Fast Algorithms, Modular Methods, Parallel Approaches and Software Engineering for Solving Polynomial Systems Symbolically

PhD Thesis Presentation by

Yuzhen Xie

Supervised by

Dr. Marc Moreno Maza

and Dr. Stephen M. Watt

Computer Science Department

University of Western Ontario, Canada

September 4, 2007

To improve the performance of symbolic solvers

- **Applications of symbolic solvers:**
  - cryptology, robotics, geometric modeling, dynamical systems, ...

- **Difficulties in solving polynomial systems symbolically:**
  - many ways of solving, especially with infinitely many solutions,
  - exponential space complexity algorithms,
  - computer resource consuming and highly complex software.

- **Our goal:**

  - to increase the range of practically solvable problems,
  - by improving algorithms, developing parallel implementations, and providing validation tools.

An example of symbolic solving

$$\text{Given } F \subset \mathbb{Q}[x,y,z]: \begin{cases} x^2+y+z=1 \\ x+y^2+z=1 \\ x+y+z^2=1 \end{cases}$$

$$F \text{ has } \underline{\textbf{Gröbner basis}}: \begin{cases} z^6-4z^4+4z^3-z^2=0 \\ 2z^2y+z^4-z^2=0 \\ y^2-y-z^2+z=0 \\ x+y+z^2-1=0 \end{cases}$$

and **triangular decomposition** :

$$\begin{cases} z=1 \\ y=0 \\ x=0 \end{cases} \bigcup \begin{cases} z=0 \\ y=1 \\ x=0 \end{cases} \bigcup \begin{cases} z=0 \\ y=0 \\ x=1 \end{cases} \bigcup \begin{cases} z^2+2z-1=0 \\ y=z \\ x=z \end{cases}$$

Solving equations: successes and challenges

• **Highly efficient algorithms and implementations for linear systems and univariate polynomials:**

- `LinBox`, `NTL`, ...
- **Main ideas:**
  - modular methods,
  - asymptotically fast algorithms,
  - efficient management of computer resources (CPU, memory).

• **Challenges for solving non-linear polynomial systems:**

- an n-variate system with total degree d can have d^n solutions,
- even if the output is small, intermediate expressions can be huge,
- the solution set may contain components of different nature,
- the idea of substitution is more complicated than in the linear case.

Our focus: Triangular Decomposition (Tr.D.)

- Let $F \subset \mathbb{K}[x_1 < \cdots < x_n]$ and $V(F) = \{z \in \overline{\mathbb{K}}^n \mid (\forall g \in F)\ g(z) = 0\}$.

- The zero set $V(F)$ admits a decomposition $V(F) = V(S_1) \cup \cdots \cup V(S_e)$ s.t. $S_1, \ldots, S_e \subset \mathbb{K}[X]$ and every $V(S_i)$ **cannot** be decomposed further.

- Up to technical details, each $V(S_i)$ is the zero set of a **triangular set** (more precisely a regular chain), as a **solved system:**

$$\begin{cases} T_n(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{n-1}, x_n) = 0 \\ T_{n-1}(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{n-1}) = 0 \\ \vdots \\ T_{d+2}(x_1, \ldots, x_d, x_{d+1}, x_{d+2}) = 0 \\ T_{d+1}(x_1, \ldots, x_d, x_{d+1}) = 0 \\ h(x_1, \ldots, x_d) \neq 0 \end{cases}$$

General algorithms for triangular decompositions

Method	**Principle**	**Inconsistent components**	**Redundant components**	**Other properties**
W.T. Wu (1987)	Elimination	Yes	Yes	No use of D5 Principle
M. Kalkbrener (1991)	Elimination	No	Yes	"generic" solutions only
D. Lazard (1991)	Incremental	No	?	Practically inefficient
D.M. Wang (1993)	Elimination	No	No	More inequalities than other mthds
MMM (Triade, 2000)	By decreasing dimension + Lazy evaluation	No	No	Better control of intermediate computations

Limitations of Tr.D. solvers before our work

- Solvers with limited performances:
  - often quite inefficient when $V(F)$ is finite,
  - no modular algorithms for computing Tr.D.,
  - lack of implementation techniques.
- No HPC solvers:
  - no use of fast arithmetic,
  - no parallel implementation of "modern algorithms".
- Specification and verification issues:
  - no clear specifications for automatic case discussion,
  - no efficient ways to remove redundant components,
  - no verification tools.

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set:

  - our results: a complexity study of the celebrated *D5 Principle.*

- Equiprojectable decomposition and a first modular algorithm for Tr.D.

- A first coarse-grained parallelization of Tr.D.

- A framework for high-performance computer algebra

- Irredundant and verified triangular decompositions

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set

- Equiprojectable decomposition and a first modular algorithm for Tr.D.

  - greatly improves the solving of systems with finitely many solutions,
  - helps implementing the *D5 principle*.

- A first coarse-grained parallelization of Tr.D.

- A framework for high-performance computer algebra

- Irredundant and verified triangular decompositions

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set

- Equiprojectable decomposition and a first modular algorithm for Tr.D.:

- A first coarse-grained parallelization of Tr.D.

  - combining the geometrical information gathered by Triade and,
  - the above modular method.

- A framework for high-performance computer algebra:

- Irredundant and verified triangular decompositions

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set

- Equiprojectable decompositions and a first modular algorithm for Tr.D.

- A first coarse-grained parallelization of Tr.D.

- A framework for high-performance computer algebra:

  - developed in the categorical programming language Aldor,
  - for supporting our parallel solver.

- Irredundant and verified triangular decompositions

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set
- Equiprojectable decomposition and a first modular algorithm for Tr.D.
- A first coarse-grained parallelization of Tr.D.
- A framework for high-performance computer algebra
- Irredundant and verified triangular decompositions

  - provide an efficient algorithm for removing redundant components,
  - can verify our Aldor solver with our Maple solver.

Overview of this thesis work

- Fast polynomial arithmetic for computing modulo a triangular set
- Equiprojectable decomposition and a first modular algorithm for Tr.D.
- A first coarse-grained parallelization of Tr.D.
- A framework for high-performance computer algebra
- Irredundant and verified triangular decompositions

Our results: new algebraic methods, new complexity results, new implementation framework, and new software tools.

Fast polynomial arithmetic over DPFs: motivation

- **Base case of DPFs:**
  - $\mathbb{L} = \mathbb{K}[x]/\langle p \rangle$, an extension of a field $\mathbb{K}$, with p square-free.
- **D5 Principle:**
  - Let $q \in \mathbb{K}[x]$ and $g := \gcd(p, q)$. We have: $q \begin{cases} \equiv 0 & \mod g \\ \text{invertible} & \mod p/g \end{cases}$
  - This is a quasi-inverse computation, complexity easy to analyze.
  - Up to splitting, it can be done as if $\mathbb{L}$ were a field.
- **Case of a DPFs given by a triangular set:**
  - Let $T = T_1(x_1), T_2(x_1, x_2), \ldots, T_n(x_1, x_2, \ldots, x_n)$ be a triangular set, generating a radical ideal. Then $\mathbb{L}_n = \mathbb{K}[X]/\langle T \rangle$ is again a DPFs.

How to achieve fast polynomial arithmetic over $\mathbb{L}_n$?
Complexity of D5 principle in this case?

Fast polynomial arithmetic over DPFs: difficulties

• Using the D5 Principle over $\mathbb{L}_n$ leads to splitting, *i.e.* replacing the triangular set T by a triangular decomposition $\Delta = T^1, \ldots, T^e$.

• Then, we have to evaluate fast the map for projecting a polynomial p.

PROJECT: $\mathbb{L}_n = \mathbb{K}[X]/\langle T \rangle \rightarrow \mathbb{K}[X]/\langle T^1 \rangle \times \ldots \times \mathbb{K}[X]/\langle T^e \rangle$

• **Example:** $T = \left\{ \begin{array}{r} (x_2-1)(x_2-2) \\ (x_1-1)(x_1-2)(x_1-3) \end{array} \right. \Rightarrow$

$$\left\{ \begin{array}{r} x_2 - 1 \\ (x_1-1)(x_1-2) \end{array} \right. \bigcup \left\{ \begin{array}{l} x_2 - 1 \\ x_1 - 3 \end{array} \right. \bigcup \left\{ \begin{array}{r} x_2 - 2 \\ (x_1-1)(x_1-3) \end{array} \right. \bigcup \left\{ \begin{array}{l} x_2 - 2 \\ x_1 - 2 \end{array} \right.$$

Project p:

$p \mapsto p \mod (x_1-1)(x_1-2),\ p \mod (x_1-3),\ p \mod (x_1-1)(x_1-3),$ and $p \mod (x_1-2)$.

• Redundant moduli !

Fast polynomial arithmetic over DPFs: our solutions

• Adapt subproduct tree, Half-GCD and coprime factorization over fields to DPFs

• **Coprime factorization:**

$$\begin{array}{ll}(x_1-1)(x_1-2), & (x_1-3),\\ (x_1-1)(x_1-3), & (x_1-1).\end{array} \longmapsto (x_1-1),(x_1-2),(x_1-3)$$

• **Remove redundant moduli:**

$$T \Rightarrow \left\{\begin{array}{c} x_2-1\\ (x_1-1)\end{array}\right. \cup \left\{\begin{array}{c} x_2-1\\ (x_1-2)\end{array}\right. \cup \left\{\begin{array}{c} x_2-1\\ (x_1-3)\end{array}\right. \cup \left\{\begin{array}{c} x_2-2\\ (x_1-1)\end{array}\right. \cup$$

$$\left\{\begin{array}{c} x_2-2\\ (x_1-2)\end{array}\right. \cup \left\{\begin{array}{c} x_2-2\\ (x_1-3)\end{array}\right.$$

No redundant moduli !

• Project fast: $p \mapsto p \mod (x_1-1),\ p \mod (x_1-2),\ p \mod (x_1-3)$

⇒ Fast multiplication, quasi-inverse and GCD over DPFs !

Fast polynomial arithmetic over DPFs: main results

- Complexity measures used: $T = (T_1, \ldots, T_n)$ and $d_i := \deg_{x_i}(T_i)$. $\mathbb{M}(d)$ is an upper bound for the cost of the multiplication of two polynomials of degree at most d. $\mathbb{M}(d) \in O(d \log(d) \log\log(d))$ (Cantor-Kaltofen, 1991)

Theorem 1 (Dahan-Moreno Maza-Schost-Xie, 2006)
There exists a constant $\mathsf{C} > 0$, such that the addition, multiplication, and quasi-inverses in $\mathbb{L}_n$ can be done in $\mathsf{C}^n \prod_{i=1}^{n} \mathbb{M}(d_i) \log^3(d_i)$, $(\text{quasi} - \text{linear in } d_1 \ldots d_n)$ operations in $\mathbb{K}$.

Theorem 2 (Dahan-Moreno Maza-Schost-Xie, 2006)
There exists a constant $\mathsf{C} > 0$, such that the GCD of two polynomials in $\mathbb{L}_n[x_{n+1}]$ of degree d can be done in $\mathsf{C}^{n+1} \mathbb{M}(d) \log(d) \prod_{i=1}^{n} \mathbb{M}(d_i) \log^3(d_i)$ operations in $\mathbb{K}$.

