Computing with Constructible Sets

Liyun Li & Yuzhen Xie

Joint work with
Changbo Chen, Marc Moreno Maza, Wei Pan

ORCCA, UWO

MOCAA M^3 Workshop, May 9, 2008

Outline

- Constructible set and removing redundancy
- Difference: $(A, B) \longmapsto A \setminus B$
- PairRefine: $(A, B) \longmapsto (A \setminus B,\ A \cap B, B \setminus A)$
- MPD: C (with redundancy) $\longmapsto$ C (without redundancy)
- SMPD:

  $C = \bigcup C_i \quad \longmapsto \quad C = \bigcup D_j$

  (with redundancy) (without redundancy and the $D_j's$ refine the $C_i's$)
- Complexity Analysis
- Experimental comparison (different algorithms for SMPD)

What is a Constructible Set?

Definition (Constructible Set)

A *constructible set* of $\overline{\mathbb{K}}^n$ is a finite union

$$(A_1 \setminus B_1) \cup \cdots \cup (A_e \setminus B_e)$$

where $A_i's$ and $B_i's$ are algebraic varieties in $\overline{\mathbb{K}}^n$.

Definition (Regular System)

A pair $[T, h]$ is a *regular system* if T is a regular chain, and $h \in \mathbb{K}[X]$ is regular with respect to $\mathrm{sat}(T)$. The zero set $Z(T, h)$ given by $[T, h]$ is $W(T) \setminus V(h)$.

Example (Regular Systems)

$$(Yes) \begin{cases} ax^2 + bx + c = 0 \\ a(b^2 - 4ac) \neq 0 \end{cases} \qquad (No) \begin{cases} x^2 - 2xy + t = 0 \\ y^2 - t = 0 \\ x - y \neq 0 \end{cases}$$

Representation of Constructible Sets

Example (Constructible Set)

For what value of a, b, c, does the equation

$$ax^2 + bx + c = 0$$

have solutions over $\mathbb{C}$?

- when a is not zero; $rs_1 = [a \neq 0]$
- when a is zero but b is not; $rs_2 = [a = 0, b \neq 0]$
- when a, b, c are all zero. $rs_3 = [a = 0, b = 0, c = 0]$

$cs = \{rs_1, rs_2, rs_3\}$ describes the answer.

Another Example

- **Example**: given two elliptic curves in the complex plane of coordinates (x, y): $g_1(x, y) = 0$ and $g_2(x, y) = 0$, where

$$\begin{aligned} g_1(x, y) &= x^3 + a_1 x - y^2 + 1, \\ g_2(x, y) &= x^3 + a_2 x - y^2 + 1 \end{aligned}$$

In invariant theory, a classical question is whether there exists a
linear fractional map from the first curve to the second one:

$$f : (x, y) \mapsto \left(\frac{A\,x + B\,y + C}{G\,x + H\,y + K}, \frac{D\,x + E\,y + F}{G\,x + H\,y + K} \right)$$

Another Example

- This problem can be turned into a parametric system:

$$g_1(x,y) - (Gx + Hy + K)^3 g_2(f(x,y)) = 0.$$

$$\begin{cases} 1 - K^3 & = 0 \\ -a_2 AK^2 + a_1 - 3GK^2 & = 0 \\ -3HK^2 - a_2 BK^2 & = 0 \\ GD^2 - a_2 G^2 A - A^3 - G^3 + 1 & = 0 \\ -3H^2K + E^2K - 1 - 2a_2 BHK & = 0 \\ -3G^2K - 2a_2 GAK + D^2K & = 0 \\ GE^2 - 2a_2 GBH - a_2 AH^2 - 3AB^2 - 3GH^2 + 2DEH & = 0 \\ E^2H - H^3 - a_2 BH^2 - B^3 & = 0 \\ D^2H - 3G^2H + 2GDE - 2a_2 GAH - 3A^2B - a_2 G^2B & = 0 \\ -3GHK - a_2 AHK - a_2 GBK + DEK & = 0 \end{cases}$$

- For which parameter values of a_1, a_2 does this system have solutions?

