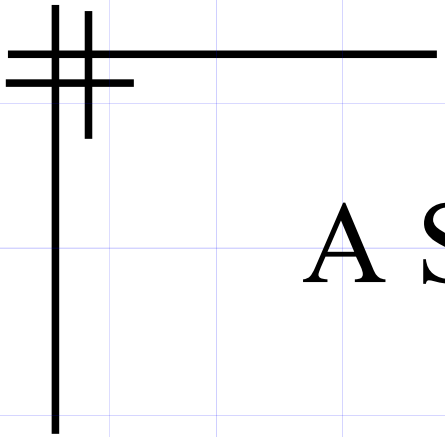


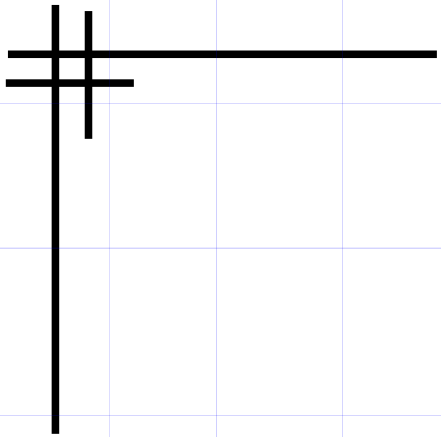
A Systems Perspective on A3L

Heinz Kredel

University of Mannheim

Algorithmic Algebra and Logic 2005

Passau, 3.-6. April 2005



Introduction

- Summarize some aspects of the development of computer algebra systems in the last 25 years.
- Focus on Aldes/SAC-2, MAS and some new developments in Java.
- Computer algebra can more and more use standard software developed in computer science to reach its goals.
- In CA systems theories of Volker Weispfenning have been implemented to varying degrees.

Relation to Volker Weispfenning

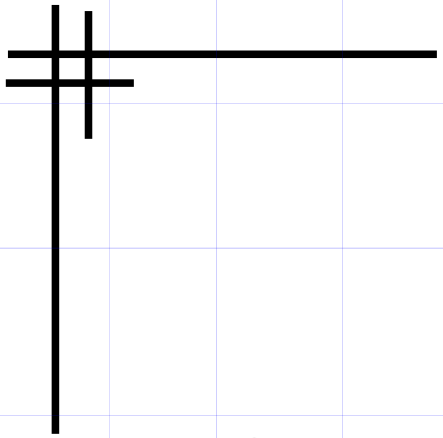
- Determine the dimension of a polynomial ideal by inspection of the head terms of the polynomials in the Gröbner base.
- Constructing software for the representation and computation of Algebraic Algorithms and Logic
- Covering Aldes/SAC-2, time in Passau using Modula-2 and today Java.

ALDES / SAC-2

- Major task was the implementation of Comprehensive Gröbner Bases in DIP by Elke Schönfeld
- Distributive Polynomial System (DIP) with R. Gebauer
- Aldes/SAC-2 developed by G. Collins and R. Loos
- Algebraic Description Language (Aldes) translator to FORTRAN
- SAC-2 run time system for list processing with automatic garbage collection

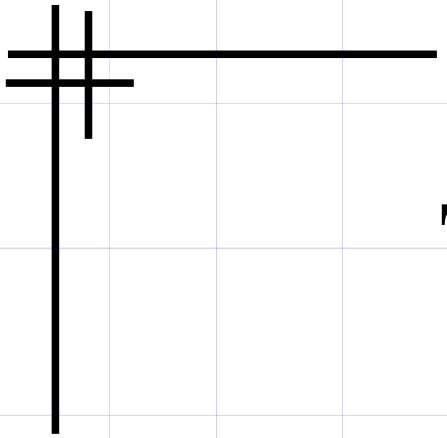
CAD

- Aldes/SAC-2 orginated in SAC-1, a pure FORTRAN implementation with a reference count garbage collecting list processing system
- Cylindrical algebraic decomposition (CAD) by G. Collins
- Quantifier elimination for real closed fields
- Provided a comprehensive library of fast and reliable algebraic algorithms
- integers, polynomials, resultants, factorization, algebraic numbers, real roots



