Intersection Formulas and Algorithms for Computing Triangular Decompositions

Changbo Chen, Marc Moreno Maza

University of Western Ontario, London, Ontario (Canada)

MACIS 2009

Plan

1. Intersection of hypersurfaces and quasi-components
2. Intersection in dimension zero
3. Intersection in positive dimension
4. Experimentation
5. Conclusion

Plan

Intersection of hypersurfaces and quasi-components (1/4)

Notation

- Let $f \in \mathbf{k}[x_1, \ldots, x_n]$ and $T \subset \mathbf{k}[x_1 < \cdots < x_n]$ be a regular chain.
- $W(T) := V(T) \setminus V(h_T)$ is the *quasi-component*, where h_T is the product of the initials of T, and $\overline{W(T)}$ its closure in Zariski topology.

Problem statement

- Compute $V(f) \cap W(T)$. This can be done as

$$V(f) \cap W(T) = Z(T_1, h_1) \cup \cdots \cup Z(T_e, h_e) \qquad (1)$$

  where (T_i, h_i) is a regular system and $Z(T_i, h_i) = W(T_i) \setminus V(h_i)$.
- Or approximate $V(f) \cap W(T)$ with regular chains $C_1, \ldots, C_e$ s. t.:

$$V(f) \cap W(T) \;\subseteq\; \cup_i W(C_i) \;\subseteq\; V(f) \cap \overline{W(T)}. \qquad (2)$$

The exact sense (1) reduces to the approximate one (2).

Intersection of hypersurfaces and quasi-components (2/4)

Motivations

- Let $(f, T) \longmapsto \mathsf{Intersect}(f, T)$ decomposing $V(f) \cap W(T)$ as:

$$V(f) \cap W(T) \subseteq \cup_i W(C_i) \subseteq V(f) \cap \overline{W(T)}.$$

- Many algorithms require $\mathsf{Intersect}(f, T)$ as a subroutine.

Example

```
dec := Intersect(x^2 + y + z -1, rc, R): map(Equations, dec, R);
                                     2
                                 [[x  + y + z - 1]]
dec := [seq(op(Intersect(x +y^2 + z -1, ts, R)), ts=dec)]: map(Equations, dec, R);
                                    2                          2
                [[x - 1 + y, z + y  - y], [x - y, z - 1 + y  + y]]

dec := [seq(op(Intersect(x + y + z^2 -1, ts, R)), ts=dec)]: map(Equations, dec, R);
                                                         2
 [[x - 1, y, z], [x, -1 + y, z], [x - z, y - z, z  + 2 z - 1], [x, y, z - 1]]
```

Intersection of hypersurfaces and quasi-components (3/4)

Incremental Solving

- Let $\{f_1, \ldots, f_m\} \subset \mathbf{k}[x_1, \cdots, x_n]$ and regular chains $U_1, \ldots, U_s \subset \mathbf{k}[x_1, \cdots, x_n]$ such that

$$V(f_1, \ldots, f_{m-1}) = W(U_1) \cup, \cdots, \cup W(U_s).$$

- The the union of the Intersect(f_m, U_i) decomposes $V(f_1, \ldots, f_m)$
- (D. Lazard 91) proposes the principle but a different Intersect(f, T).
- (M. Moreno Maza 00) introduces regular GCDs and gives a complete incremental algorithm based on regular chains.
- Computing Intersect(f, T) reduces to "generalized" polynomial GCD computations, leading to modular methods and fast arithmetic.
- Incremental algorithm in other polynomial system solving algorithms: F5 by J.-C. Faugère and Kronecker by G. Lecerf (and the TERA group).

Intersection of hypersurfaces and quasi-components (4/4)

Computational challenges

- Algorithms for Intersect(f, T) follow a *projection-extension* scheme
- The *projection* step may introduce components which cannot be extended: not our purpose today.
- The *extension* step may recompute things that were "essentially" computed during the projection step: today's subject.