Another Example

- The output produced by the command `ComprehensiveTriangularize` of the module `ParametricSystemTools` consists of 11 regular chains $[T_1, \ldots, T_{11}]$ and 3 constructible sets C_1, C_2 and C_3.

$$\begin{array}{lcl} C_1 & : & a_1^3 = a_2^3 = 9 \\ C_2 & : & a_1 = a_2 = 0 \\ C_3 & : & a_1^3 = a_2^3,\ a_2 \neq 0,\ a_2^3 \neq 9\ . \end{array}$$

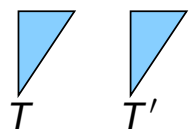
- The union of C_1, C_2, C_3 is the answer to our question: for which parameter values does the input system have solutions?

Redundancy in Computing with Constructible Sets

- Redundancy in a single constructible set
  - Two regular systems have a common part.
  - Remove redundancy: make regular systems pairwise disjoint (MPD)
- Redundancy in a list of constructible sets
  - Some zeroes appear in more than one constructible sets.
- Building block: compute the difference of two regular systems.

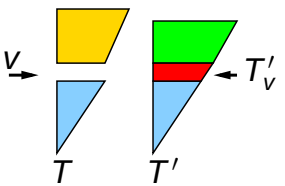
Sketch of Difference Algorithm to compute $V(T) \setminus V(T')$ by exploiting the triangular structure level by level.

Case 1:



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

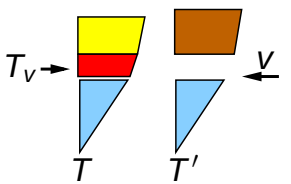
Case 2:



Output $[T, T'_v]$ and

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

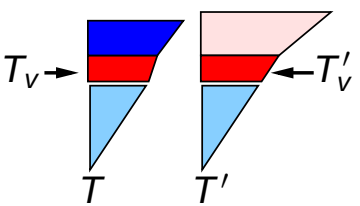
Case 3:



Output $\texttt{Difference}$

$(T, V(T_v) \cap V(T'))$;

Case 4:



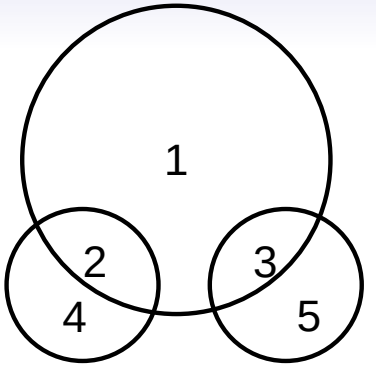
- $g = \mathrm{GCD}(T_v, T'_v, T_{<v})$; • $g \in \mathbb{K} \Rightarrow$ Output $[T, 1]$;
- $\mathrm{mvar}(g) < v \Rightarrow$ Output $[T, g]$, $\texttt{Difference}(V(g) \cap V(T), T')$;
- Output $\texttt{Difference}(T_{<v} \cup \{g\} \cup T_{>v}, T')$;
- Output $\texttt{Difference}(T_{<v} \cup \{T_v/g\}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

Algorithm 2 MPD

Input: a list L of regular systems

Output: a pairwise disjoint representation of L

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $d \leftarrow L[n]$
6: $L^* \leftarrow \text{MPD}(L[1, \ldots, n-1])$
7: **for** $\ell' \in L^*$ **do**
8: $d \leftarrow \text{Difference}(d, \ell')$
9: **end for**
10: **return** $d \cup L^*$
11: **end if**



$A = \{1, 2, 3\}$,

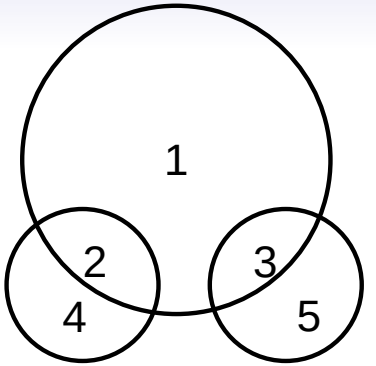
$B = \boxed{\{2,4\}}, C = \boxed{\{3,5\}}$.

Algorithm 2 MPD

Input: a list L of regular systems

Output: a pairwise disjoint representation of L

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $d \leftarrow L[n]$
6: $L^* \leftarrow$MPD$(L[1, \ldots, n-1])$
7: **for** $\ell' \in L^*$ **do**
8: $d \leftarrow$ Difference(d, ℓ')
9: **end for**
10: **return** $d \cup L^*$
11: **end if**



$A = \{1, 2, 3\}$,

$B = \boxed{\{2,4\}}, C = \boxed{\{3,5\}}$.