D5 principle: related work

Della-Dora, Dicreszenzo, Duval (**1985**), D5 principle, but no complexity estimate.

Kaltofen (**1985**), fast parallel absolute irreducibility testing.

Langemyr (**1991**), Complexity results for GCD's over products of fields are used, but with no proof.

Moreno Maza and Rioboo (**1995**) introduced polynomial GCDs over DPFs and quasi-inverses.

Our algorithms use Subproduct-tree techniques, which were introduced in the **1970's** (Fiduccia, Borodin-Moenck, Strassen), and Half-GCD by Yap, ...

Coprime factorization is also known as gcd-free basis computation. Bernstein (**2005**) gives a first quasi-linear time algorithm; Gautier and Roch (**1997**) gave an NC^2 algorithm.

Main application: the equiprojectable decomposition of 0-dimensional varieties (Dahan, Moreno Maza, Schost, Wu and Xie **2005**).

Equiprojectable decomposition (Eq.D.): what it improves

- **Eq.D.:**

A canonical way of decomposing a zero-dimensional variety V into a union of equiprojectable ones.

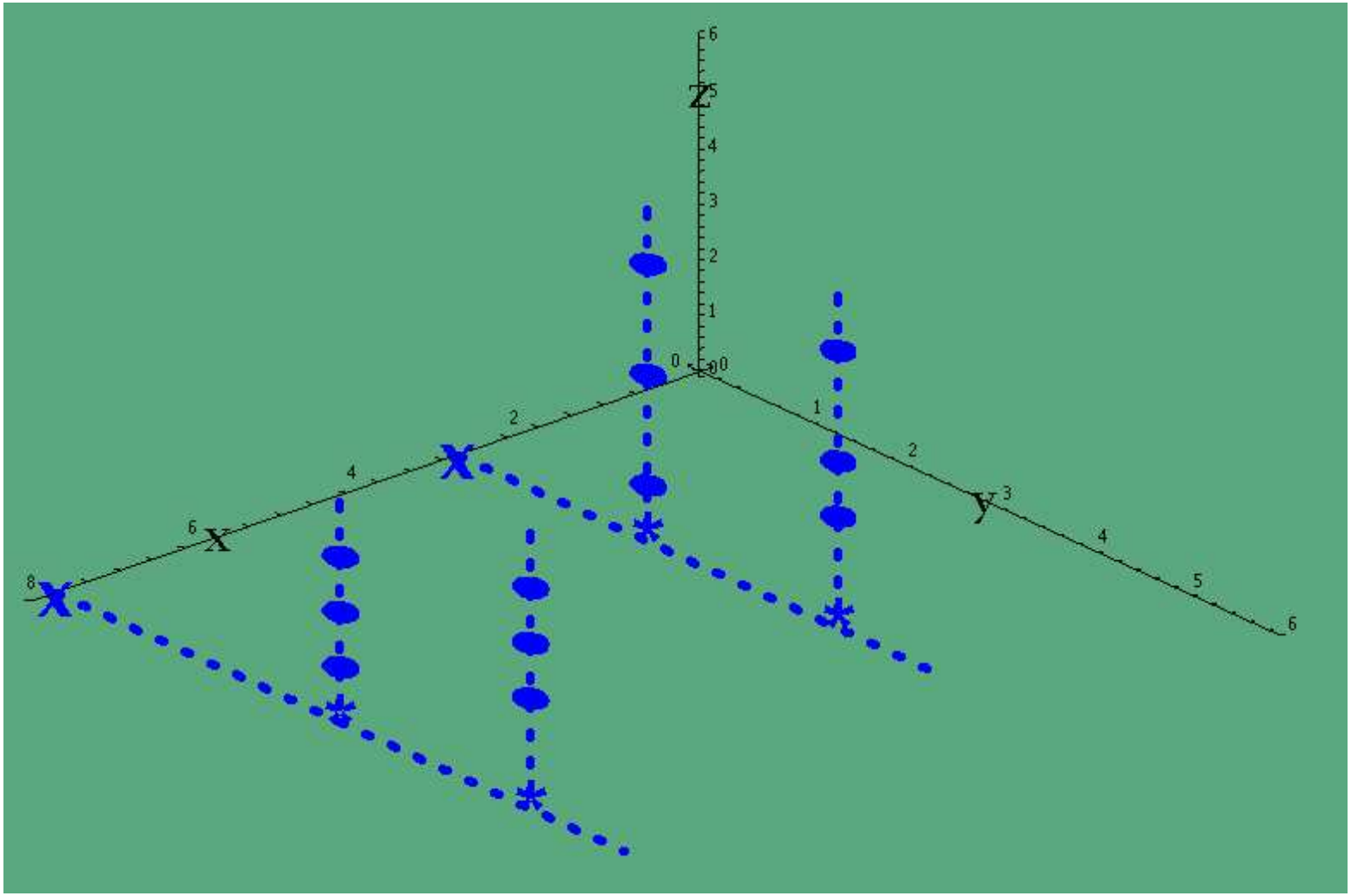
- **Motivated by:**

A zero-dimensional variety over a perfect field $\mathbb{K}$ is equiprojectable iff its defining ideal is generated by a triangular set (Aubry and Valibouze, 2000).

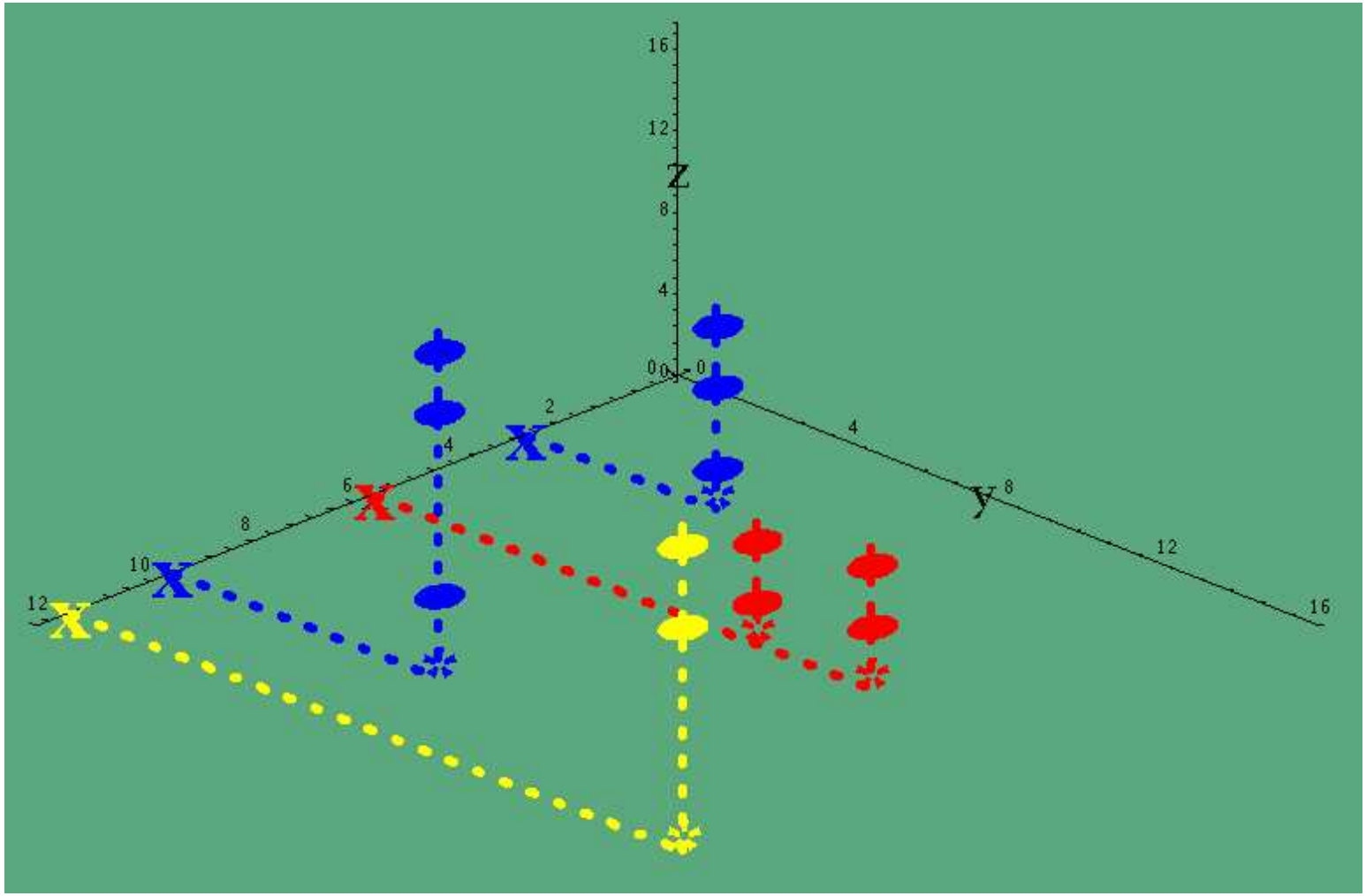
- **Properties of Eq.D.:**

  - has good specialization properties modulo a prime number p.
  - can be computed from any triangular decomposition of V.
  - leads to a **modular algorithm** computing its Eq.D..

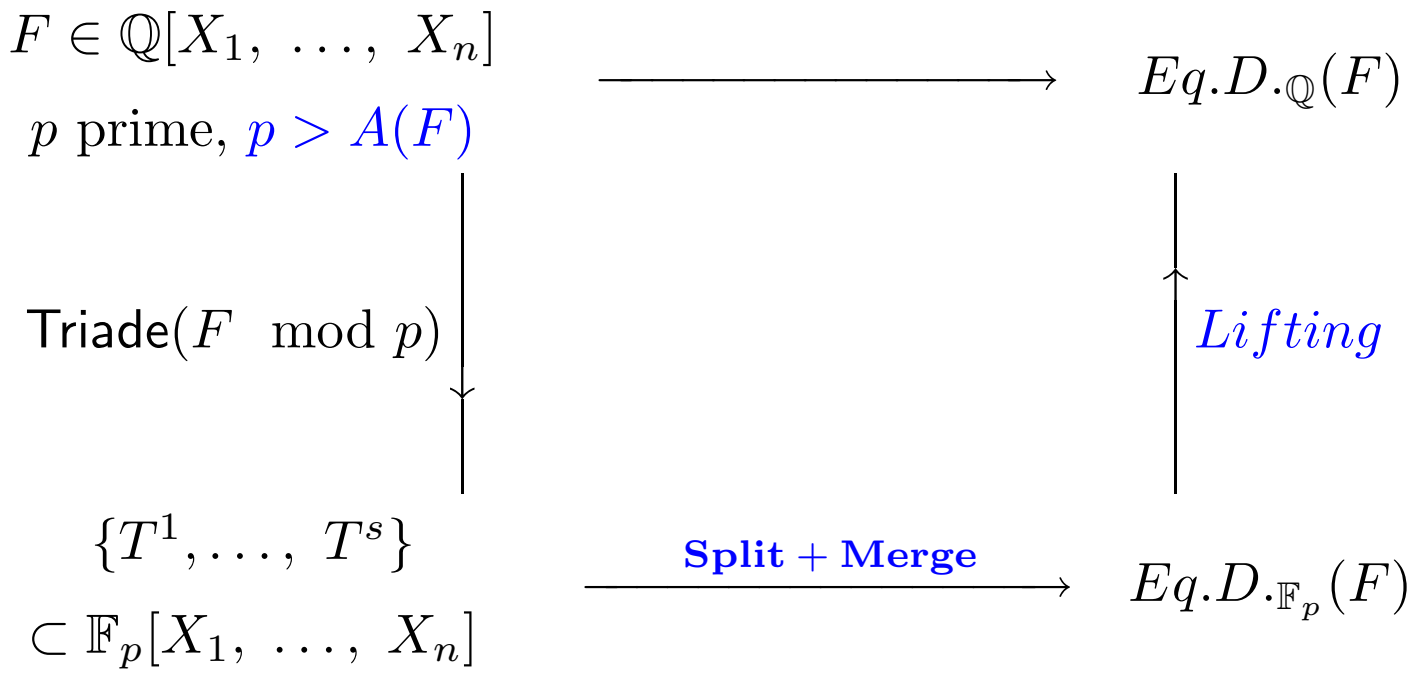
Equiprojectable variety definition



Equiprojectable decomposition definition



Our modular method for Tr.D.



