 1, y  + (x  - 1)y + x  - x  - 2x  + x  + x + 1= 0,
      12     9     8     6     5     4     2
     x   - 2x  - 3x  + 2x  + 4x  + 3x  - 2x  - 2x - 2= 0]
     ]
                          Type: List List Equation Fraction Polynomial Integer
                 Time: 0.01 (IN) + 9.63 (EV) + 0.03 (OT) + 0.96 (GC) = 10.63 sec

realApproximateSolve(syst(4),[x,y,z],epsilon)


   (13)
   [[- 0.8755550011 8456161483,- 0.8755550011 8456161483,1.2878847152 284854786]
    ,
    [- 0.8755550011 8456161482,1.2878847152 284854786,- 0.8755550011 8456161482]
    ,
    [1.2878847152 284854786,- 0.8755550011 8456161482,- 0.8755550011 8456161482]
    ,
    [- 1.3953369944 670730188,- 1.3953369944 670730188,- 1.3953369944 670730188]
    ,
    [0.4746266175 6260555033,0.4746266175 6260555033,0.4746266175 6260555033],
    [1.0,0.0,0.0], [0.0,1.0,0.0], [0.0,0.0,1.0]]
                                                          Type: List List Float
                           Time: 9.63 (EV) + 0.01 (OT) + 0.83 (GC) = 10.47 sec


-- completeExactSolve(syst(5))    MORE THAN FIVE MUNITES !!!!
```

4.6 Conclusions

- Observations.
  - Computer algebra algorithms perform symbolic computations: numbers are manipulated by using their mathematical definitions.
  - Hence results are exact and complete.
  - However they can be huge!
  - Moreover intermediate expressions may be much bigger than the input and output.
- Objectives.
  - Our main interest will be here to study the implementation of algorithms
    - that keep the swell of the intermediate expressions under control
    - and offer *optimal* time complexity.
  - Sine this is obviously a vast subject we will focus on univariate polynomials, bivariate polynomials and matrix operations.
  - But in fact these algorithms benefit to many other Computer algebra algorithms.
- Typical (trivial) example.
  - Let $p(x)$ and $q(x)$ be two univariate polynomials over the integer numbers and with degrees n and m respectively such that $n < m$.
  - Suppose we want to decide whether $p(x)$ divides $q(x)$.
  - One can show that the division with remainder of $q(x)$ by $p(x)$ requires at most $(2m+1)(n-m+1)$ operations in the coefficient ring.
  - If $p(x)$ divides $q(x)$ then for a given integer value x_0 of x the integer $p(x_0)$ divides $q(x_0)$.

– Computing $p(x_0)$ and $q(x_0)$ require $2\,n+2\,m+2$ operations in the coefficient ring.

– So trying whether $p(x_0)$ divides $q(x_0)$ (before computing the remainder of $q(x)$ by $p(x)$) will save time (and space) in the average.

5 A review of complexity notions

5.1 Orders of magnitude

Let f, g et h be functions from $\mathbb{N}$ to $\mathbb{R}$

- We say that $g(n)$ is in the ORDER OF MAGNITUDE of $f(n)$ and we write $f(n) \in \Theta(g(n))$ if there exist two (strictly) positive constants c_1 and c_2 such that for n big enough we have

$$0 \quad \leq \quad c_1 \;\; g(n) \quad \leq \quad f(n) \quad \leq \quad c_2 \;\; g(n). \tag{2}$$

- We say that $g(n)$ is an ASYMPTOTIC UPPER BOUND of $f(n)$ and we write $f(n) \in \mathcal{O}(g(n))$ if there exists a (strictly) positive constants c_2 such that for n big enough we have

$$0 \quad \leq \quad f(n) \quad \leq \quad c_2 \;\; g(n). \tag{3}$$

- We say that $g(n)$ is an ASYMPTOTIC LOWER BOUND of $f(n)$ and we write $f(n) \in \Omega(g(n))$ if there exists a (strictly) positive constants c_1 such that for n big enough we have

$$0 \quad \leq \quad c_1 \;\; g(n) \quad \leq \quad f(n). \tag{4}$$

Let us give some examples.

- With $f(n) = \frac{1}{2}n^2 - 3n$ and $g(n) = n^2$ we have $f(n) \in \Theta(g(n))$. Indeed we have

$$c_1\, n^2 \quad \leq \quad \frac{1}{2}n^2 - 3n \quad \leq \quad c_2\, n^2. \tag{5}$$

  for $n \geq 12$ with $c_1 = \frac{1}{4}$ and $c_2 = \frac{1}{2}$.