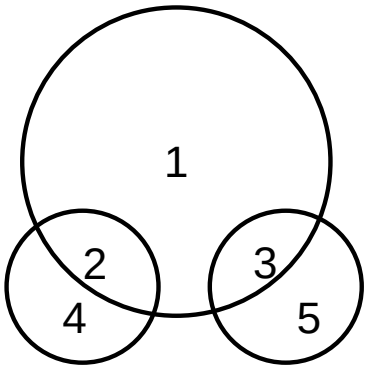
$A = A \setminus B = \{1, 3\}$

Algorithm 2 MPD

Input: a list L of monic square-free zero-dimensional regular chains

Output: a pairwise disjoint representation of L

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $d \leftarrow L[n]$
6: $L^* \leftarrow \text{MPD}(L[1, \ldots, n-1])$
7: **for** $\ell' \in L^*$ **do**
8: $d \leftarrow \text{Difference}(d, \ell')$
9: **end for**
10: **return** $d \cup L^*$
11: **end if**



$A = \{1, 2, 3\}$,

$B = \boxed{\{2,4\}}, C = \boxed{\{3,5\}}$.

$A = A \setminus B = \{1, 3\}$

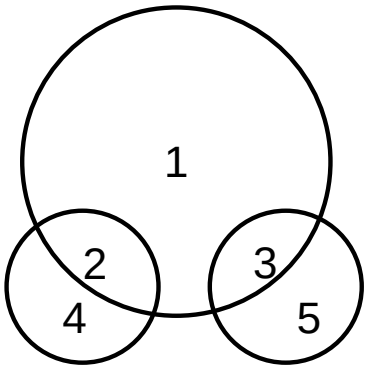
$A = A \setminus C = \boxed{\{1\}}$

Algorithm 2 MPD

Input: a list L of monic square-free zero-dimensional regular chains

Output: a pairwise disjoint representation of L

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $d \leftarrow L[n]$
6: $L^* \leftarrow \text{MPD}(L[1, \ldots, n-1])$
7: **for** $\ell' \in L^*$ **do**
8: $d \leftarrow \text{Difference}(d, \ell')$
9: **end for**
10: **return** $d \cup L^*$
11: **end if**



$A = \{1, 2, 3\}$,

$B = \boxed{\{2,4\}}, C = \boxed{\{3,5\}}$.

$A = A \setminus B = \{1, 3\}$

$A = A \setminus C = \boxed{\{1\}}$

$\text{MPD}(A, B, C) = \{1\}, \{2, 4\}, \{3, 5\}$

RCPairRefine

In dimension 0, we can do better.

- RCPairRefine performs Difference in a simple case.
- Compute two-side differences and intersection **in one pass**.

Example

$$\text{RCPairRefine}([x = 0, y(y+1) = 0], [x = 0, y(y+2) = 0])$$

outputs

$$\underbrace{[x = 0, y + 1 = 0]}_{\textit{difference}}, \quad \underbrace{[x = 0, y = 0]}_{\textit{intersection}}, \quad \underbrace{[x = 0, y + 2 = 0]}_{\textit{difference}}$$

RCPairRefine Algorithm

Algorithm 1 RCPairRefine

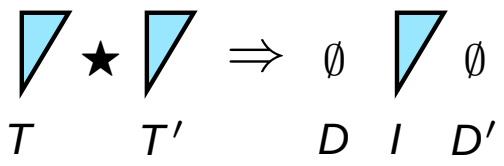
Input: two monic squarefree zero-dimensional regular chains T and T'

Output: three constructible sets D, I and D', such that $V(T) \setminus V(T') = Z(D)$, $V(T) \cap V(T') = Z(I)$ and $V(T') \setminus V(T) = Z(D')$