• In practice we choose p much smaller with a probability of success, i.e. $> 99\%$ with $p \approx ln(A(F))$ (Dahan-Moreno Maza-Schost-Wu-Xie, 2006).

Table 1: Features of the polynomial systems and prime number for the modular algorithm

Sys	Name	n	d	p_1
1	fabfaux	3	3	121458749
2	geneig	6	3	303179363351
3	eco6	6	3	509110405373
4	Weispfenning-94	3	5	3441898787
5	Issac97	4	2	49956859
6	dessin-2	10	2	2011551274283
7	eco7	7	3	5433767329489
8	Reimer-4	4	5	180771302617
9	Methan61	10	2	3557395585699
10	Uteshev-Bikker	4	3	2197378999

Table 2: Experimental results from Maple

- Trian.Mod: implemented in the `RegularChains` library in Maple
- `Trian`: `RegularChains` solver based on Triade, non-modular algorithm.
- `gsolve`: Maple solver based on Gröbner bases.

Sys	Trian.Mod (sec)	`Trian` (sec)	`gsolve` (sec)		Trian.Mod (MB)	`Trian` (MB)	`gsolve` (MB)
1	27	512	1041		9	275	34
2	18	2.5	-		5	4	fail
3	50	5	9		6	5	5
4	100	3000	4950		12	250	66
5	161	-	1050		20	fail	31
6	524	-	-		14	fail	error
7	3795	1593	-		18	18	fail
8	5575	-	-		38	fail	fail
9	6184	-	-		12	-	fail
10	8726	-	-		64	fail	fail

Linear algebra over non-integral domains

- Matrix combine: an application of equiprojectable decomposition

  (demonstrate a Maple worksheet)

Modular method: related work

Non modular methods for Tr.D. algorithms:

- (Wu, 1987), (Chou & Gao 1990), (Wang 1993), ...
- (Kalkbrener 1991), (Lazard 1991), (Moreno Maza 2000), ...

Modular method for only one triangular set:

- using Hensel lifting (Schost 2002)
- using Chinese remaindering (Moreno Maza 2003)

Modular methods for Gröbner bases:

- (Trinks 1985), (Winkler 1988), (Arnold 2003), ...

Modular methods Primitive element representation:

- using Hensel lifting: (TERA group, 1997 - now)
- using Chinese remaindering: (Rouillier 1999) (Noro & Yokoyama 1999) ...

Our implementation environment:

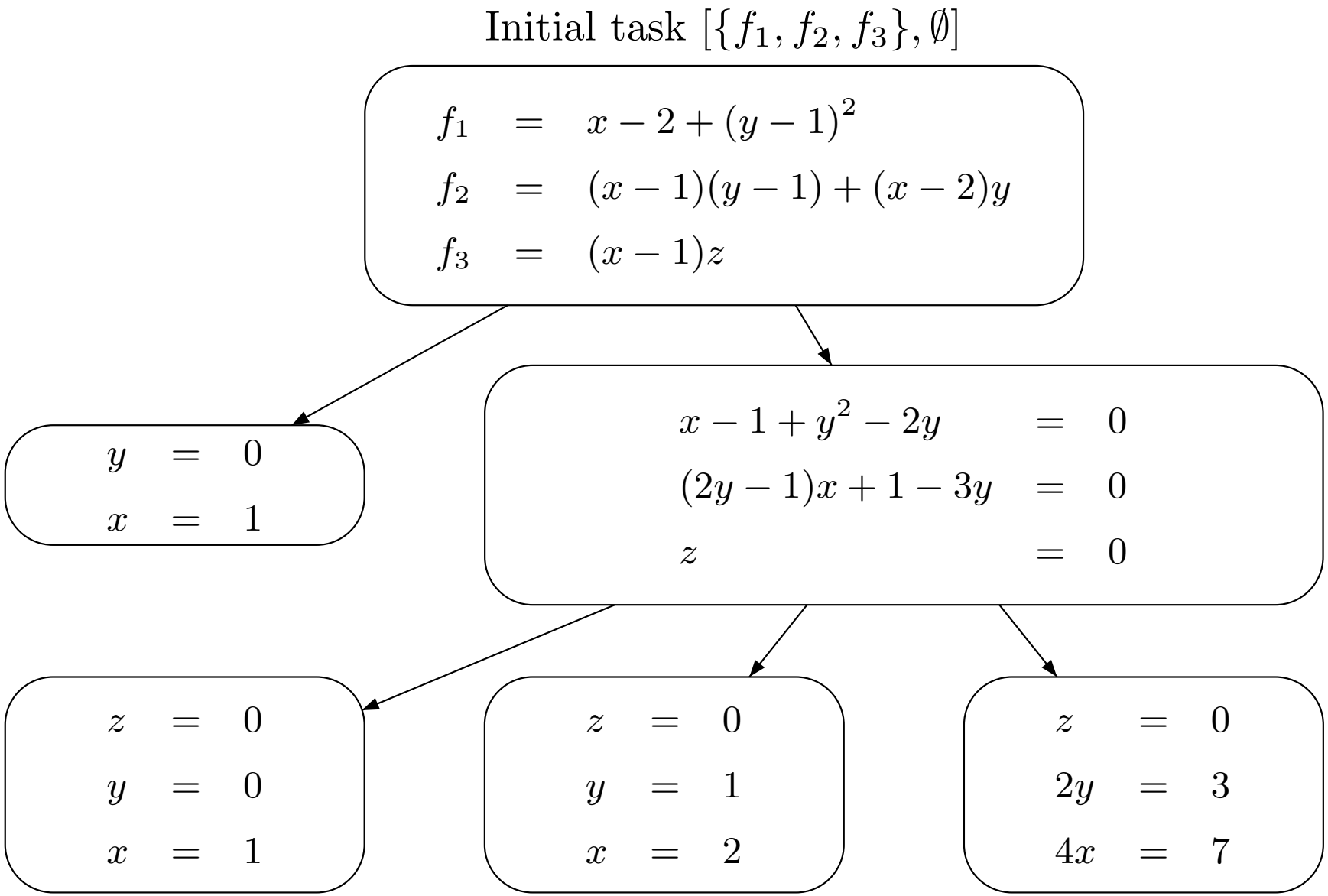
- the `RegularChains` library in Maple (Lemaire, Moreno Maza & Xie, 2005)

Component-level parallelization of Tr.D.

- **Motivation:**
  - renaissance of parallelism,
  - new algorithms, modular triangular decompositions, offering better opportunities for parallel execution,
  - to gain practical efficiency by high-performance computing

- **Our long-term goal:** multi-level parallelism for Tr.D.
  - coarse grained "component-level":
    for tasks computing geometric objects (Moreno Maza-Xie, 2007)
  - medium/fine grained level:
    for polynomial arithmetic within each task (e.g. Li-Moreno Maza, 2007)
  - In the component-level, the number of processes in use depends on the geometry of the solution set.

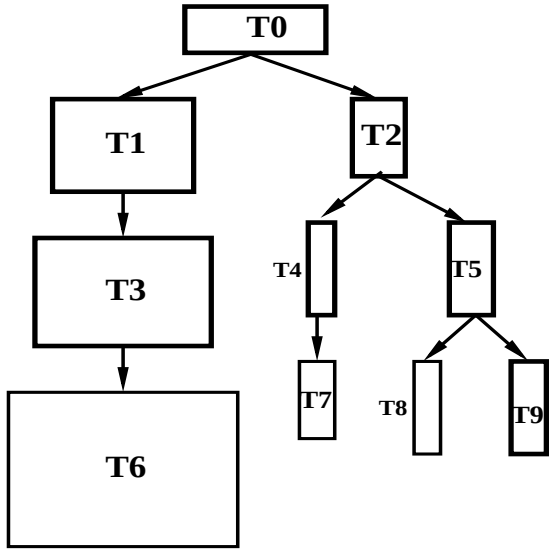
A Triade task tree



Component-level parallelization of Triade: difficulties

- **Dynamic and very irregular tasks**
  - #tasks, CPU, memory, data-communication

- **Almost no parallelism over $\mathbb{Q}$**
  - Most polynomial systems $F \subseteq \mathbb{Q}[X]$ can be represented by a single triangular set.
    * This is true in theory, Shape Lemma: (Becker-Mora-Marinari-Traverso, 1994).
    * and in practice: SymbolicData.org.



- **Even worse: redundant tasks**

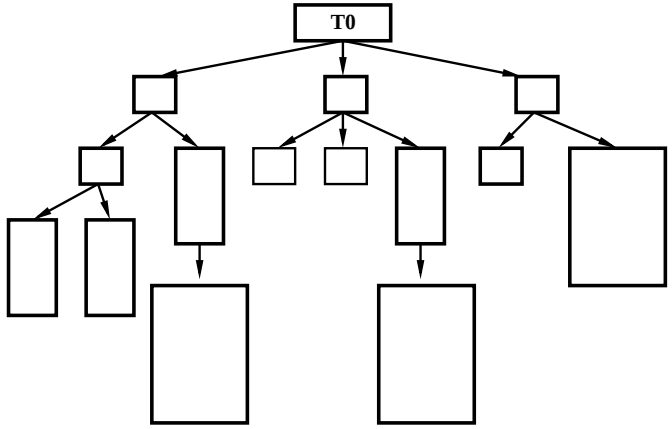
Create parallelism: using modular methods

- **For solving $F \subseteq \mathbb{Q}[X]$ we use modular methods**

  (Dahan-Moreno Maza-Schost-Wu-Xie, 2005)

  Indeed, for a prime p:

  - irreducible polynomials in $\mathbb{Q}[X]$ are likely to factor modulo p,
  - for p big enough, the result over $\mathbb{Q}$ can be recovered from the one over $Z/pZ[X]$.



- We apply parallel execution to the modular solving phase (with dominant cost) to gain practical efficiency.