- Assume that there exists a positive integer n_0 such that $f(n) > 0$ and $g(n) > 0$ for every $n \geq n_0$. Then we have

$$max(f(n), g(n)) \in \Theta(f(n) + g(n)). \tag{6}$$

  Indeed we have

$$\frac{1}{2}(f(n) + g(n)) \quad \leq \quad max(f(n), g(n)) \quad \leq \quad (f(n) + g(n)). \tag{7}$$

- Assume a and b are positive real constants. Then we have

$$(n + a)^b \in \Theta(n^b). \tag{8}$$

  Indeed for $n \geq a$ we have

$$0 \quad \leq \quad n^b \quad \leq \quad (n + a)^b \quad \leq \quad (2n)^b. \tag{9}$$

  Hence we can choose $c_1 = 1$ and $c_2 = 2^b$.

We have the following properties.

- $f(n) \in \Theta(g(n))$ holds iff $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$ hold together.
- Each of the predicates $f(n) \in \Theta(g(n))$, $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$ define a reflexive and transitive binary relation among the $\mathbb{N}$-to-$\mathbb{R}$ functions. Moreover $f(n) \in \Theta(g(n))$ is symmetric.
- We have the following TRANSPOSITION FORMULA

$$f(n) \in \mathcal{O}(g(n)) \iff g(n) \in \Omega(f(n)). \tag{10}$$

In practice $\in$ is replaced by $=$ in each of the expressions $f(n) \in \Theta(g(n))$, $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$. Hence, the following

$$f(n) = h(n) + \Theta(g(n)) \tag{11}$$

means

$$f(n) - h(n) \in \Theta(g(n)). \tag{12}$$

Let us give another fundamental example. Let $p(n)$ be a (univariate) polynomial with degree $d > 0$. Let a_d be its leading coefficient and assume $a_d > 0$. Then we have

(1) if $k \geq d$ then $p(n) \in \mathcal{O}(n^k)$,

(2) if $k \leq d$ then $p(n) \in \Omega(n^k)$,

(3) if $k = d$ then $p(n) \in \Theta(n^k)$.

Exercise: Prove the following

$$\Sigma_{k=1}^{k=n} k \ \in \ \Theta(n^2). \tag{13}$$

5.2 The complexity of divide-and-conquer algorithms

DIVIDE-AND-CONQUER ALGORITHMS proceed as follows.

Divide the input problem into sub-problems.

Conquer on the sub-problems by solving them directly if they are small enough or proceed recursively.

Combine the solutions of the sub-problems to obtain the solution of the input problem.

EQUATION SATISFIED BY $T(n)$. Assume that the size of the input problem increases with an integer n. Let $T(n)$ be the time complexity of a divide-and-conquer algorithm to solve this problem. Then $T(n)$ satisfies an equation of the form:

$$T(n) = a\,T(n/b) + f(n). \tag{14}$$

where $f(n)$ is the cost of the combine-part, $a \geq 1$ is the number of recursively calls and n/b with $b > 1$ is the size of a sub-problem.

LABELED TREE ASSOCIATED WITH THE EQUATION. Assume n is a power of b, say $n = b^p$. To *solve* this equation we can associate a labeled tree $\mathcal{A}(n)$ to it as follows.

(1) If $n = 1$, then $\mathcal{A}(n)$ is reduced to a single leaf by labeled $T(1)$.

(2) If $n > 1$, then the root of $\mathcal{A}(n)$ is labeled by $f(n)$ and $\mathcal{A}(n)$ possesses a labeled sub-trees all equal to $\mathcal{A}(n/b)$.

The labeled tree $\mathcal{A}(n)$ associated with $T(n) = a\,T(n/b) + f(n)$ has height $p + 1$. Moreover the sum of its labels is $T(n)$.

LET US GIVE TWO EXAMPLES.

- Consider the relation:

$$T(n) = 2\,T(n/2) + n^2. \tag{15}$$

We obtain:

$$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \cdots + \frac{n^2}{2^p} + n\,T(1). \tag{16}$$

Hence we have:

$$T(n) \in \Theta(n^2). \tag{17}$$

- Consider the relation:

$$T(n) = 3T(n/3) + n. \tag{18}$$

We obtain:

$$T(n) \in \Theta(\log_3(n)n). \tag{19}$$

A FORMULA TO ESTIMATE $T(n)$. Let $a > 0$ be an integer and let $S, T : \mathbb{N} \longrightarrow \mathbb{R}_+$ be functions such that

(*i*) $S(2\,n) \;\geq\; 2\,S(n)$ and $S(n) \;\geq\; n$.