Our contributions

- Ensure that the extension step recycles from the projection step.
- From Intersect(f, T), derive decompositions in the sense of (Kalkbrener, 1991)

$$V(f_1, \ldots, f_m) = \overline{W(U_1)} \cup, \cdots, \cup \overline{W(U_s)}.$$

- Report on a preliminary Maple implementation.

Plan

Zero-dimensional Regular Chains

Definition

$T \subset \mathbf{k}[x_1 < \cdots < x_n] \setminus \mathbf{k}$ is a *zero-dimensional regular chain* if

- $T = \{T_1(x_1), T_2(x_1, x_2), \ldots, T_n(x_1, \ldots, x_n)\}$,
- $\mathrm{lc}(T_i, x_i)$ is invertible modulo $\langle T_1, \ldots, T_{i-1} \rangle$ for $1 < i \leq n$,

Additional Properties

1. *Reduced*: $\deg(T_i, x_j) < \deg(T_j, x_j)$ for $1 \leq j < i \leq n$.
2. *Squarefree*: T_i and $\frac{\partial T_i}{\partial x_i}$ are relatively prime modulo $\langle T_1, \ldots, T_{i-1} \rangle$ for $1 \leq i \leq n$,
3. *Normalized*: $\mathrm{lc}(T_i, x_i) = 1$ for $1 \leq i \leq n$.

Remark

If T is a zero-dimensional regular chain then $W(T) = V(T)$.

Intersection and Invertibility Test

Intersection

For $T \subset \mathbf{k}[x_1 < \cdots < x_n]$ zero-dimensional regular chain and $p \in \mathbf{k}[x_1 < \cdots < x_n]$, the call Intersect$(f, T)$ returns regular chains $C_1, \ldots, C_e \subset \mathbf{k}[x_1, \cdots, x_n]$ such that

$$V(T) \cap V(p) = V(C_1) \cup \cdots \cup V(C_e).$$

Invertibility Test

For $T \subset \mathbf{k}[x_1 < \cdots < x_n]$ zero-dimensional regular chain and $p \in \mathbf{k}[x_1 < \cdots < x_n]$, the call RegularizeDim0(p, T) returns regular chains $C_1, \ldots, C_e \subset \mathbf{k}[x_1, \cdots, x_n]$ such that

- $V(T) = V(C_1) \cup \cdots \cup V(C_e)$,
- $V(C_i) \subset V(p)$ or $V(C_i) \cap V(p) = \emptyset$ for all $1 \leq i \leq e$.

Invertibility Test

Underlying tools

- **Proposition**: p invertible modulo $\langle T \rangle$ iff resultant$(T, p) \neq 0$.
- Regular GCDs, which can be computed via subresultants.

Principle of the algorithm

- If $\mathrm{mvar}(p) = v$ then compute $src :=$ SubresultantChain(p, T_v, v) and $r :=$ resultant(src)
- Call RegularizeDim0(r, T) recursively; let $D \in$ RegularizeDim0(r, T).
- If r is invertible modulo $\langle D \rangle$ then p is invertible modulo $\langle D \rangle$ too.
- If resultant$(T_v, p, v) \in \langle D \rangle$ then T_v, p have a regular GCD g modulo $\langle D \rangle$ obtained from src; then T splits since $T_v \equiv g \frac{T_v}{g} \mod T_{<v}$.
- (X. Li, M.M.M. & W. Pan, ISSAC 2009)

Algorithm 1: RegularizeDim0(p, T)

Input: a polynomial p and a zero-dimensional regular chain T of $\mathbf{k}[x_1 < \cdots < x_n]$