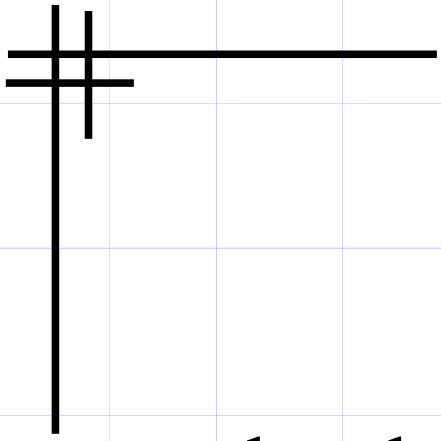
Gröbner bases

- One of the first Buchberger algorithms in Aldes/SAC-2
- not restricted, no static bounds
- Used for zero-dimensional primary ideal decomposition
- and real roots of zero-dimensional ideals



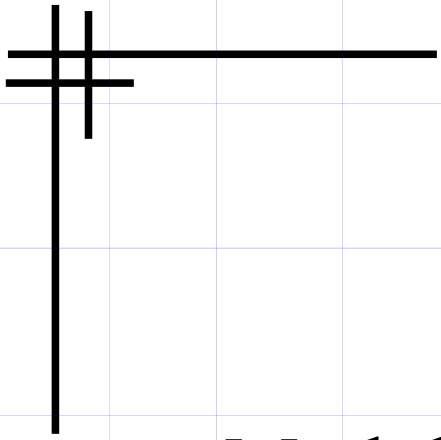

Time of micro computers

- up to then mainframe based development environments
- wanted modern interactive development environment like Turbo-Pascal
- tried several Pascal compilers
- but no way to implement a suitable list processing system
- things getting better with Modula-2




Modula-2

- development of run time support for a list processing system with automatic garbage collection
- Boehm: garbage collector in an uncooperative environment in C
- bootstrapping translator to Modula-2 within the Aldes/SAC-2 system
- all of the existing Aldes algorithms (one exception) were transformed to Modula-2
- called Modula Algebra System (MAS)




Interpreter

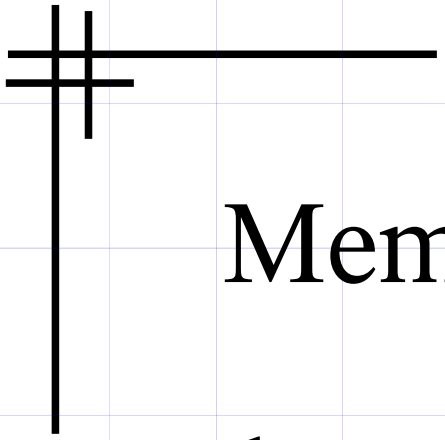
- Modula-2 procedure parameters
- interpreted language similar to Modula-2
- release 0.30, November 1989
- language extensions as in algebraic specification languages (ASL)
- term rewriting systems, Prolog like resolution calculus
- interfacing to numerical (Modula-2) libraries
- (Python in 1990)

MAS content (1)

- implementation of theories of V. Weispfenning:
- real quantifier elimination (Dolzmann)
- comprehensive Gröbner bases (Schönfeld, Pesch)
- universal Gröbner bases (Belkahia)
- solvable polynomial rings
- skew polynomial rings (Pesch)
- real root counting using Hermites method (Lippold)

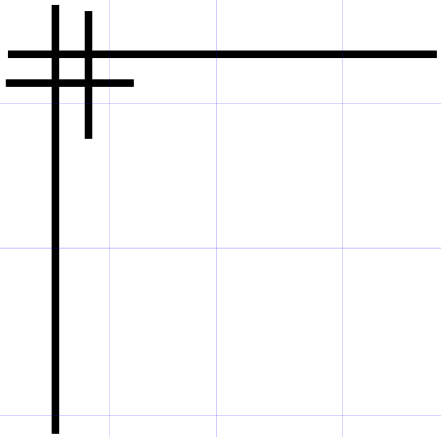
MAS content (2)