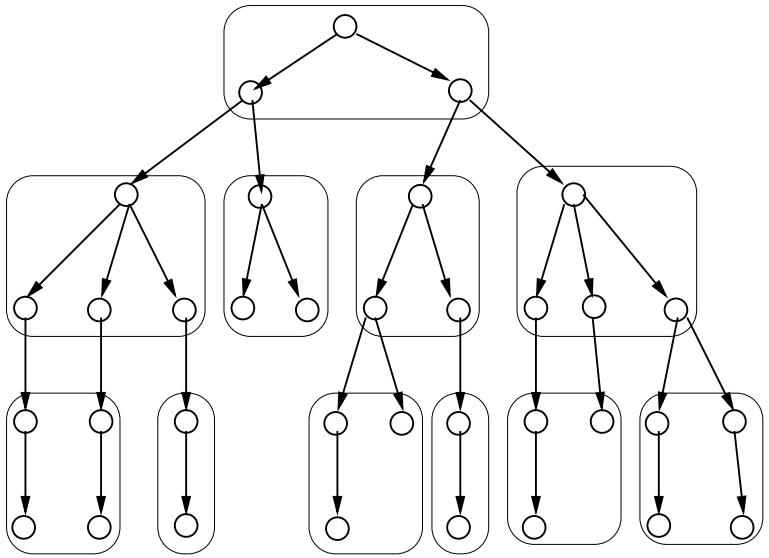
Effect of modular solving

Sys	Name	n	d	p	*Degrees* in Tr.D. (mod p)
1	eco6	6	3	105761	[1,1,2,4,4,4]
2	eco7	7	3	387799	[1,1,1,1,4,2, 4,4,4,4,4,2]
3	CassouNogues2	4	6	155317	[8]
4	CassouNogues	4	8	513899	[8,8]
5	Nooburg4	4	3	7703	[18,6,6,3,3,4,4,4,4,2,2,2, 2,2,2,2,1,1,1,1,1]
6	UteshevBikker	4	3	7841	[1,1,1,1,2,30]
7	Cohn2	4	6	188261	[3,5,2,1,2,1,1,16,12,10,8,8, 4,6,4,4,4,4,2,1,1,1,1,1,1,1, 1,1,1,1,1,1,1]

- Each of these examples, except for Cohn2, is equiprojectable.
- Tr.D. modulo p results in more number of components.

Exploit parallelism!

- **Driving idea: limit the irregularity of tasks**.
  - to avoid inexpensive computations leading to expensive data communication,
  - to balance the work among the workers,
    * use regularized initial and split-by-height,
    * estimate the cost of a task by its rank and dimension to guide the scheduling.



Implementation of the component-level parallel Tr.D.

- **Implementation challenges**
  - scheduling of highly irregular tasks,
  - dynamic process management,
  - communication and synchronization for complex data types, e.g. sparse multivariate polynomial

- **Our solutions**
  - Parallel framework: multiprocessed parallelism in Aldor
  - Supported with `BasicMath` library and sequential `Triade` solver in Aldor, on machine Silky (1.6GHz 128 cpus SMP) in SHARCNET
  - Task Pool with Dimension and Rank Guided dynamic scheduling

Preliminary result: speedup vs #processor

#P	Sys1	Sys2	Sys3	Sys4	Sys5	Sys6	Sys7
1	4 (sec)	707	463	2133	4	866	298
3	1.3	2.1	1.7	1.5	2.0	1.4	2.9
5	2.1	3.2	2.2	2.2	2.0	1.8	3.1
7	2.1	5.1	2.3	2.3	2.2	1.8	3.1
9	2.1	6.1	2.3	2.4	2.3	1.9	3.2
11	2.0	6.1	2.3	2.4	2.6	1.9	3.2
13	2.1	6.1	2.3	2.4	2.5	1.9	3.2

- These belong to small to medium sized problems.
- The speedup ratio reflects the number of large components in the output.

Parallel solving: related work

- Parallelizing the computation of Gröbner bases
  - (S. Chakrabarti & K. Yelick, 1993 - 1994)
  - (R. Bündgen, M. Göbel & W. Küchlin, 1994), (J.-C. Faugère, 1994)
  - (G. Attardi & C. Traverso, 1996)
  - (A. Leykin, 2004)

- Parallelizing the computation of characteristic sets
  - (D.M. Wang, 1994), (I.A. Ajwa, 1998)
  - (Y.W. Wu, W.D. Liao, D.D. Liu & P.S. Wang, 2003)
  - (Y.W. Wu, G.W. Yang, H. Yang, H.M. Zheng & D.D. Liu, 2005)

The parallelized operation is polynomial reduction (or simplification).

A framework for high-performance computer algebra

• **Multiprocessed parallelism support in Aldor on SMPs and multicores**

(Moreno Maza-Stephenson-Watt-Xie, 2007)

- Aldor is a categorical programming language and interoperable with other languages like C and machine resources for high-performance computing.
- `BasicMath` library and `triade` solver by the ESPRIT Project FRISCO (1996-1999)

- **Parallel constructs**
  - packages for inter-process data communication and synchronization via shared memory segments
  - simple interface for dynamic process management and user-level scheduling
  - packages for serializing and de-serializing high-level Aldor objects

Data communication and synchronization

- **Via two shared memory segments (UNIX System V standard for IPC)**

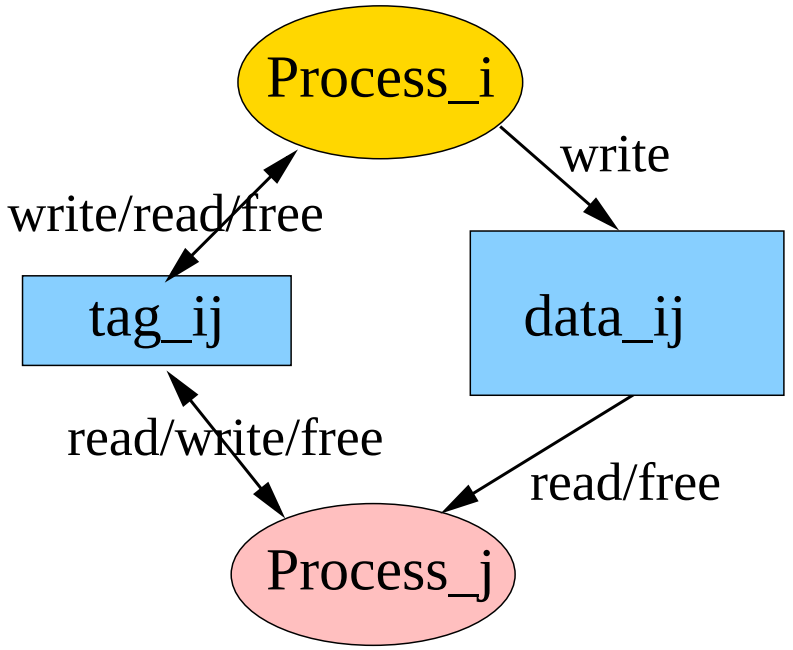
- **Synchronization protocol:**

Sending:

if $tag\_ij = 0$ then
write data into $data\_ij$;
$tag\_ij \leftarrow$ sizeOfdata;

Receiving:

if $tag\_ij > 0$ then
sizeOfdata $\leftarrow$ tag_ij;
$tag\_ij \leftarrow -1$;
Read data from $data\_ij$;
$tag\_ij \leftarrow 0$;



Dynamic process management

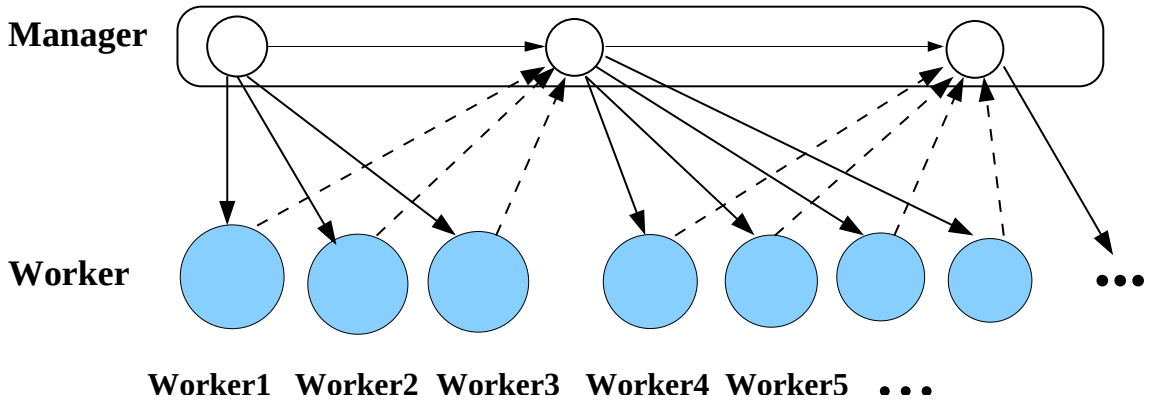
- `Spawn`(*command, argument*)
  - Aldor's `run()`, `system()` in C on UNIX
  - can be used within a process to launch one or more additional processes that will run other programs independently.

- User defined **Task** with **virtue process identifier (VPID)**
  - analogous to a processor's rank in MPI

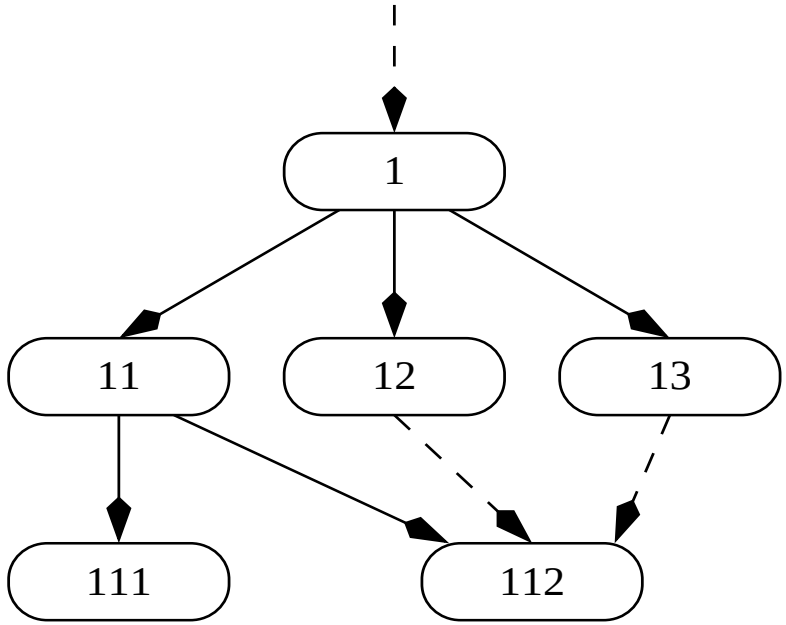
- This **VPID** is used by a process to communicate with other processes
  - by creating the unique keys to the two shared memory segments, i.e. tag and data

Dynamic process management and user-level scheduling

Task farming scheme: easy to apply greedy scheduling.