(*ii*) If $n = 2^p$ then $T(n) \;\leq\; a\,T(n/2) + S(n)$.

Then for $n = 2^p$ we have

(1) if $a = 1$ then

$$T(n) \leq (2 - 2/n)\,S(n) + T(1) \;\in\; \mathcal{O}(S(n)), \tag{20}$$

(2) if $a = 2$ then

$$T(n) \leq S(n)\log_2(n) + T(1)\,n \;\in\; \mathcal{O}(\log_2(n)\,S(n)), \tag{21}$$

(3) if $a \geq 3$ then

$$T(n) \leq \frac{2}{a-2}\left(n^{\log_2(a)-1} - 1\right)\,S(n) + T(1)\,n^{\log_2(a)} \;\in\; \mathcal{O}(S(n)\,n^{\log_2(a)-1}). \tag{22}$$

Indeed

$$\begin{array}{rcl} T(2^p) & \leq & a\,T(2^{p-1}) + S(2^p) \\ & \leq & a\,[a\,T(2^{p-2}) + S(2^{p-1})] + S(2^p) \\ & = & a^2\,T(2^{p-2}) + a\,S(2^{p-1}) + S(2^p) \\ & \leq & a^2\,[a\,T(2^{p-3}) + S(2^{p-2})] + a\,S(2^{p-1}) + S(2^p) \\ & = & a^3\,T(2^{p-3}) + a^2\,S(2^{p-2}) + a\,S(2^{p-1}) + S(2^p) \\ & \leq & a^p\,T(1) + \Sigma_{j=0}^{j=p-1}\ a^j\,S(2^{p-j}) \end{array} \tag{23}$$

Moreover

$$\begin{array}{rcl} S(2^p) & \geq & 2\,S(2^{p-1}) \\ S(2^p) & \geq & 2^2\,S(2^{p-2}) \\ \vdots & \vdots & \vdots \\ S(2^p) & \geq & 2^j\,S(2^{p-j}) \end{array} \tag{24}$$

Thus

$$\Sigma_{j=0}^{j=p-1}\ a^j\,S(2^{p-j}) \ \leq\ S(2^p)\,\Sigma_{j=0}^{j=p-1}\ \left(\frac{a}{2}\right)^j. \tag{25}$$

Hence

$$T(2^p) \ \leq\ a^p\,T(1) + S(2^p)\,\Sigma_{j=0}^{j=p-1}\ \left(\frac{a}{2}\right)^j. \tag{26}$$

For $a = 1$ we obtain

$$\begin{array}{rcl} T(2^p) & \leq & T(1) + S(2^p)\,\Sigma_{j=0}^{j=p-1}\ \left(\frac{1}{2}\right)^j \\ & = & T(1) + S(2^p)\,\frac{\frac{1}{2^p}-1}{\frac{1}{2}-1} \\ & = & T(1) + S(n)\,(2 - 2/n). \end{array} \tag{27}$$

For $a = 2$ we obtain

$$\begin{array}{rcl} T(2^p) & \leq & 2^p\,T(1) + S(2^p)\,p \\ & = & n\,T(1) + S(n)\log_2(n). \end{array} \tag{28}$$

6 Implementing Computer Algebra: basic ideas

6.1 Which mathematical types do we need to implement?

Obviously we need to implement the various usual sets of numbers:

- the ring of INTEGER NUMBERS $\mathbb{Z}$,

- the field of RATIONAL NUMBERS $\mathbb{Q}$,
- the rings of MODULAR INTEGERS $\mathbb{Z}/n\mathbb{Z}$ (and especially those which are prime fields),
- the different sets of FLOATING POINT NUMBERS (with fixed or arbitrary precision),
- the ordered ring of REAL ALGEBRAIC NUMBERS,
- and many others that we will discover later.

Clearly we need also:

- UNIVARIATE OR MULTIVARIATE POLYNOMIALS over each of these sets of numbers,
- MATRICES over the same sets,
- and also polynomial or matrix rings over any ring we can consider, including polynomial or matrix rings themselves.

So we need infinitely many algebraic types. Let us call these types RINGS.

Moreover we need ALGEBRAIC STRUCTURES (ring, commutative ring, Euclidean domain, field, finite field, prime field, univariate polynomial ring, ...) for the following reasons.