Output: regular chains $\{T_1, \ldots, T_e\}$ s.t. $(p, T) \longrightarrow T_1, \ldots, T_e$ and p is zero or invertible modulo $\langle T_i \rangle$.

```
begin
    if p ∈ k or p ∈ ⟨T⟩ or T = ∅ then return {T}
    v := mvar(p)
    for C ∈ RegularizeDim0(init(p), T) do
        if init(p) ∈ ⟨C⟩ then output RegularizeDim0(tail(p), C); next
        src := SubresultantChain(p, T_v, v); r := resultant(src)
        for D ∈ RegularizeDim0(r, T_{<v}) do
            if r ∉ ⟨D⟩ then output D + T_{≥v}
            else
                for (g, E) ∈ RegularGcd(p, T_v, v, src, D) do
                    if mdeg(g) = mdeg(T_v) then output E + T_{≥v}; next
                    output E + g + T_{>v}
                    q := pquo(T_v, g)
                    output RegularizeDim0(p, E + q + T_{>v})
end
```

Regularity Test (= Saturation)

d_1	d_2	d_3	Regularize	Fast Regularize	Magma
2	2	3	0.032	0.004	0.010
3	4	6	0.160	0.016	0.020
4	6	9	0.404	0.024	0.060
5	8	12	>100	0.129	0.330
6	10	15	>100	0.272	1.300
7	12	18	>100	0.704	5.100
8	14	21	>100	1.276	14.530
9	16	24	>100	5.836	40.770
10	18	27	>100	9.332	107.280
11	20	30	>100	15.904	229.950
12	22	33	>100	33.146	493.490

Table: Generic dense 3-variable.

- In the non-generic case, both gaps are even larger.
- "Fast Regularize" means `RegularizeDim0` in `Maple` 13.

Plan

Regular Chains

Notations

Let $T \subset \mathbf{k}[x_1 < \cdots < x_n] \setminus \mathbf{k}$ be a *triangular set*,
Let $\mathrm{mvar}(T) := \{\mathrm{mvar}(t) \mid t \in T\}$, $\mathrm{init}(t) := \mathrm{lc}(t, \mathrm{mvar}(t))$ for all $t \in T$, and $h_T := \prod_{t \in T} \mathrm{init}(t)$.

Saturated Ideal

The *saturated ideal* of T is the ideal of $\mathbf{k}[x_1 < \cdots < x_n]$

$$\mathrm{sat}(T) := \langle T \rangle : (h_T)^{\infty}.$$

Regular Chain

T is a *regular chain* if for each $v \in \mathrm{mvar}(T)$ the initial of T_v is regular modulo $\mathrm{sat}(T_{<v})$.

Regular GCDs

Notations

Let T be a regular chain and f be a non-constant polynomial, wuth $v = \mathrm{mvar}(p)$, s. t. $v \in \mathrm{mvar}(T)$ and $\mathrm{init}(f)$ is regular modulo $\mathrm{sat}(T)$.

Definition: Base case

A polynomial g is a regular GCD of f, T_v modulo $\mathrm{sat}(T_{<v})$ if

- $\mathrm{lc}(g, v)$ is regular modulo $\mathrm{sat}(T_{<v})$.
- $g \in \langle f, T_v, \mathrm{sat}(T_{<v}) \rangle$
- $\deg(g, v) > 0 \quad \Rightarrow \quad \mathrm{prem}_v(f, g), \mathrm{prem}_v(T_v, g) \in \mathrm{sat}(T_{<v})$.

Definition: General case

One can compute regular chains $C^1, \ldots, C^e$ such that we have

- $W(T) \;\subseteq\; \cup_i W(C^i) \;\subseteq\; \overline{W(T)}$.
- $\forall i$, if $\mid C^i \mid = \mid T \mid$ then f, C^i_v admit a regular GCD $\bmod\ \mathrm{sat}(C^i_{<v})$.

Computing Regular GCDs

Existence

Assume $\mathrm{mvar}(p) = v$, with v algebraic w.r.t. T. Assume $\mathrm{init}(p)$ is regular modulo $\mathrm{sat}(T_{<v})$ and resultant$(p, T_v, v) \in \mathrm{sat}(T_{<v})$. Then p and T_v admit a regular GCD of positive degree modulo $\mathrm{sat}(T_{<v})$.