- other implemented theories:
- permutation invariant polynomials (Göbel)
- factorized, optimized Gröbner Bases (Pfeil, Rose)
- involutive bases (Grosse-Gehling)
- syzygies and module Gröbner bases (Phillip)
- d- and e-Gröbner bases (Becker, Mark)




Memory caching micro processors

- dramatic speed differences between cache and main memory
- concequences for long running computations:
- the list elements of algebraic data structures tend to be scattered randomly throughout main memory
- thus leading to cache misses and CPU stalls at every tiny step
- Other systems replace integer arithmetic with libraries like Gnu-MP

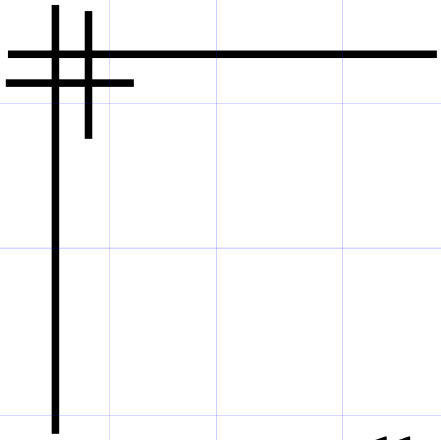



MAS problems (1)

- no transparent way of replacing integer arithmetic in MAS
- due to the ingenious and elegant way G. Collins represented integers
- small integers (< 2^29 = beta) are represented as 32-bit integers
- large integers (>=beta) are transformed to lists
- code full of case distinctions 'IF i < beta THEN'
- distinction between BETA and SIL, but LIST as alias of LONGINT

MAS problems (2)

- Integer and recursive polynomials are not implemented as proper datatype (as defined in computer science)
- zero elements of algebraic structures as integer '0'
- this avoided constructors and eliminated problems with uniqueness but lost all structural information




MAS parallel computing

- parallel computers (32 – 128 CPUs) in Mannheim
- parallel garbage collector and parallel list processing subsystem using POSIX threads
- parallel version of Buchberger's algorithm
- pipelined version of the polynomial reduction algorithm
- but no reliable speedup on many processors
- version was not released due to tight integration with KSR hardware

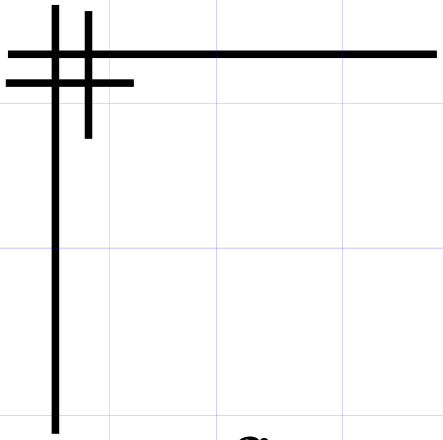
Problems

1. respect and exploit the memory hierarchy
2. find good load balancing and task granularity
3. find a portable way of parallel software development

- for basic building blocks of a system
- for implementation of each algorithm

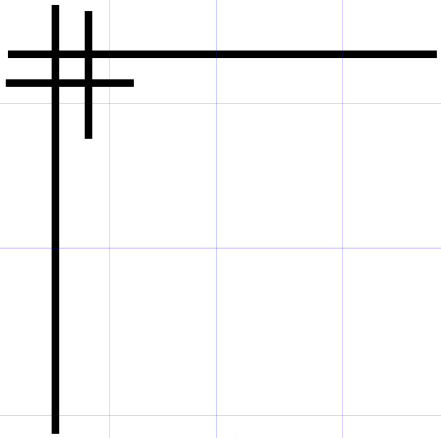
Alternatives

- developments of languages of N. Wirth, Modula-2 and Oberon was not as expected
- others used C language for the implementation
  - like H. Hong with SACLIB
  - W. Küchlin with PARSAC
- others used C++ for algebraic software
  - like LiDIA from T. Papanikolaou
  - like Singular of H. Schönemann
- others turned to commercial systems like Maple, Mathematica




Java

- first use for parallel software development
- got confident in the performance of Java implementations
- and object oriented software development
- in 2000: Modula-2 to Java translator
- first atempt with old style list processing directly ported to Java
- about 5-8 times slower on Trinks6 Gröbner base than MAS