Genericity We can organize our infinitely many RINGS in a finite number of RING types. Then, for instance, we can define the *Euclidean algorithm* once for all for every Euclidean domain. For instance, for any Euclidean domain R we need to say

```
gcd(a,b) ==
  while not zero? b repeat
     (a,b):= (b,a rem b)
  return a
```

Conditional declaration A matrix ring (or polynomial ring) may possess an operation (inverse, gcd, ...) that another does not and this fact depends on its coefficient ring. So our MATRIX RINGS and POLYNOMIAL RINGS should depend on the properties of their coefficient RING. For instance we need to say:

if R is a field then the ring $R[x]$ of univariate polynomials over R is an Euclidean domain.

Conditional implementation The implementation of an operation defined in several univariate polynomial rings like

$$(p, n) \longmapsto p^n \tag{29}$$

may depend on the properties of the coefficient RING. Let R be a ring and $p(X) = aX + b$ be a univariate polynomial in X with coefficient a, b in R. If R is a non-commutative ring then

$$(aX+b)^3 \;=\; a^3X^3+(a^2b+aba+ba^2)X^2+(abb+bba+bab)X+b^3. \tag{30}$$

If R is any commutative ring then

$$(aX + b)^3 \;=\; a^3X^3 + 3\,a^2bX^2 + 3\,ab^2X + b^3. \tag{31}$$

If R is $\mathbb{Z}/3\mathbb{Z}$ then

$$(aX + b)^3 \;=\; a^3X^3 + b^3. \tag{32}$$

Specialization Among all RINGS of a given type one may have a much better algorithm for a given RING. So for this particular one, we may wish to replace the *generic definition* of an operation. For instance for the Euclidean domain of the integers we may wish to say (with some optimizations ...)

```
gcd(a,b) ==
  while a ~= b repeat
     if a < b then b := b-a else a := a-b
  return a
```

6.2 How to encode the elements of these types?

We will discuss here the case of numbers and polynomials. In this whole section we assume that we are using a machine with N-bit words. (Typically $N = 32$ or $N = 64$.)

6.2.1 Integers.

One machine word can contain a SINGLE PRECISION INTEGER in the range $[0, 2^N-1]$. To represent integers outside of this range, so called MULTIPRECISION INTEGER, we use arrays of N-bit words. To be precise we consider the 2^N-ary (or radix 2^N) expansion of a nonzero integer:

$$a \quad = \quad (-1)^s \ \Sigma_{0\leq 1\leq n} a_i 2^{Ni} \tag{33}$$

where

- every $a_i \in [0, 2^N - 1]$ is a digit in the 2^N-ary expansion of a,
- $s \in \{0, 1\}$ determines the sign of a,
- $n+1 \in [1, 2^N]$ is the number of 2^N-digits in the 2^N-ary expansion of a,
- $a_n \neq 0$.

Then the integer a with $(n+1)$ 2^N-digits can be represented by an array (of N-bit words) with length $n+3$ since we need

- one word for s and
- one word for n

The reasonable UNIT OPERATION for integers is the the WORD OPERATION. One can easily check that

- the ADDITION of two integers with at most n 2^N-digits requires $\mathcal{O}(n)$ word operations
- the MULTIPLICATION of two integers with at most n 2^N-digits requires $\mathcal{O}(n^2)$ word operations.

6.2.2 Rational numbers.

It is natural to represent the rational number n/d as of the record of the two integers n and d such that

- $d > 0$ and
- $\gcd(n, d) = 1$.

6.2.3 Modular Integers.

There are many ways to implement the rings $\mathbb{Z}/m\mathbb{Z}$ and even more for the fields $\mathbb{Z}/p\mathbb{Z}$ (where p is a prime). Let us restrict to the fields $\mathbb{Z}/p\mathbb{Z}$. Here are some examples.

- For p in the range $[2, 2^N - 1]$ we can use single integers. Multiplication and addition can be made in $\mathcal{O}(1)$ word operations. **N.B.** multiplication requires a two single integers buffer unless p in the range $[2, 2^{N-1} - 1]$. Exponentiation and computation of inverses require a bit more of works but are still in $\mathcal{O}(1)$ word operations. For instance here's an ALDOR code for the division of two integers modulo a third one

```
mod_/(i: %, j: %, n: %): % == {
        local c0, d0, c1, d1: %;
        (c0, d0) := (j, n);
        (c1, d1) := (1, 0);
        while not zero? d0 repeat {
                q := c0 quo d0;
                (c0, d0) := (d0, c0 - q*d0);
                (c1, d1) := (d1, c1 - q*d1)
        }
        if c0 ~= 1 then error "inverse does not exist";
        if c1 < 0  then c1 := c1 + n;
        mod_*(i, c1, n);
```

- The set of the nonzero elements of $\mathbb{Z}/p\mathbb{Z}$ is a cyclic group. In other words, there exists a generator $g \in \mathbb{Z}/p\mathbb{Z}$ such that every nonzero $x \in \mathbb{Z}/p\mathbb{Z}$ is a power of g called the discrete logarithm of x. Hence we can use discrete logarithms to represent nonzero elements of $\mathbb{Z}/p\mathbb{Z}$. Then exponentiation and computation of inverses become much more easy (for inverses we use Little Fermat Theorem) than with the previous encoding. However addition requires the use of tables of logarithms and exponentiations, for efficiency. Again all arithmetic operations are in $\mathcal{O}(1)$ word operations.