Criterion

Let p, T be as above. Let S_j be the subresultant of index j of p, T_v as polynomials in v. Assume that there exists an index d such that

- $S_0 :=$ resultant$(p, T_v, v) \in \mathrm{sat}(T_{<v})$,
- $S_j \in \mathrm{sat}(T_{<v})$ for all $0 \leq j < d$.
- $\mathrm{lc}(S_d, v)$ is regular modulo. $\mathrm{sat}(T_{<v})$,
- $\mathrm{coeff}(S_j, v^j)$ is regular or zero modulo. $\mathrm{sat}(T_{<v})$, for all S_j with $j > d$.

Then S_d is a regular GCD of p and T_v modulo $\mathrm{sat}(T_{<v})$.

Intersection Algorithm

Principle of the algorithm

Projection: Essentially computes resultants and stores the corresponding subresultant chains.

Extension: All needed regular GCDs are derived from the stored subresultant chains. The main cost is reduced to regularity test.

Challenges

- In positive dimension, splitting a regular chain can bring components of lower dimension.
- This leads to many corner cases. In particular regularity test calls the intersection algorithm.
- In positive dimension, the intersection and regularity test are no longer equivalent: p regular $\mathrm{sat}(T)$ does **not** imply $V(p) \cap \overline{W(T)} = \emptyset$.

Function Calls Diagram

We have the following diagram on the recursive calls of the algorithms to each other.

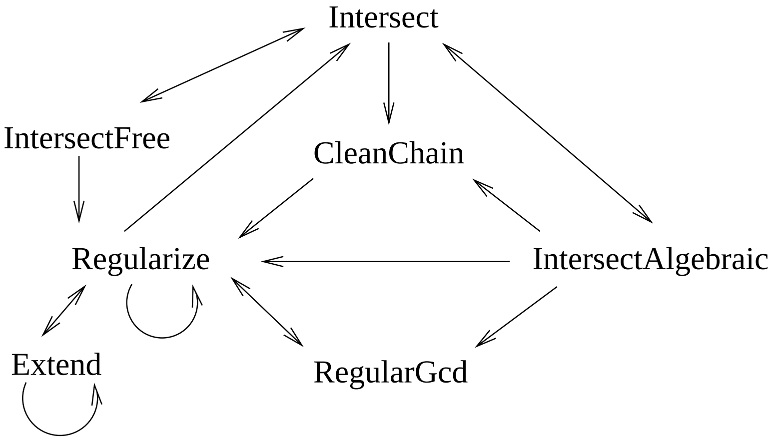

Figure: Subalgorithm Dependencies

Algorithm 2: Intersect(p, T)

begin
- $p = 0$ **return** T; $p \in \mathbf{k}$ **return** $\emptyset$
- $r \leftarrow p$; $P \leftarrow \{r\}$; $S \leftarrow \emptyset$
- **while** $v \leftarrow \mathrm{mvar}(r) \in \mathrm{mvar}(T)$ **do**
  - $src \leftarrow$ SubresultantChain(r, T_v, v, R); $S \leftarrow S \cup src$
  - $r \leftarrow$ resultant(src)
  - $r = 0 \Longrightarrow$ break; $r \in \mathbf{k}$ **return** $\emptyset$
  - $P \leftarrow P \cup r$
- $\mathfrak{L} \leftarrow \{\varnothing\}$; $\mathfrak{L}' \leftarrow \emptyset$; $i \leftarrow 1$
- **while** $i \leq n$ **do**
  - **for** $C \in \mathfrak{L}$ **do**
    - $x_i \notin \mathrm{mvar}(P) \Longrightarrow \mathfrak{L}' \leftarrow \mathfrak{L}' \cup \{C \cup T_i\}$
    - $x_i \notin \mathrm{mvar}(T) \Longrightarrow \mathfrak{L}' \leftarrow \mathfrak{L}' \cup \{C \cup P_i\}$ $\cup$ Intersect$(\{\mathrm{tail}(P_i), \mathrm{init}(P_i)\}, C)$
    - **for** $(g, D) \in$ RegularGcd(P_i, T_i, x_i, S_i, C) **do**
      - $\mathfrak{L}' \leftarrow \mathfrak{L}' \cup \{D \cup g\}$
  - $\mathfrak{L} \leftarrow \mathfrak{L}'$; $\mathfrak{L}' \leftarrow \emptyset$; $i \leftarrow i + 1$
- return $\mathfrak{L}$