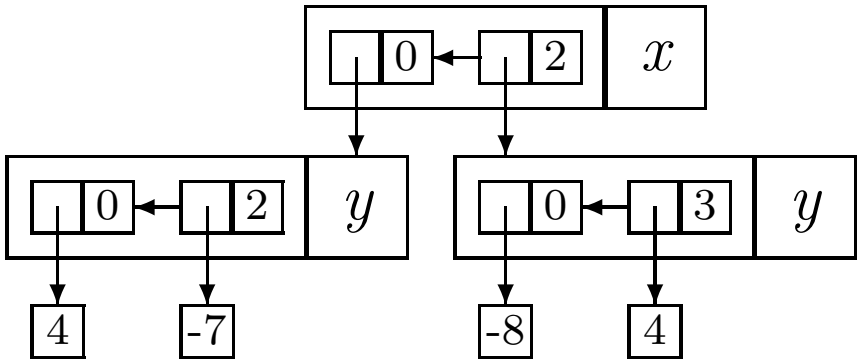
Dynamic fully-strict task processing: this solution is akin to the scheme for handling the rank of spawned processes in MPICH2.



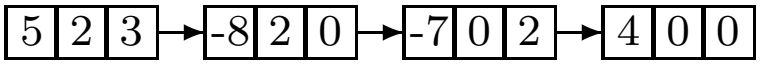
Serialization of high-level objects

- Polynomial types in `BasicMath` library: e.g. $g = 5x^2y^3 - 8x^2 - 7y^2 + 4$

  - `SparseMultivariatePolynomial` (SMPLY): suitable for Tr.D.
    $g \to (4 - 7y^2) + (-8 + 5y^3)x^2$ for $x > y$;



  - `DistributedMultivariatePolynomial` (DMPOLY): suitable for GB



- Aldor package: `Serialization`

  - `SerializeSMPbyKronecker()`: $g \to \{5, 0, 0, 0, -7, 0, 0, 0, -8, 0, 4\}$

  - `SerializeSMPbyDMP()`: $g \to \{5, 2, 3, 8, 2, 0, 7, 0, 2, 4, 0, 0\}$

Benchmark: overhead of the parallel constructs

- Using the component-level parallel solver, we measure the cost of:
  - process spawning
  - use of tag segments
  - data communication/serialization by
    - write by `SerializeSMPbyKronecker()`,
      read by `UnserializeSMPbyKronecker()`
    - write by `SerializeSMPbyDMP()`,
      read by `UnserializeSMPbyDMP()`

Analysis of workers' overhead for Kronecker

Sys	Per Spawn (ms)	Per Tag (μs)	Read and Unserialize (μs per int)	Serialize and Write (μs per int)	Over-head (%)
1	40	118	119	21	24.4
2	24	142	43	6	0.3
3	24	152	169	11	2.4
4	40	172	318	2	3.3
5	32	138	36	-	26.4
6	32	158	168	6	2.8
7	21	136	60	9	1.3
AVG	30	145	130	9	-

- Overhead is negligible, or satisfactory for uneasy problems
- Unserializing is more expensive than serializing

Analysis of workers' overhead for DMPOLY

Sys	Per Spawn (ms)	Per Tag (μs)	Read and Unserialize (μs per int)	Serialize and Write (μs per int)	Over-head (%)
1	35	134	68	17	20.0
2	29	146	36	18	0.4
3	45	146	180	21	0.8
4	41	179	478	14	3.8
5	39	158	59	-	36.2
6	41	160	192	19	6.7
7	26	151	56	20	3. 2
AVG	37	153	152	18	-

- Overhead is negligible, or satisfactory for uneasy problems
- Unserializing is more expensive than serializing: another illustration.

Parallel framework: related work

- Aldor related:
  - Piit (Gautier-Mannhart, 1998)
- Other parallel computer algebra systems:
  - PARSAC-2 (Küchlin, 1990), PACLIB (Schreiner-Hong, 1993)
  - DSC (Díaz-Hitz-Kaltofen-Lobo-Valente, 1995)
  - see a complete overview in *Computer Algebra Handbook* (Grabmeier-Kaltofen-Weispfenning, 2003)
- Others:
  - KAAPI (KAAPI group)
  - Cilk (Cilk group)

Verification of solvers: introductory example

- Polynomial system solvers: complex software and no canonical output!

An example: $F : \begin{cases} x^{31} - x^6 - x - y = 0 \\ x^8 - z = 0 \\ x^{10} - t = 0 \end{cases}$,

Output 1:

$$\begin{cases} (t^4 - t)\, x - ty - z^2 = 0 \\ t^3y^2 + 2t^2z^2y - (t^6 - 2t^3 - t + 1)\, z^4 = 0 \\ z^5 - t^4 = 0 \\ t^4 - t \neq 0 \end{cases}, \quad \begin{cases} x^2 - z^4 = 0 \\ y + t^2z^2 = 0 \\ z^5 - t = 0 \\ t^3 - 1 = 0 \end{cases}, \quad \begin{cases} x = 0 \\ y = 0 \\ z = 0 \\ t = 0 \end{cases}$$

Output 2:

$$\begin{cases} (t^4 - t)\, x - ty - z^2 = 0 \\ tzy^2 + 2z^3y - t^8 + 2t^5 + t^3 - t^2 = 0 \\ z^5 - t^4 = 0 \\ z\,(t^4 - t) \neq 0 \end{cases}, \quad \begin{cases} zx^2 - t = 0 \\ ty + z^2 = 0 \\ z^5 - t = 0 \\ t^3 - 1 = 0 \\ tz \neq 0 \end{cases}, \quad \begin{cases} x = 0 \\ y = 0 \\ z = 0 \\ t = 0 \end{cases}$$

- **Are both correct?**
- **Verifying them can be harder than solving F!**

  No tools were reported!

Verification: what is a solver?

- Let $F = \{F_1, \ldots, F_m\} \subset \mathbb{K}[x_1, \ldots, x_n]$. A point $\mathbf{x} \in \overline{\mathbb{K}}^n$ solves F if

$$F_1(\mathrm{x}) = \cdots = F_m(\mathrm{x}) = 0.$$

- A polynomial system solver computes solved systems $S_1, \ldots, S_e$ s. t.:

a point x solves F iff x solves one of the $S_1, \ldots, S_e$.

- A solved system is a pair $S = [T, h]$ where $T \subset \mathbb{K}[x_1, \ldots, x_n]$ is a triangular set and $h \in \mathbb{K}[x_1, \ldots, x_n]$. A point $\mathrm{x} \in \overline{\mathbb{K}}^n$ solves $[T, h]$ if

$$\begin{cases} T_n(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{n-1}, x_n) = 0 \\ T_{n-1}(x_1, \ldots, x_d, x_{d+1}, x_{d+2}, \ldots, x_{n-1}) = 0 \\ \vdots \\ T_{d+1}(x_1, \ldots, x_d, x_{d+1}) = 0 \\ h(x_1, \ldots, x_d) \neq 0 \end{cases}$$

Let $\mathbf{Z(T, h)} \subseteq \overline{\mathbb{K}}^n$ be the solution set of $[T, h]$ and $V(F) \subseteq \overline{\mathbb{K}}^n$ that of F.

Verification of solvers: approaches

• **The verification problem:** given an input system F and solved systems $[T_1, h_1], \ldots, [T_e, h_e]$, decide whether we have:

$$V(F) = \bigcup_{i=1}^{e} Z(T_i, h_i).$$

• **Gröbner bases offer a direct solution:**

- straightforward but very expensive (using radical membership tests),
- moreover, does not exploit the structure of the solved systems.

• **Our approach:**

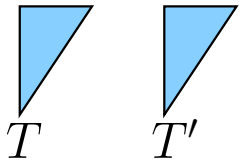
- compare the candidate output $[T_1, h_1], \ldots, [T_e, h_e]$,
- with the trusted output $[T'_1, h'_1], \ldots, [T'_s, h'_s]$ of another solver.
- This reduces to compute $Z(T, h) \setminus Z(T', h')$:

∗ Inclusion test $Z(T, h) \subseteq Z(T', h')$ for removing redundant components reduces to difference computations!

Verification of solvers: our `Difference` algorithm

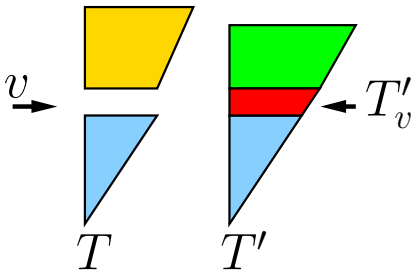
- Computing $Z(T,h) \setminus Z(T',h')$ with $h = h' = 1$ by exploiting the triangular structure level by level (Chen-Moreno Maza-Pan-Xie, 2007).

Case 1:



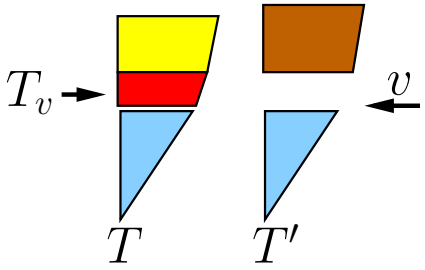
$\langle T \rangle = \langle T' \rangle$, Easy!

Case 2:



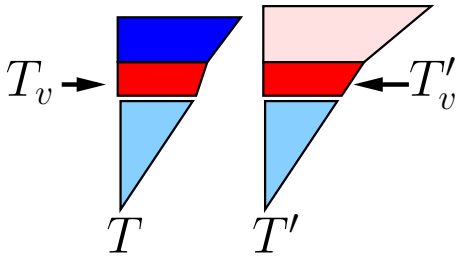
Output $[T, T_v']$;

Case 3:



Output $\texttt{Difference}(T, V(T_v) \cap V(T'))$;

Case 4:



- $g := \mathsf{GCD}(T_v, T_v', T_{<v})$;
- $g \in \mathbb{K} \Rightarrow$ Output $[T, 1]$;
- $\mathrm{mvar}(g) < v \Rightarrow$ { Output $[T, g]$;
  Output $\texttt{Difference}(V(g) \cap V(T), T')$ };
- Output $\texttt{Difference}(T_{<v} \cup \{g\} \cup T_{>v}, T')$;
- Output $\texttt{Difference}(T_{<v} \cup \{T_v/g\} \cup T_{>v}, T')$;

Verification of solvers: experimentation

- `Diff-verifier()` and `GB-verifier()`: implemented in `RegularChains`.

- Some well-known polynomial system solvers:

Solver	**Library**	**Author**	**Algorithm**
`RegSer`	`Epsilon`	Wang	(Wang, 2000)
`SimSer`	`Epsilon`	Wang	(Wang, 1998)
`Triangularize`	`RegularChains`	Lemaire et al.	(Moreno Maza, 2000)
`triade`	`BasicMath`	Moreno Maza	(Moreno Maza, 2000)

*All these libraries are Maple code except `BasicMath` written in Aldor.

- By solving and verifying 31 problems:
  - `Diff-verifier()` and `GB-verifier()` agree on the problems that can be checked by `GB-verifier()`.
  - `GB-verifier()` failed to check many examples.

* `Diff-verifier()` runs much faster.
* `Diff-verifier()` works for all examples that can be solved.

Summary and future work

- **Our contributions**
  - Fast polynomial arithmetic for computing modulo a triangular set
  - Equiprojectable decompositions and a first modular algorithm for Tr.D.
  - A first coarse-grained parallelization of Tr.D.
  - A framework for high-performance computer algebra
  - Irredundant and verified triangular decompositions

Our results: new algebraic methods, new complexity results, new implementation framework, and new software tools.