6.2.4 Real algebraic numbers.

Will be discussed during later.

6.2.5 Univariate polynomials.

Let R be a ring. The univariate polynomial ring $R[x]$ can be implemented using different data types.

Expression trees. All arithmetic operations are in $\mathcal{O}(1)$. But the representation is **not canonical**: two different expression trees can encode the same polynomial. Therefore operations like DEGREE, LEADING COEFFICIENT, REDUCTUM are in $\mathcal{O}(n)$ where n is the degree of the polynomial. Even worst the equality-test is $\mathcal{O}(n^2)$ (for non-balanced trees).

Arrays of coefficients. The polynomial

$$p(x) \quad = \quad a_n x^n + \cdots + a_1 x + a_0 \tag{34}$$

is coded by a record consisting of

- a single integer s,
- a single integer $d \leq s + 1$,
- an array of size s such that $a_0 + ... + a_n x^n$ is represented by $[a_0, ..., a_n, ...]$ and $d = n$.

This representation is said *dense* becasue all a_i are coded, even those which are null. This representation is canonical. Hence operations like DEGREE, LEADING COEFFICIENT, REDUCTUM are in $\mathcal{O}(1)$. Addition and equality-test are in $\mathcal{O}(n)$ and multiplication is in $\mathcal{O}(n^2)$. This representation is especially good when the ring of coefficients is a small prime field, i.e. $\mathbb{Z}/p\mathbb{Z}$ with p prime and in the range $[2, 2^N - 1]$.

List of terms. The polynomial

$$p(x) \quad = \quad a_n x^n + \cdots + a_1 x + a_0 \tag{35}$$

is coded by the list L of records $[a_i, i]$ where a_i is a nonzero coefficient and such that L is sorted decreasingly w.r.t. i. This representation is said *sparse*, since only the nonzero a_i are coded. This representation is also canonical. Hence operations like DEGREE, LEADING COEFFICIENT, REDUCTUM are in $\mathcal{O}(1)$. Moreover the operation REDUCTUM does not require coefficient duplication (on the contrary of the previous representation). Addition and equality-test are in $\mathcal{O}(n)$ and multiplication is in $\mathcal{O}(n^2)$. This representation is especially good when the ring of coefficients is itself a ring of sparse polynomials.

6.2.6 Multivariate polynomials.

Let R be a ring and $X = \{x_1, \ldots, x_n\}$ be a finite set of variables. The multivariate polynomial ring $R[X]$ can be implemented in different ways.
Recursively.

- If $n = 1$ we can use a univariate representation
- otherwise we can view $R[X]$ as a univariate polynomial ring with a multivariate polynomial ring as coefficient ring, say for instance $R[x_1, \ldots, x_{n-1}][x_n]$.

This implies to choose an ordering on the variables and a representation for univariate polynomials.

As linear combinations of monomials.

- By *monomial* we mean any product of variables.
- The product of variable is assumed to be commutative and induces a commutative multiplication for monomials.
- Each polynomial can be viewed as a linear combination of monomials (with coefficients in R).
- Then the polynomial

$$p \;=\; a_1 m_1 + \cdots + a_p m_p. \tag{36}$$

  where the m_i are pairwise different monomials and the a_i are nonzero coefficients, can be represented as an aggregate of terms $[a_i, m_i]$.

- This provides us with a canonical representation.
- To have a fast equality-test, the aggregate of terms must be *linear*. In other words, there should be a *first* term, a *second* term, ... Thus this aggregate should better be a *list* or an *array* and the monomials should be totally ordered.
- Two types of monomial orderings are frequently used.