end

Computing Kalkbrener Decompositions

How `Triangularize` computes them in Maple 13

- On input polynomials $f_1, \ldots, f_m \in \mathbf{k}[x_1, \ldots, x_n]$ we want to compute regular chains $U_1, \ldots, U_s \subset \mathbf{k}[x_1, \ldots, x_n]$ such that

$$V(f_1, \ldots, f_m) = \overline{W(U_1)} \cup, \cdots, \cup \overline{W(U_s)}.$$

- To do so, apply Krull's dimension Theorem (plus a few other tricks) to a decomposition of the form

$$V(f_1, \ldots, f_m) = W(C_1) \cup, \cdots, \cup W(C_e).$$

How `Triangularize` computes them now

- The maximum height of a regular chain is passed to all sub-algorithms. The extension step of Intersect takes advantage of it.
- The decomposition tree is pruned dynamically: only regular chains of height $h \leq m$ are generated.

Plan

Triangularize (Lazard mode) in Maple 13 and NOW

Sys	#v	#e	characteristic	Maple13	NOW
eco7	7	7	387799	206.928	8.776
Methan61	10	10	450367	488.722	25.189
Reimer-4	4	4	55313	434.603	23.545
Uteshev-Bikker	4	4	7841	582.024	33.442
gametwo5	5	5	159223	1258.930	56.311
nld-3-5	5	5	3323702051	316.803	16.573
Cassou-Nogues	4	4	3265548718319	> 3600	84.865
nld-4-5	5	5	105451527661	> 3600	116.791
Leykin-1	8	6	0	89.885	5.396
Liu-Lorenz-Li	5	4	0	149.541	1.764
Pavelle	8	4	0	> 3600	14.332
Morgenstein	9	5	0	> 3600	12.040
collins-jsc02	5	4	0	> 3600	1.192
hereman-8.8	8	6	0	> 3600	19.601
f-744	12	12	0	> 3600	35.394
DonatiTraverso-rev	4	3	0	> 3600	> 3600

Triangularize in Kalkbrener and Lazard Modes

Sys	#v	#e	Lazard	Kalkbrener
Pavelle	8	4	14.332	2.677
Pappus	12	6	14.288	2.445
Morgenstein	9	5	12.040	1.728
hereman-8.8	8	6	19.601	17.613
f-744	12	12	35.394	38.058
8-3-config-Li	12	7	26.425	5.396
Lazard-ascm2001	7	3	12.188	0.576
Lichtblau	3	2	243.251	0.660
Cinquin-Demongeot-3-3	4	3	157.705	0.572
xy-5-7-2	6	3	> 3600	3.284
Cinquin-Demongeot-3-4	4	3	> 3600	5.524
Cheaters-homotopy-easy	7	3	> 3600	0.356
Cheaters-homotopy-hard	7	2	> 3600	0.308
4corps-1parameter-homog	4	3	> 3600	599.509
DonatiTraverso-rev	4	3	> 3600	6.192
Bezier	5	3	> 3600	> 3600