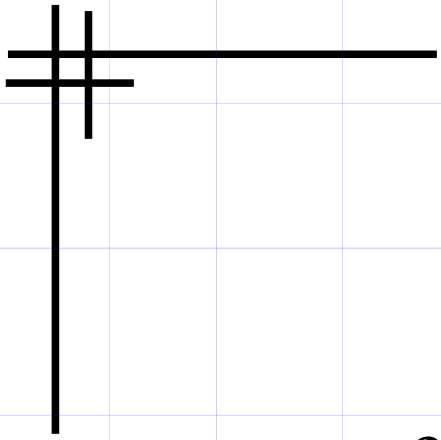



Basic refactoring

- integer arithmetic with Java's BigInteger class showed an improvement by a factor of 10-15 for Java
- so all list processing code had to be abandoned and native Java data structures should be used
- Polynomials were reimplemented using java.util.TreeMap
- now polynomials are, as in theory, a map from a monoid to a coefficient ring
- factor of 8-9 better on Trinks6 Gröbner base

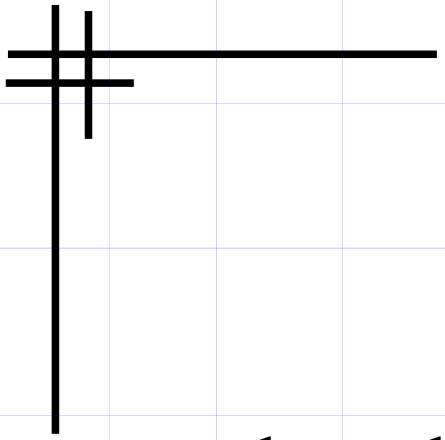
OO and Polynomial complexity

- Unordered versus ordered polynomials
- LinkedHashMap versus TreeMap (10 x faster)
- sum of a and b, l(a) = length(a):
  - Hash: 2*(l(a)+l(b))
  - Tree: 2*l(a)+l(b)+l(b)*log2(l(a+b))
- product of a and b: coefficients: lab = l(a)*l(b) :
  - Hash: plus 2*l(a*b)*l(b)
  - Tree: plus l(a)*l(b)*log2(l(a*b))
  - sparse pol: TreeMap better, dense: HashMap better
  - sparse l(a*b) ~ lab, dense l(a*b) ~ l(a)[+l(b)]



Developments

- use of more and more object oriented principles
- shared memory and a distributed memory parallel version for the computation of Gröbner bases
- solvable polynomial rings
- modules over polynomial rings and syzygies
- Unit-Tests for most Classes with Junit
- Logging with Apache log4j
- Python / Jython interpreter frontend

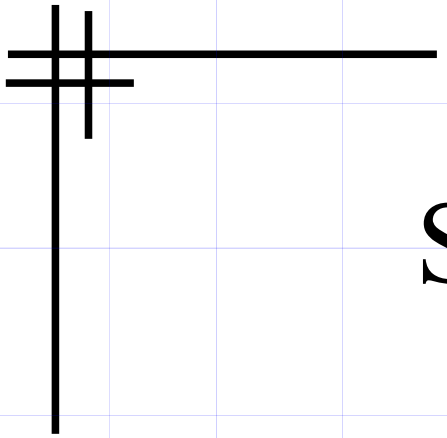



Parallel Gröbner bases

- shared memory implementation with Threads
- reductions of S-polynomials in parallel
- uses a critical pair scheduler as work-queue
- scalability is perfect up to 8 CPUs on shared memory
- provided the JVM uses the parallel Garbage Collector and aggressive memory management
- correct JVM parameters essential

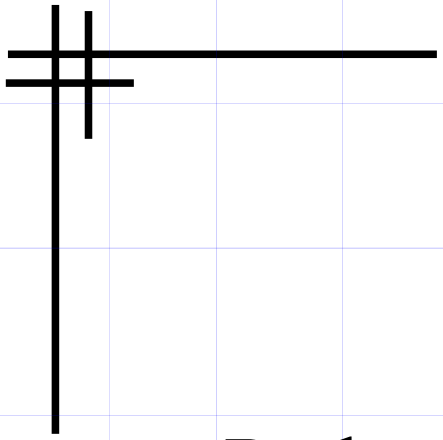
Distributed Gröbner bases

- distributed memory implementation using TCP/IP Sockets and Object serialization
- reduction of S-polynomials on distributed computing nodes
- uses the same (central) critical pair scheduler as in parallel case
- distributed hash table for the polynomials in the ideal base with central index managing
- communication of polynomials is easily done using Java's object serialization capabilities