- **Future work**
  - Complexity analysis of algorithms computing Tr.D. (starting with 0-dim systems)
  - Extend the modular method to more systems and coefficient fields
  - Multi-level parallelization of Tr.D.

Acknowledgment

- My Supervisors: Dr. Marc Moreno Maza and Dr. Stephen M. Watt
- My coauthors: Changbo Chen, Dr. Xavier Dahan, Dr. François Lemaire, Wei Pan, Dr. Éric Schost, Dr. Ben Stephenson and Dr. Wenyuan Wu
- All the Profs, Staffs and members in ORCCA and CS Department, UWO
- My thesis examiners:

  Professor Rob Corless, Applied Mathematics, UWO

  Professor Erich Kaltofen, Mathematics & Computer Science, North Carolina State University

  Professor Hanan Lutfiyya, Computer Science, UWO

  Professor Sheng Yu, Computer Science, UWO

Thank you!

What I know, a drop, what I do not know, an ocean!

Main Methods for Solving Non-linear Systems (1/3)

- Gröbner Basis (GB)
- Triangular Decomposition (TRD)
- Primitive Element Representation

Example: $F \in \mathbb{Q}[x, y, z]$

$$\begin{cases} -z^5 - z^4 - 2z^3 - z^2 + 3z + y + x + 3 = 0 \\ -2z^5 - 4z^3 - z^2 + 6z + y^2 + 2y = 0 \\ z^3 + yz^2 - z - y = 0 \\ z^6 + z^4 - 4z^2 + 2 = 0. \end{cases}$$

Main Methods for Solving Non-linear Systems (2/3)

F has lexicographical $(x > y > z)$ **Gröbner basis** :

$$\begin{cases} x^3 + 2x^2 - 2x + z^2 - 5 \\ -3x^3 - 4x^2 + yx + zx + 8x + zy + y + z + 11 \\ -x^3 + zx^2 - x^2 + 3x + y - 2z + 3 \\ 3x^3 + 4x^2 - 8x + y^2 - 11 \\ x^3 + yx^2 + x^2 - 3x - 3y - 3 \\ x^4 - 5x^2 + 6 \ , \end{cases}$$

and **triangular decomposition** $(x > y > z)$:

$$\begin{cases} z^4 + 2z^2 - 2 = 0 \\ y + z = 0 \\ x + z^2 + 1 = 0 \end{cases} \quad \text{and} \quad \begin{cases} z^2 - 1 = 0 \\ y^2 + 2y - 1 = 0 \\ x + y + 1 = 0. \end{cases}$$

Main Methods for Solving Non-linear Systems (3/3)

F has **primitive element representation** :

$$\begin{cases} x = -t^2 - 1 \\ y = -t \\ z = t \end{cases} \quad \text{where} \quad t^4 + 2t^2 - 2 = 0 \quad \text{or,}$$

$$\begin{cases} x = t \\ y = -t - 1 \\ z = -1 \end{cases} \quad \text{where} \quad t^2 - 2 = 0 \quad \text{or,}$$

$$\begin{cases} x = t - 2 \\ y = -t + 1 \\ z = 1 \end{cases} \quad \text{where} \quad t^2 - 4t + 2 = 0.$$

Main Objectives of this Thesis

- **To improve the performance of a symbolic solver for triangular decomposition by**

  (1) fast algorithms,

  (2) modular methods,

  (3) parallel approaches, and

  (4) software engineering

- Why TRD?

  a more geometrical approach ...

Regular chains 1/2

Consider ordered variables $\mathbf{X} = X_1 > \cdots > X_n$. Let $\mathbf{C} = C_1, \ldots, C_s$ be in $\mathbb{K}[\mathbf{X}]$, with main variables $X_{\ell_1} < \cdots < X_{\ell_s}$. For $i \leq s$, the initial h_i is the leading coefficient of C_i in X_{ℓ_i}.

The saturated ideal is $\mathrm{Sat}_i(\mathbf{C}) = (C_1, \ldots, C_i) : (h_1 \ldots h_i)^\infty$.

$\mathbf{C}$ is a regular chain if h_i is regular mod $\mathrm{Sat}_i(\mathbf{C})$ for all i.

The quasi-component $W(\mathbf{C}) := V(\mathbf{C}) \setminus V(h_1 \cdots h_{\ell_s})$ satisfies $\overline{W(\mathbf{C})} = V(\mathrm{Sat}_n(\mathbf{C}))$.

The algebraic variables are those which appear as main variables. The other ones are free.

Example

$$\left| \begin{array}{l} C_2 = (X_1 + X_2){X_3}^2 + X_3 + 1 \\ C_1 = {X_1}^2 + 1. \end{array} \right. , \text{ with } \left| \begin{array}{l} \mathsf{mvar}(C_2) = X_3 \\ \mathsf{mvar}(C_1) = X_1 \end{array} \right. .$$

Regular chains (2/2)

The regular chains are simple data structures, well-suited to describe the generic points of varieties of positive dimension.

In positive dimension, lexicographic Gröbner bases become complicated to understand. Modular algorithms become harder to design.

References:

- Lazard. A new method for solving... (1991)
- Kalkbrener. Generalized Euclidean algorithm... (1993)
- Moreno Maza. On triangular decompositions... (2000)
- Lemaire - Moreno Maza - Xie. The `RegularChains` library. (2005)

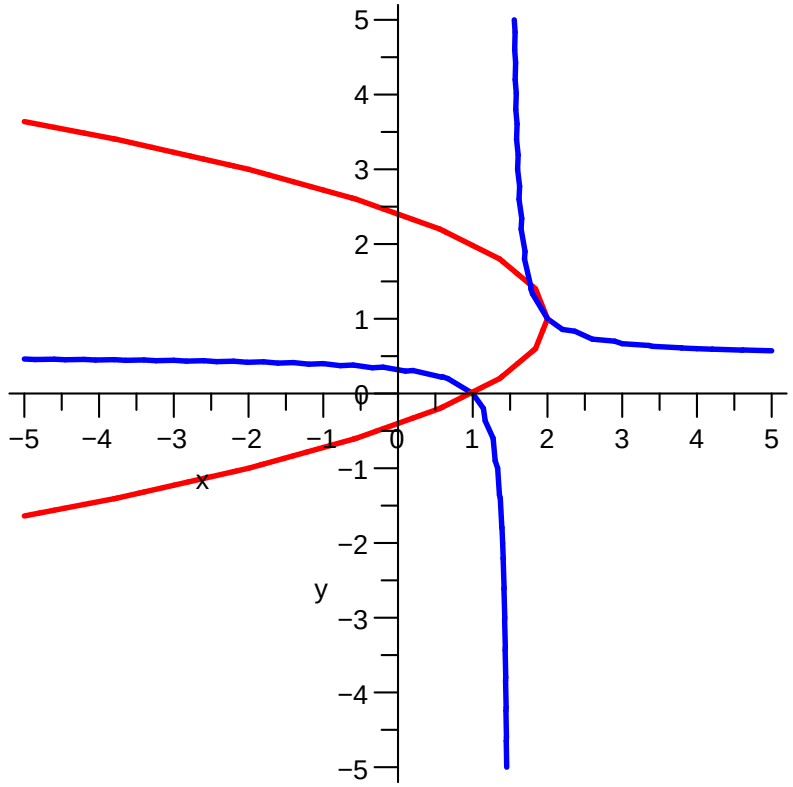
Triade top level

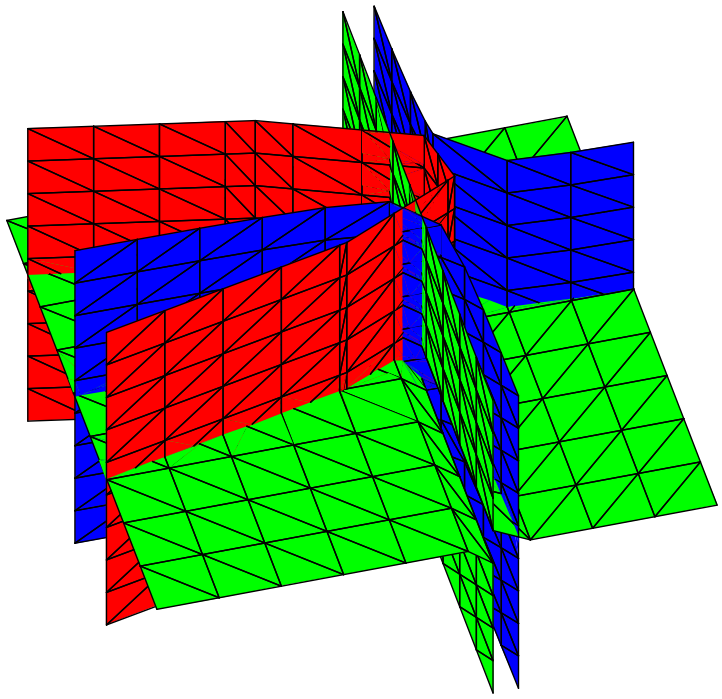
Input: $F \subset \mathbb{K}[X]$.

Output: $\mathcal{T}$ a triangular decomposition of $V(F)$.

$ToDo := [F, \emptyset]$; $\mathcal{T} := [\,]$
repeat
(S) $Tasks :=$ Select$(ToDo)$
(R) $Results :=$ LazySolve$(Tasks)$
(U) $(ToDo, \mathcal{T}) :=$ Update$(Results, ToDo, \mathcal{T})$
until $ToDo = \emptyset$
return $\mathcal{T}$

Difficulty 1: Removing redundant computation





The red and blue surfaces intersect on the line $x - 1 = y = 0$ contained in the green plane $x = 1$. With the other green plane $z = 0$, they intersect at $(2, 1, 0)$, $(\frac{7}{4}, \frac{3}{2}, 0)$ but also at $x - 1 = y = z = 0$, which is redundant.

Summary

- Created opportunities by using modular methods, for coarse grained component-level parallel solving of polynomial systems in $\mathbb{Q}[X]$
- Exploited these opportunities by transforming the Triade algorithm: strengthen its notion of a task by regularized initial and split-by-height.
- Geometrical information guided scheduling.
- A preliminary implementation using multi-processed parallelism support in Aldor.
- Launched the first step towards multi-level parallelization.
- Expect the speedup in component-level parallelization would add a multiplicative factor to the medium/fine level.
- Limitation of this implementation: memory

Towards efficient multi-level parallelization

- Build Aldor threads to support fine parallelism for symbolic computations targeting SMP and multi-cores. In particular,
  - properly treat parametric types, such as polynomial data types,
  - thread scheduling by *work-stealing* and *work first principle*.
- Investigate multi-level parallelism for triangular decompositions over clusters:
  - coarse grained level (multi-processed) for tasks to compute geometric of the solution sets.
  - medium/fine grained level (multi-threaded) for polynomial arithmetic such as multiplication, GCD/resultant, and factorization.
  - to make good use of CPU and memory resources on emerging architectures.