1: **if** $T = T'$ **then**
2: **return** $\emptyset,\ [T],\ \emptyset$
3: **else**
4: $D \leftarrow \emptyset;\ I \leftarrow \emptyset;\ D' \leftarrow \emptyset$
5: Let v be the largest variable s.t. $T_{<v} = T'_{<v}$
6: **for** $(g, G) \in \mathrm{GCD}(T_v, T'_v, T_{<v})$ **do**
7: **if** $g \in \mathbb{K}$ **or** $\mathrm{mvar}(g) < v$ **then**
8: $T_q \leftarrow G \cup \{T_v\} \cup T_{>v}$;
9: $T'_q \leftarrow G \cup \{T'_v\} \cup T'_{>v}$;
10: $D \leftarrow D \cup T_q$;
11: $D' \leftarrow D' \cup T'_q$
12: **else**
13: $q \leftarrow \mathrm{pquo}(T_v, g, G)$;
14: $q' \leftarrow \mathrm{pquo}(T'_v, g, G)$;
15: $T_g \leftarrow G \cup \{g\} \cup T_{>v}$;
16: $T'_g \leftarrow G \cup \{g\} \cup T'_{>v}$
17: **if** $\mathrm{mvar}(q) = v$ **then**
18: $T_q \leftarrow G \cup \{q\} \cup T_{>v}$;
19: $D \leftarrow D \cup T_q$
20: **end if**
21: **if** $\mathrm{mvar}(q') = v$ **then**
22: $T'_q \leftarrow G \cup \{q'\} \cup T'_{>v}$;
23: $D' \leftarrow D' \cup T'_q$
24: **end if**
25: $W, J, W' \leftarrow$ RCPairRefine(T_g, T'_g);
26: $D \leftarrow D \cup W$;
27: $I \leftarrow I \cup J$;
28: $D' \leftarrow D' \cup W'$
29: **end if**
30: **end for**
31: **return** D, I, D'
32: **end if**

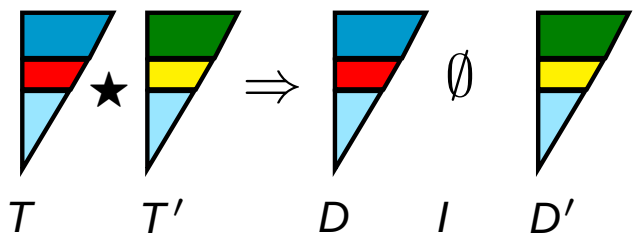
RCPairRefine Algorithm

Case 1: $T = T'$



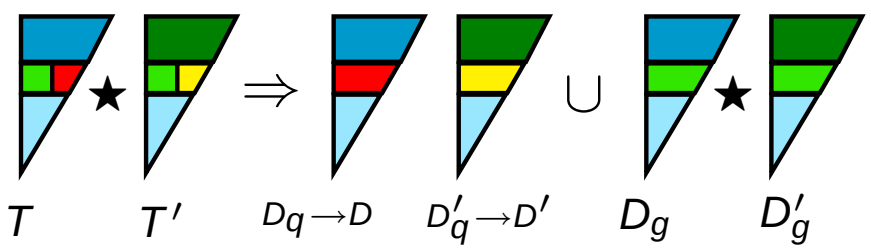
Case 2: $g = GCD(T_v, T'_v, T_{<v})$

$g \in \mathbb{K}$ or $\mathrm{mvar}(g) < v$



Case 3:

$g = GCD(T_v, T'_v, T_{<v})$; $\mathrm{mvar}(g) = v$



$$
\begin{aligned}
D &= T \setminus T' \\
I &= T \cap T' \\
D' &= T' \setminus T
\end{aligned}
$$

Irredundant Representation of a Family of Constructible Sets by *Symmetrically Make Pairwise Disjoint* (SMPD)

- Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of constructible sets.
- An intersection-free basis of $\mathcal{C}$: $\mathcal{D} = \{D_1, \ldots, D_n\}$ satisfies
  (1) $D_i \cap D_j = \emptyset$ for $1 \leq i \neq j \leq n$,
  (2) each C_i can be uniquely written as a union of some D_j's.

- CSPairRefine $= \mathrm{SMPD}(C_1, C_2) = C_1 \backslash C_2,\ C_1 \cap C_2,\ C_2 \backslash C_1$

- Based on RCPairRefine, CSPairRefine can be deduced naturally.