ComprehensiveTriangularize in Maple 13 and NOW

Sys	PCTD (13)	PCTD (NOW)	CTD (13)	CTD (NOW)
chemical	283.317	60.719	293.626	59.007
Alonso-Li	>3600	4.312	>3600	25.057
Morgenstein	>3600	3027.337	>3600	>3600
MontesS14	7.004	0.608	7.420	0.640
Wang168	709.424	8.148	708.588	8.896
collins-jsc02	>3600	1.576	>3600	69.484
Leykin-1	91.149	5.452	92.489	6.896
genLinSyst-3-3	3.964	3.384	32.794	31.953
AlKashiSinus	3.572	3.364	7.320	5.984
Pavelle	>3600	106.998	>3600	845.564
Pappus	24.177	15.876	227.442	306.231
hereman-8.8	>3600	239.250	>3600	236.046
f-744	>3600	>3600	>3600	>3600
8-3-config-Li	95.537	31.077	186.507	217.713
Lazard-ascm2001	261.692	16.629	356.970	84.893
Lichtblau	>3600	>3600	>3600	>3600
C-D-3-3	>3600	127.587	>3600	126.827

- dim : dimension of the regular chain
- mdeg: the product of the main degrees of polynomials in the regular chain
- cdeg: the maximum degree of the coefficients of polynomials in the regular chain w.r.t all main variables of the chain
- clength: the maximum height of the integer coefficients of polynomials of the regular chain w.r.t all its variables.

Sys	Lazard (dim, mdeg, cdeg, clength)	Kalkbrener (dim, mdeg, cdeg, clength)
Lichtblau	(1, 11, 11, 113), (0, 88, 0, 3597), $(0,1,0,1)^2$	(1, 11, 11, 113)
C-D-3-3	$(1,24,4,5)$, $(1,9,3,4)$ $(1,9,2,5)$, $(1,9,2,4)$ $(1,4,1,2)$ $(0,18,0,2)$, $(0,9,0,2)$ $(0,6,0,5)$, $(0,4,0,1)^3$ $(0,2,0,1)^7$, $(0,1,0,1)^4$	(1, 24, 4, 5), (1, 9, 3, 4) (1, 9, 2, 5), (1, 9, 2, 4) (1, 4, 1, 2)

Triangularize versus Groebner[Basis] (lex order) in Maple

sys	Triangularize		Groebner[Basis]	
	time	length	time	length
Pavelle	2.600	1079	1.784	17990
Hairer-2-BGK	1.956	364	24.337	12126
Wang168	0.760	800	11.116	7935
Collins-jsc02	0.260	1296	868.230	3455570
Leykin-1	3.700	531	98.222	24717
Hereman-8.8	17.613	11646	> 3600	N/A
f-744	37.326	4510	31.497	102085
8-3-config-Li	5.396	1390	106.954	67974
Lichtblau	0.660	5241	125.123	6600096
Cinquin-Demongeot-3-3	0.572	896	63.871	1652065
Cinquin-Demongeot-3-4	5.524	2328	> 3600	N/A
Cheaters-homotopy-easy	0.356	290	3527.400	26387447
4corps-1parameter-homog	599.509	30740	> 3600	N/A
Cheaters-homotopy-hard	0.308	327	3409.753	8662753
DonatiTraverso-rev	6.192	2484	154.177	2312043

Plan

Concluding Remarks

- We have discussed how to compute $V(f) \cap W(T)$.
- Applications are: incremental solving, operations on constructible sets.
- One highlight: the extension step should be projection-aware.
- I have hidden a lot of technical difficulties and focussed on one technique: recycling subresultant chains.
- We have replaced our old Intersect(f, T) by the new one in our **Maple interpreted code**. Then, we have observed large speedup factors (without using modular methods nor fast arithmetic yet).
- We have improved both the *Kalkbrener* and *Lazard* modes.
- We have observed favorable output size matching (Dahan, Kadri & Schost, 2009).
- Work in progress: integrate the `FasyArithmeticTools` of the `Modpn` library into Intersect(f, T) and port the whole thing to C code.