  - The lexicographical ordering. With $X = \{x > y > z\}$ we have

$$1 < z < \cdots < z^n \cdots < y < yz < \cdots < yz^n \cdots < y^2 < y^2 z < \cdots < y^2 z^n \cdots \tag{37}$$

– The degree-lexicographical ordering. With $X = \{x > y > z\}$ we have

$$1 < z < y < x < z^2 < zy < y^2 < zx < xy < x^2 < \cdots < \quad (38)$$

6.3 What are our requirements for the language?

6.3.1 Bounded genericity with dependent types

Recall that the RINGS R in Computer Algebra are obtained by applying rules like

(1) $R = \mathbb{Z}$,

(2) $n \longmapsto \mathbb{Z}/n\mathbb{Z}$,

(3) $R \longmapsto R[X]$,

(4) $(R, \mathcal{I}) \longmapsto R/\mathcal{I}$

(5) $R \longmapsto \mathbf{Fr}(R)$,

(6) $(R_1, R_2) \longmapsto R_1 \times R_2$.

To implement the above rules we would like to do the following constructions in our programming language.

- To define functions which return a type (as in rule **(2)**).
- To define functions that can be parametrized by a type (as in rule **(3)**). Hence we need GENERICITY.
- To handle INTERFACE (or type of types). Indeed in **(3)** we need R to be a ring. Hence we need BOUNDED GENERICITY.
- In rule **(4)** we need to say that $\mathcal{I}$ is an ideal of R. Hence we need BOUNDED GENERICITY WITH DEPENDENT TYPES.
- We should be able to have functions as parameters of other functions. For instance an ordered abelian free monoid genetated by a set S depends on a total ordering on S. Hence functions must have types too. Finally, both functions and types must have types and must be possible parameters for functions. Hence functions and types must be TREATED AS VALUES.

6.3.2 Inheritance

Recall that we need to implement algebraic structures like: ring, commutative ring, euclidean domain, field, ... Moreover we want to say that *an euclidean domain is a commutative ring.* Hence we want INHERITANCE BETWEEN INTERFACES. Of course we need also INHERITANCE BETWEEN CONCRETE CLASSES. For instance:

- A prime finite field $\mathbb{Z}/p\mathbb{Z}$ is a particular ring of modular integers (whose characteristic is a prime). Hence its implementation should inherit from that of the modular integer ring $\mathbb{Z}/m\mathbb{Z}$ (where m is not necessarily a prime).
- A multivariate polynomial ring $R[x_1, x_2, \ldots, x_n]$ is a particular univariate polynomial ring $(R[x_1, x_2, \ldots, x_{n-1}])[x_n)$. Hence its implementation should be able to reuse that of a *generic* univariate polynomial ring.

6.3.3 Post-facto extensions

We need a mechanism to enhance a given type. For instance,

- to define the *monoid* of non-negative integers we need the type MONOID and
- to define the exponentiation $(p, n) \longmapsto p^n$ when defining monoids, we need the set of non-negative integers to be defined already.

The solution is to be able

1. to define first the set of non-negative integers without viewing it as a MONOID,
2. then to define the interface MONOID
3. and finally to enhance the concrete class of non-negative integers by adding to it the property of being a MONOID.

6.3.4 Efficiency

In any MONOID exponentiation can de defined by the binary powering algorithm

```
(^)(x: %, n: NonNegativeInteger): % ==
        (n < 0) =>
                error "power:  negative exponent";
        n = 0 => 1;
        n = 1 => x;
        n = 2 => x*x;

        -- General code.
        i: SingleInteger := 0;
        l: SingleInteger := length n;
        u: % := 1;
        repeat {
                if bit(n, i) then u := u * x;
                if i >= l then break;
                x := x * x;
                i := i + 1;
        }
        u
}
```

Assume that in the interface MONOID the exponentiation has been defined as above. In the MONOID $\mathbb{Z}/p\mathbb{Z}$ (where p is a prime) we can do better because of little Fermat theorem which states that for $x \neq 0$ we have $x^{p-1} = 1$. Hence we need to overwrite the above definition for $\mathbb{Z}/p\mathbb{Z}$ by something like

```
(^)(x: %, n: NonNegativeInteger): % == {
    n := positiveRemainder(n,p);
    zero? n => 1;
    one? x => x;
    local acc: %;
    if odd? n then acc := x else acc := 1;
    repeat {
        n := shift(n,-1);
        zero? n => return acc;
        x := (x * x) mod p;
        if odd? n then acc := (acc * x) mod p;
    }
}
```

This will be made possible by INHERITANCE. In a polynomial ring $R[x]$ one can do better too in some special cases depending on the coefficient ring, as we have seen before. This will be made possible by CONDITIONAL IMPLEMENTATION.