Example

$\mathrm{SMPD}(\{[x = 0]\},\ \{[y = 0]\})$ outputs:

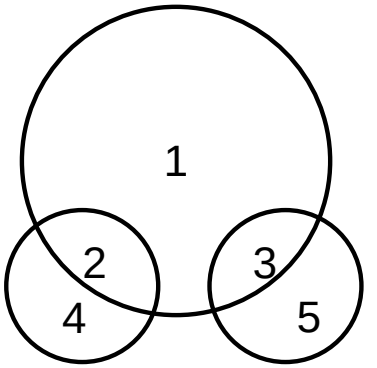
$\{[x = 0,\ y \neq 0]\},\ \{[x = 0, y = 0]\},\ \{[x \neq 0,\ y = 0]\}$

Algorithm 3 BachSMPD

Input: a list L of constructible sets
with each consisting of a
family of monic squarefree
zero-dimensional regular chains

Output: an intersection-free basis of L

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $I \leftarrow \emptyset;\ D' \leftarrow \emptyset;\ d \leftarrow L[n]$
6: $L^* \leftarrow \text{BachSMPD}(L[1, \ldots, n-1])$
7: **for** $l' \in L^*$ **do**
8: $d, i, d' \leftarrow \text{CSPairRefine}(d, l')$
9: $I \leftarrow I \cup i;\ D' \leftarrow D' \cup d'$
10: **end for**
11: **return** $\{d\} \cup I \cup D'$
12: **end if**



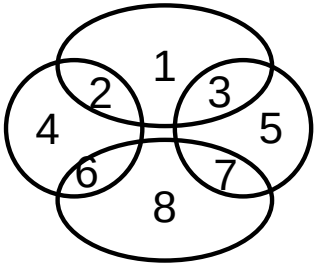
$$
\begin{aligned}
A &= \{1,2,3\},\\
B &= \{2,4\},\\
C &= \{3,5\}.\\
A \bigstar B &= \{1,3\},\{2\},\{4\}\\
d \bigstar C &= \{1\},\{3\},\{5\}
\end{aligned}
$$

$$
\text{SMPD}(A,B,C) = \{1\},\{2\},\{4\},\{3\},\{5\}
$$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $I \leftarrow I \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup I \cup L_2$
15: **end if**

Merge: Line 8-line 13

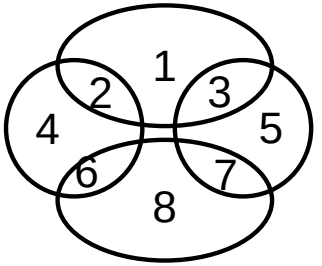


$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $I \leftarrow I \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup I \cup L_2$
15: **end if**

Merge: Line 8-line 13



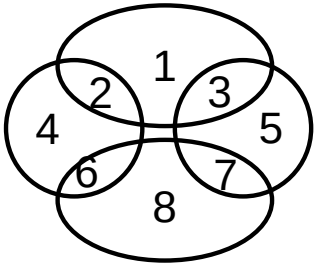
$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

$A \bigstar C = \{4, 6\}, \boxed{\{2\}}, \{1, 3\}$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $l \leftarrow l \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup l \cup L_2$
15: **end if**

Merge: Line 8-line 13



$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

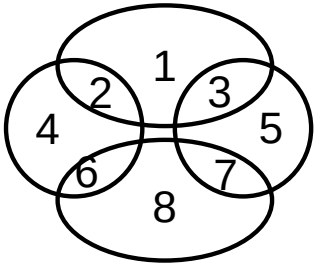
$A \bigstar C = \{4, 6\}, \boxed{\{2\}}, \{1, 3\}$

$A \bigstar D = \boxed{\{4\}}, \boxed{\{6\}}, \{7, 8\}$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $I \leftarrow I \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup I \cup L_2$
15: **end if**

Merge: Line 8-line 13



$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

$A \bigstar C = \{4, 6\}, \boxed{\{2\}}, \{1, 3\}$

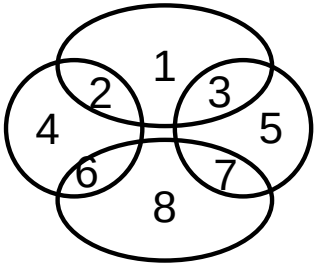
$A \bigstar D = \boxed{\{4\}}, \boxed{\{6\}}, \{7, 8\}$

$B \bigstar C = \{5, 7\}, \boxed{\{3\}}, \boxed{\{1\}}$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $I \leftarrow I \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup I \cup L_2$
15: **end if**