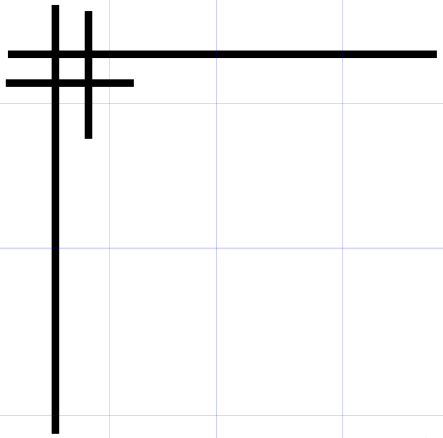
Solvable polynomial rings

- new relation table implementation
- extend commutative polynomials




Jython

- Python interpreter in Java
- full access to all Java classes and libraries
- some syntactic sugar in jas.py

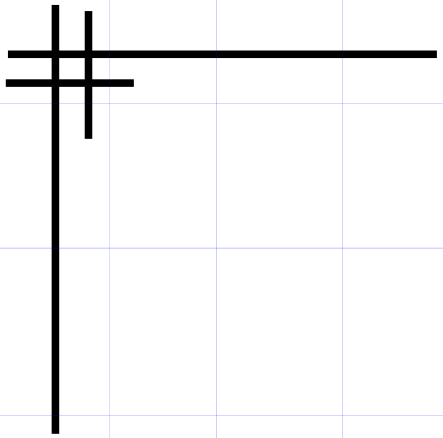



ToDo

- generics coming in with JDK 1.5
- Cilk algorithms in java.util.concurent
- three (orthogonal) axis:
  - parallel and distributed algorithms
  - commutative polynomial rings
  - solvable polynomial rings

Conclusions (1)

- Not all mathematically ingenious solutions like the small integer case can persist in software development.
- A growing part of software need no more be developed specially for CA systems but can be taken from libraries developed elsewhere by computer science
- e.g. STL for C++ or java.util




Conclusions (2)

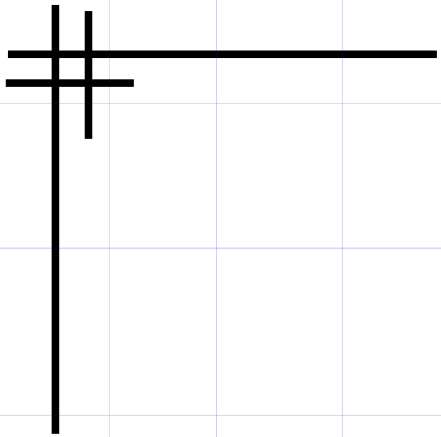
- programming language features needed in CAS
  - dynamic memory management with garbage collection,
  - object orientation (including modularization)
  - generic data types
  - concurrent and distributed programming
- are now included in languages like Java (or C#)

Conclusions (3)

- In the beginning of CA systems development only a small part was taken from computer science (namely FORTRAN).
  - 10% computer science in CAS
- Then a bigger part in Modula-2 or C++ based systems was employed.
  - 30% computer science in CAS
- Today more than the half part (Java) can be used from the work of software engineers
  - 60% computer science in CAS

Conclusions (4)

- go and use the improvements of computer science and systems engineering for implementation of A3L algorithms
- But don't forget to observe and adapt to hardware developments:
  - memory hierarchy
  - multi-core CPUs
  - distributed systems




Thank you

- Questions?
- Comments?
- http://krum.rz.uni-mannheim.de/jas
- Thanks to
  - Volker Weispfenning
  - Thomas Becker, Michael Pesch
  - Andreas Dolzmann, Thomas Sturm, Manfred Göbel
  - all others