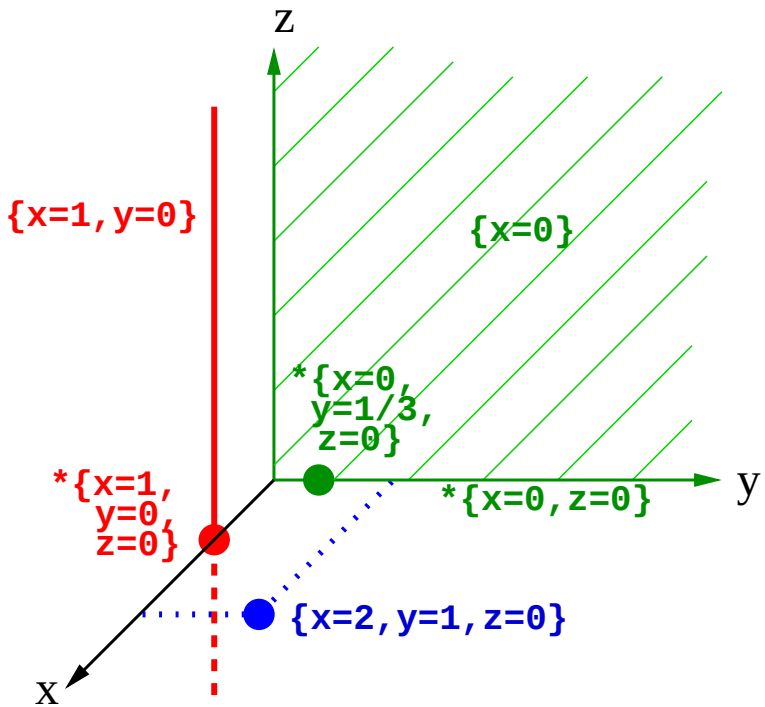
Redundance in Tr.D.

• Redundance - solutions of being superfluous and unneeded.

Example: $\begin{cases} x^3 - 3x^2 + 2x = 0 \\ 2yx^2 - x^2 - 3yx + x = 0 \\ zx^2 - zx = 0 \end{cases} \Rightarrow$

$$\left\{ x = 0 \quad \cup \begin{cases} y = 0 \\ x = 1 \end{cases} \quad \cup \begin{cases} z = 0 \\ y = 1 \\ x = 2 \end{cases} \quad \cup \right.$$

$$\begin{cases} z = 0 \\ y = 0 \\ x = 1 \end{cases} \cup \begin{cases} z = 0 \\ 3y = 1 \\ x = 0 \end{cases} \cup \begin{cases} z = 0 \\ x = 0 \end{cases}$$



- An important issue with all decomposition algorithms
- If do not remove, then computational cost increases, and outputs are hard to read and hard to use.

Inclusion test criteria

(Chen-Lemaire-Moreno Maza-Pan-Xie, 2007)

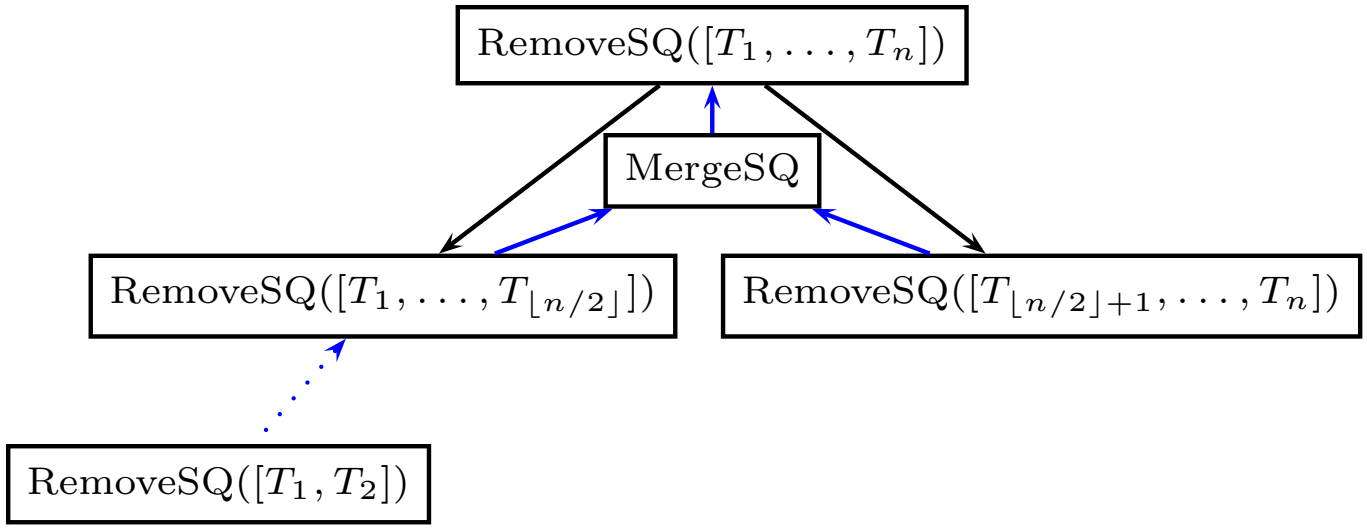
Proposition 1:
The inclusion $W(T) \subseteq W(U)$ holds if and only if the following both statements hold

(C_1) for all $p \in U$, $p \in \sqrt{\mathrm{Sat(T)}}$,

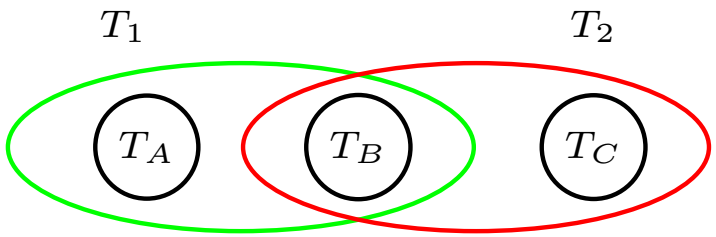
(C_2) $W(T) \cap V(h_U) = \emptyset$.

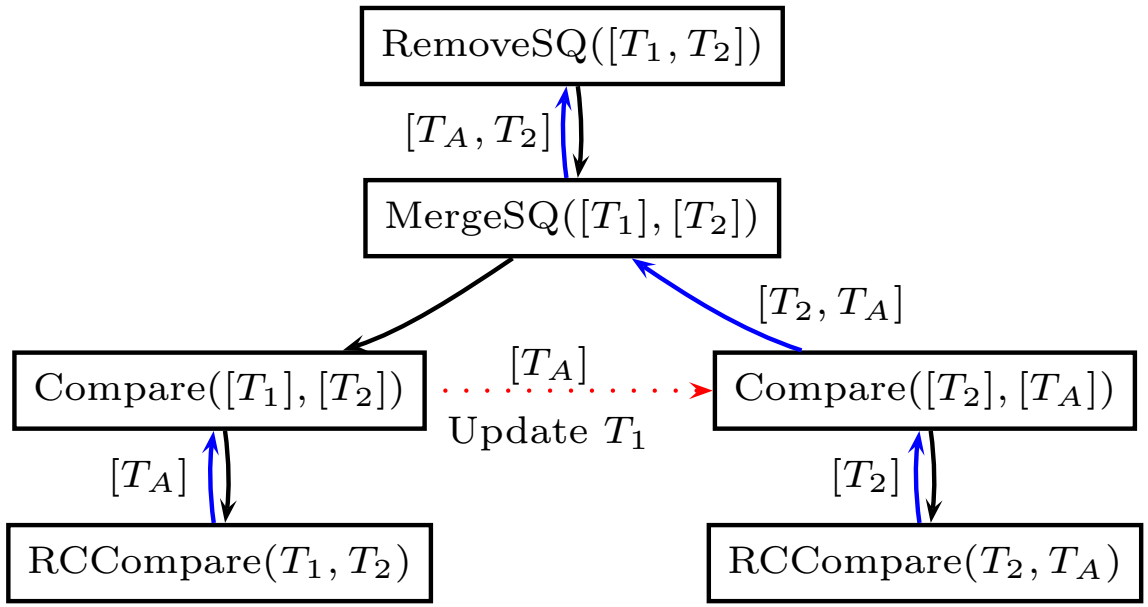
- If Sat(T) is radical, then condition (C_1) can be replaced by:
  - (C_1') for all $p \in U$, $p \in \mathrm{Sat(T)}$, easier to check.
- Checking (C_2) can be approached in different ways, depending on the computational cost that one is willing to pay.
  - (C_2') for all $S \in \mathrm{Triangularize(T \cup \{h_U\})}$, $h_T \in \sqrt{\mathrm{Sat(S)}}$, certified but higher cost!
  - (C_h) $\mathrm{Intersect(h_U, T)} = \emptyset$, lower cost (fact: T is a regular chain), but if (C_h) does not hold, we can not conclude.

Remove redundance in Tr.D.: divide-conquer



Divide-conquer: base case





Remove redundance in Tr.D.: benchmark (1/2)

- `RemoveSQ()` : implemented in the **`RegularChains`** library in Maple

Sys	Name	Triangularize (No removal)		Certified (C_1) and (C_2')	
		♯ RC	time(s)	♯ RC	time(s)
1	genLinSyst-3-2	20	1.684	17	1.182
2	Butcher	15	9.528	7	0.267
3	MacLane	161	12.733	27	7.144
4	neural	10	14.349	4	8.948
5	Vermeer	6	27.870	5	58.396
6	Liu-Lorenz	23	29.044	16	121.793
7	chemical	7	71.364	5	7.727
8	Pappus	393	37.122	120	141.702
9	Liu-Lorenz-Li	22	1796.622	9	96.364
10	KdV572c11s21	41	8898.024	7	6.980

Certified removal is quite expensive!

Remove redundance in Tr.D.: benchmark (2/2)

Sys	Heuristic (C_1') and (C_h) (without split)		Certification (C_1) and (C_2')		Heuristic (C_1) and (C_h) (with split)		Certified (C_1) and (C_2')	
	♯ RC	time(s)	♯ RC	time(s)	♯ RC	time(s)	♯ RC	time(s)
1	17	0.382	17	1.240	17	0.270	17	1.214
2	7	0.178	7	0.259	7	0.147	7	0.325
3	27	3.437	27	8.470	27	3.358	27	8.239
4	4	1.881	4	8.353	4	6.429	4	14.045
5	5	0.771	5	60.108	8	54.455	8	109.928
6	16	1.937	16	123.052	18	96.492	18	203.937
7	5	0.243	5	7.828	5	5.180	5	12.842
8	124	42.817	120	135.780	124	48.756	120	148.341
9	9	8.186	9	101.668	10	105.598	10	217.837
10	7	4.878	7	6.688	7	5.881	7	7.424

Heuristic removal is inexpensive and effective!