7 Asymptotically fast algorithms

7.1 The Karatsuba trick

Let f, g be two univariate polynomials in x with degrees p and q respectively. Let $\mathbb{R}$ be their ring of coefficients. Define

$$f(x) = a_p x^p + \cdots + a_1 x + a_0 \tag{39}$$

and

$$g(x) = b_q x^q + \cdots + b_1 x + b_0. \tag{40}$$

COMPUTING THE PRODUCT $f(x)g(x)$ by the straightforward method requires

- $(p+1)(q+1)$ multiplications in $\mathbb{R}$ and
- pq additions in $\mathbb{R}$.

So the complexity of this computation is $\mathcal{O}(pq)$ (in term of operations in $\mathbb{R}$).

Thus the complexity of the computation of the product of two univariate polynomials of degree at most n is $\mathcal{O}(n^2)$ (in term of operations in $\mathbb{R}$).
COMPUTING THE SUM $f(x) + g(x)$ is $\mathcal{O}(n)$ (in term of operations in $\mathbb{R}$) if f and g have of degree at most n.

ADDING IS OFTEN CHEAPER THAN MULTIPLYING. So adding polynomials is cheaper than multiplying them. For many rings $\mathbb{R}$ addition is also cheaper than multiplication. So when computing $f(x)g(x)$ let's try to save on the number of multiplications in $\mathbb{R}$.

THE KARATSUBA'S TRICK. Assume f and g have degree (strictly) less than $n = 2^k$ (where k is an integer). The Karatsuba's trick computes the product $f(x)g(x)$ as follows.

(1) If $n = 1$ then compute the element $f\,g$ of $\mathbb{R}$.

(2) Define $f = F_1\, x^{n/2} + F_0$ and $g = G_1\, x^{n/2} + G_0$ where F_1, F_0, G_1, G_0 have degree $< n/2$.

(3) Compute $F_0\, G_0$, $F_1\, G_1$, $(F_0 + F_1)(G_0 + G_1)$ recursively

(4) Return

$$F_1 G_1\, x^n + ((F_0 + F_1)(G_0 + G_1) - F_0\, G_0 - F_1\, G_1)\, x^{n/2} + F_0\, G_0. \quad (41)$$

ITS COMPLEXITY. Let us count how many operations in $\mathbb{R}$ we perform with the Karatsuba's trick.

- At step (3) we perform at most n additions to compute $(F_0 + F_1)$ and $(G_0 + G_1)$.
- At step (4) we perform at most $2\, n$ subtractions to compute $(F_0 + F_1)(G_0 + G_1) - F_0\, G_0 - F_1\, G_1$
- At step (5) we perform at most n additions to add this latter polynomial to $F_1 G_1\, x^n + F_0\, G_0$.

Hence we can apply the previous formula (for estimating $T(n)$) with

$$\left\{ \begin{array}{rcl} a & = & 3 \\ T(1) & = & 1 \\ S(n) & = & 4\, n \end{array} \right. \quad (42)$$

We obtain

$$T(n) \in \mathcal{O}(4\, n\, n^{\log_2(3)-1}). \quad (43)$$

that is

$$T(n) \in \mathcal{O}(n^{\log_2(3)}). \quad (44)$$

Observe that:

$$\log_2(3) \leq 1,59. \quad (45)$$

7.1.1 Experimentation

See Project 1 of Winter 2005.

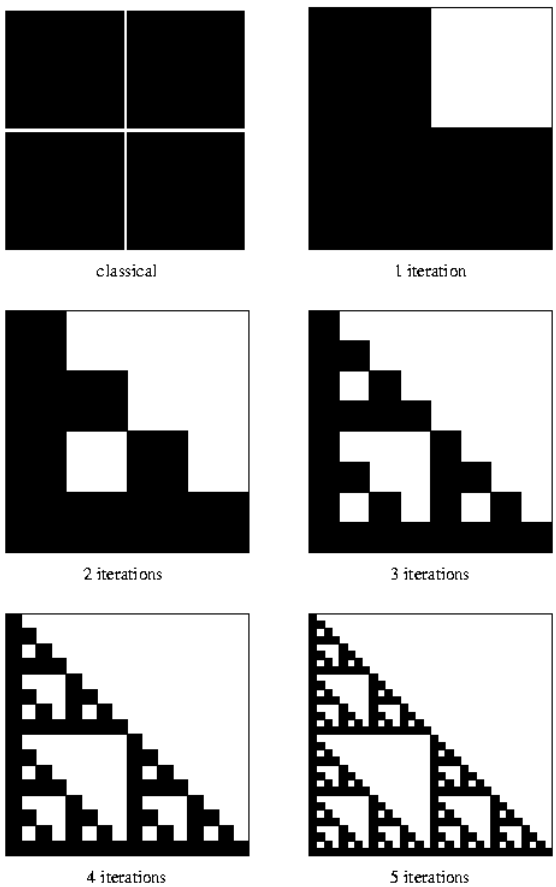

Figure 5: Complexity of the Karatsuba algorithm.

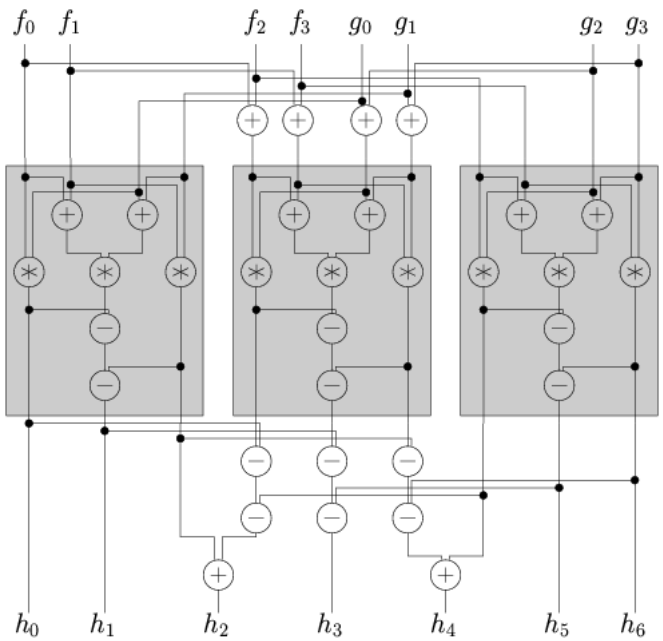

Figure 6: An arithmetic circuit for the Karatsuba algorithm.

8 Acknowledgments

Figures 5, 6 appear in the the book *Modern Computer Algebra* by J. von zur Gathen and Jürgen Gerhard (published by Cambridge University Press in 1999) and can be downloaded from the book's web site:
`http://www-math.uni-paderborn.de/mca`.

References

[ald04] aldor.org. *Aldor 1.0.2*. University of Western Ontario, Canada, 2004. http://www.aldor.org.

[Ber67] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46:1853–1859, 1967.

[Buc65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[CG86] S.R. Czapor and K.O. Geddes. On implementing Buchberger's algorithm for Gröbner bases. In *Proc. SYMSAC'86*, pages 425–440. Waterloo, B.W. Char, 1986.

[DST93] J. Davenport, Y. Siret, and E. Tournier. *Calcul Formel*. Masson, 1993. 2^e édition révisée.

[EP99] Ioannis Z. Emiris and Victor Y. Pan. Fast fourier transform and its applications. In Mikhail J. Atallah, editor, *Handbook of Algorithms and Theory of Computations*. CRC Press Inc, 1999.

[Esc01] Jean-Pierre Escofier. *Galois Theory*. Springer, 2001. Translated from the French by Leila Schneps.

[GG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[htt06] http://www.shoup.net/ntl. *The Number Theory Library*. V. Shoup, 1996–2006.

[JS92] R. D. Jenks and R. S. Sutor. *AXIOM, The Scientific Computation System*. Springer-Verlag, 1992. AXIOM is a trade mark of NAG Ltd, Oxford UK.

[KO63] A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, (7):595–596, 1963.

[Kro97a] L. Kronecker. Die Zerlegung der ganzen Grössen eines natürlichen Rationalitäts-Bereichs in ihre irreductibeln Factoren. *J. für die Reine und Angewandte Mathematik*, 94:344–348, 1897.

[Kro97b] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *J. für die Reine und Angewandte Mathematik*, 92:1–122, 1897.

[LHWLL82] A. K. Lenstra, Jr. H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen.*, 261:515–534, 1982.

[Li05] X. Li. Efficient management of symbolic computations with polynomials, 2005. University of Western Ontario.

[Map05] Maplesoft. *Maple 10*. http://www.maplesoft.com/, 2005.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik.*, 13:354–356, 1969.

[Yap93] C.K. Yap. *Fundamental Problems in Algorithmic Algebra*. Princeton University Press, 1993.

[Zas69] Zassenhaus, H. On Hensel factorization I. *Journal of Number Theory*, 1:291–311, 1969.

[Zip93] R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Press, 1993.