Comparison of three implementations of Triade

The `RegularChains` library in Maple (Lemaire-Moreno Maza-Xie, 2005)

- general purpose, and very broad users; require ease-of-use
- two-level user-interface for 50 functions: top-level module, `ChainTools`and `MatrixTools` submodules
- code is organized as files to mimic O-O
- native Maple polynomial representation (DAG)
- regular chain is a table (3 fields for polys, cached info and flags)
- large test suites for 0-dim

The AXIOM implementation (Moreno Maza, 1998)

- categorical programming, generic and reliable procedures
- dynamic typing to avoid type instantiations and conversions
- implement the algorithms without any restrictions w.r.t. the theory
- interface high-performance software for output verification

The Aldor implementation (Moreno Maza-Xie, 1999-)

- an extension of AXIOM but interoperable with other languages like C for high-performance
- Solvers as servers, limited to $\mathbb{Q}$ and finite fields, and limited user interaction
- high-performance computing

Difficulties: specialization problem

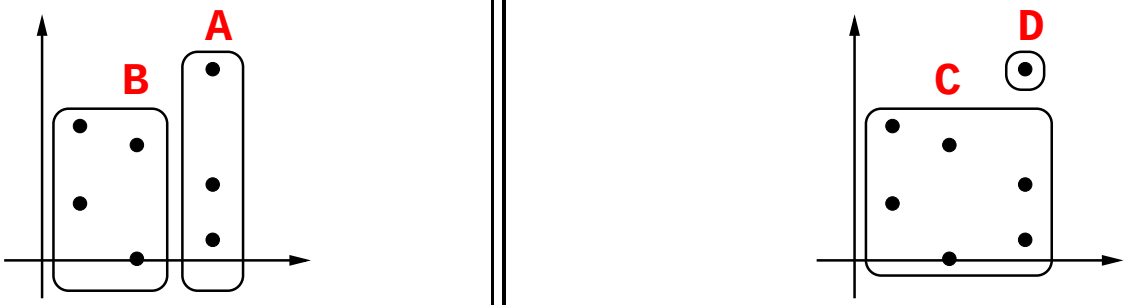
The following example illustrates the difficulties of designing a modular algorithm for triangular decompositions.

Let V be the zero-dimensional variety defined over $\mathbb{Q}$ by

$$\{326x - 10y^6 + 51y^5 + 17y^4 + 306y^2 + 102y + 34,\ y^7 + 6y^4 + 2y^3 + 12\}.$$

The unique decomposition for $x < y$ is A and B. Modulo $p = 7$, the zeros can be described by C and D.

$$A \left| \begin{array}{l} y^3 + 6 \\ x - 1 \end{array} \right. , \quad B \left| \begin{array}{l} y^2 + x \\ x^2 + 2 \end{array} \right. \quad \| \quad C \left| \begin{array}{l} y^2 + 6yx^2 + 2y + x \\ x^3 + 6x^2 + 5x + 2 \end{array} \right. , \quad D \left| \begin{array}{l} y + 6 \\ x + 6 \end{array} \right.$$

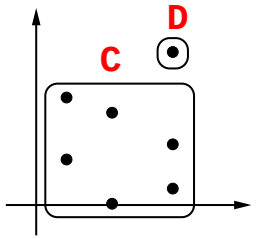


From triangular to equiprojectable decomposition: *split* + *merge* algorithm

- split: reducing what we call *critical pairs* by means of GCD computations modulo triangular sets,
- merge: reducing what we call *solvable pairs* by means of CRT computations modulo triangular sets.

Example: *split*+*merge* modulo 7

$$C \left| \begin{array}{l} C_2 = y^2 + 6yx^2 + 2y + x \\ C_1 = x^3 + 6x^2 + 5x + 2 \end{array} \right. , \quad D \left| \begin{array}{l} D_2 = y + 6 \\ D_1 = x + 6 \end{array} \right.$$

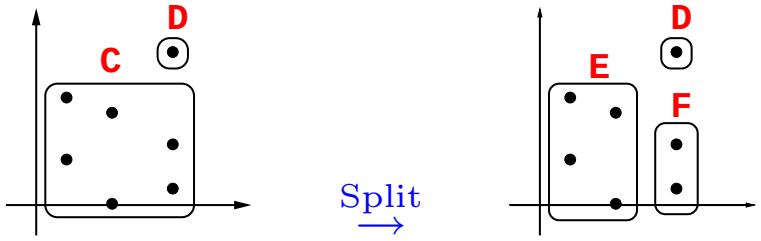


Example: *split+merge* modulo 7

$$C \left| \begin{array}{l} C_2 = y^2 + 6yx^2 + 2y + x \\ C_1 = x^3 + 6x^2 + 5x + 2 \end{array} \right. , \quad D \left| \begin{array}{l} D_2 = y + 6 \\ D_1 = x + 6 \end{array} \right.$$

$$\downarrow \quad \text{Split C : GCD} \quad \downarrow$$

$$E \left| \begin{array}{l} C_2{}' = y^2 + x \\ C_1{}' = x^2 + 5 \end{array} \right. , \quad F \left| \begin{array}{l} C_2'' = y^2 + y + 1 \\ C_1'' = x + 6 \end{array} \right. , \quad D \left| \begin{array}{l} D_2 = y + 6 \\ D_1 = x + 6 \end{array} \right.$$



Example: *split+merge* modulo 7

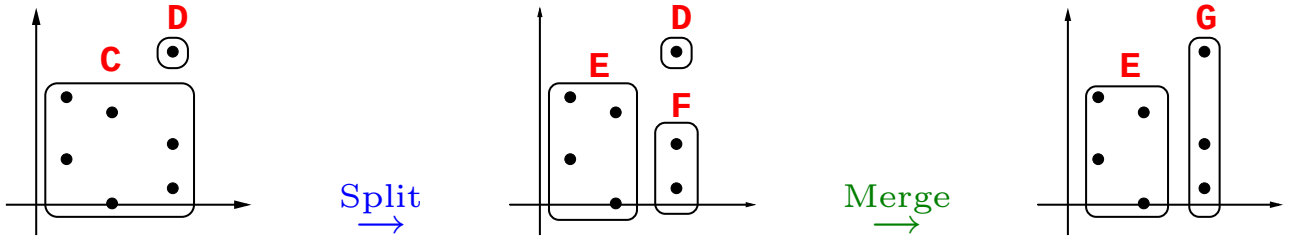
$$C\left|\begin{array}{l} C_2 = y^2 + 6yx^2 + 2y + x \\ C_1 = x^3 + 6x^2 + 5x + 2 \end{array}\right. , \quad D\left|\begin{array}{l} D_2 = y + 6 \\ D_1 = x + 6 \end{array}\right.$$

$$\downarrow \text{ Split C : GCD } \downarrow$$

$$E\left|\begin{array}{l} {C_2}' = y^2 + x \\ {C_1}' = x^2 + 5 \end{array}\right. , \quad F\left|\begin{array}{l} C_2'' = y^2 + y + 1 \\ C_1'' = x + 6 \end{array}\right. , \quad D\left|\begin{array}{l} D_2 = y + 6 \\ D_1 = x + 6 \end{array}\right.$$

$$\downarrow \text{ Merge F and D : CRT } \downarrow$$

$$E\left|\begin{array}{l} C_2' = y^2 + x \\ C_1' = x^2 + 5 \end{array}\right. , \quad G\left|\begin{array}{l} G_2 = y^3 + 6 \\ G_1 = x + 6 \end{array}\right.$$

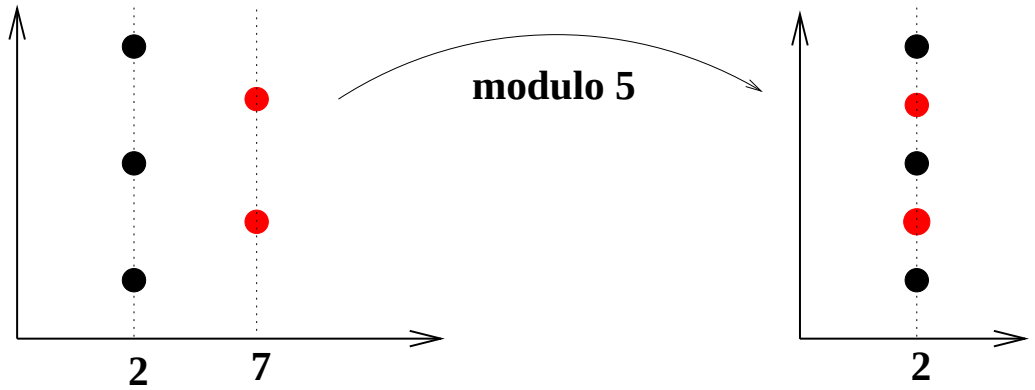


Specialization properties and modular algorithm

- **Oversimplified case:** all points in V are in $\mathbb{Q}^n$.

Theorem 3 (Dahan-Moreno Maza-Schost-Wu-Xie, 2005)
If: 1. p divides no denominator of the coordinates;
2. the cardinality of none of the projections of V decreases mod p;
then the equiprojectable decomposition specializes mod p.

- Example of specialization with a bad prime



- **The modular algorithm:**

Tr.D. mod p → Equiprojectable decomposition → Hensel lifting

Verification of solvers: using Gröbner bases

- Let $F \subset \mathbb{K}[x_1, \ldots, x_n]$ and let $[T_1, h_1], \ldots, [T_e, h_e]$ be candidate output.

- The following statements are equivalent:

(i) $\bigcup_{i=1}^{e} Z(T_i, h_i) \subseteq V(F)$,

(ii) For all $1 \leq i \leq e$, we have

$$\langle h_i F \rangle \subset \sqrt{\langle T_i \rangle}.$$

- The following statements are equivalent:

(i) $V(F) \subseteq \bigcup_{i=1}^{e} Z(T_i, h_i)$,

(ii) For all $\{i_1, \ldots, i_s\} \subseteq \{1, \ldots, e\}$, $1 \leq s \leq e$,

$$\sqrt{F \cup \{h_{i_1}, \ldots, h_{i_s}\}} \supseteq \prod_{k \in \{0,\ldots,e\} \setminus \{i_1,\ldots,i_s\}} \langle T_k \rangle.$$

$\Rightarrow$ A Gröbner basis can verify a polynomial system solver.

Verification of solvers: a naive difference algorithm

- **Input**: $[T, h]$, $[T', h']$ two solved (regular) systems.
- **Output**: solved (regular) systems $\{[T_i, h_i] \mid i = 1 \ldots e\}$ such that

$$Z(T,h) \setminus Z(T',h') = \bigcup_{i=1}^{e} Z(T_i, h_i).$$

Setting $T' = \{g_1, g_2, \cdots, g_t\}$, there is a naive approach based on:

$$\begin{aligned} Z(T,h) \setminus Z(T',h') &= Z(T,h) \bigcap \left(V(T') \cap (V(h'))^c\right)^c \\ &= Z(T,h) \bigcap \left((V(T'))^c \cup V(h')\right) \\ &= \bigcup_{i=1}^{t} Z(T, h\, g_i) \bigcup Z(h', T, h). \end{aligned}$$

- Triade operation $(p, T, h) \longmapsto \mathsf{Intersect}(p, T, h)$ solves $Z(p, T, h)$
- But this naive approach ignores the structure of T'.

Inclusion test $Z(T,h) \subseteq Z(T'h')$ reduces to difference computations!

Summary and future work

- **Our contributions**
  - Fast polynomial arithmetic over DPFs and complexity results
  - Equiprojectable decomposition, Split & Merge algorithm, modular algorithm and linear algebra over non-integral domains
  - Parallel framework in Aldor for high performance computer algebra
  - Component-level parallelization of Tr.D.
  - Efficient method for removing redundant components
  - Efficient method for verification of output
- **Future work**
  - Complexity analysis of algorithms computing Tr.D. (starting with 0-dim systems)
  - Extend the modular method to more systems and coefficient fields
  - Multi-level parallelization of Tr.D.