Merge: Line 8-line 13



$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

$A \bigstar C = \{4, 6\}, \boxed{\{2\}}, \{1, 3\}$

$A \bigstar D = \boxed{\{4\}}, \boxed{\{6\}}, \{7, 8\}$

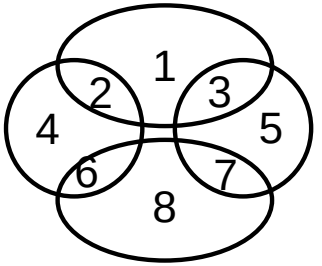
$B \bigstar C = \{5, 7\}, \boxed{\{3\}}, \boxed{\{1\}}$

$B \bigstar D = \boxed{\{5\}}, \boxed{\{7\}}, \boxed{\{8\}}$

Algorithm 4 DCSMPD

1: $n \leftarrow |L|$
2: **if** $n < 2$ **then**
3: **return** L
4: **else**
5: $z \leftarrow \lfloor n/2 \rfloor$
6: $L_1 \leftarrow \text{DCSMPD}(L[1, \ldots, z])$
7: $L_2 \leftarrow \text{DCSMPD}(L[z+1, \ldots, n])$
8: **for** j in $1..|L_1|$ **do**
9: **for** k in $1..|L_2|$ **do**
10: $L_1[j], i, L_2[k] \leftarrow$ CSPairRefine$(L_1[j], L_2[k])$
11: $I \leftarrow I \cup i$
12: **end for**
13: **end for**
14: **return** $L_1 \cup I \cup L_2$
15: **end if**

Merge: Line 8-line 13



$A = \{2, 4, 6\}, B = \{3, 5, 7\},$
$C = \{1, 2, 3\}, D = \{6, 7, 8\}.$

$A \bigstar C = \{4, 6\}, \boxed{\{2\}}, \{1, 3\}$

$A \bigstar D = \boxed{\{4\}}, \boxed{\{6\}}, \{7, 8\}$

$B \bigstar C = \{5, 7\}, \boxed{\{3\}}, \boxed{\{1\}}$

$B \bigstar D = \boxed{\{5\}}, \boxed{\{7\}}, \boxed{\{8\}}$

$\text{SMPD}(A, B, C) = \{1\}, \{2\},$
$\{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}$

Complexity Analysis

- The main cost comes from GCD computation modulo regular chains.
- By means of evaluation/interpolation techniques, these GCDs can be performed modulo 0-dim regular chains.
- Let $T = [T_1, \ldots, T_n]$ be a squarefree regular chain in $\mathbb{K}[X_1 < \cdots < X_n]$.
- $\mathbb{K}(T) := \mathbb{K}[X_1, \ldots, X_n]/\langle T \rangle$ is a direct product of fields (DPF).
- Let $\deg_i T := \deg(T_i, Xi)$, for $1 \leq i \leq n$, and $\deg_T$ be their product.

Complexity Analysis

- For practical purpose, we rely on classical i.e. quadratic algorithms.
- We adapt the extended Euclidean algorithm for polynomials with coefficients in $\mathbb{K}$ to $\mathbb{K}(T)$.
- Recall an extended GCD of f_1 and f_2 in $\mathbb{K}[x]$ with degrees $d_1 \geq d_2$, can be computed in $O(d_1 d_2)$ operations in $\mathbb{K}$.

Complexity Analysis

- What is different in $\mathbb{K}(T)$ for EEA?

  In a Euclidean division step i, to do

  $$r_{i-1} \; rem \; r_i \; mod \; T,$$

  we need first to get the inverse u_i of $lc(r_i)$ s.t.

  $$u_i \times lc(r_i) = 1 \; mod \; T,$$

  which is called a quasi-inverse.

  Let X_k be the main variable of $lc(r_i)$. We need to compute

  $$GCD(lc(r_i), T_{x_k}) \; mod \; T_{<x_k}$$

Complexity Analysis

- In case of a zero-divisor, T splits into a family of $T^1, \cdots, T^m$. Then computation splits into m branches.
- All the elements then need to project to each branch. To perform this projection efficiently, each family of

$$T^1_{x_j}, \cdots, T^m_{x_j} \text{ for } 1 \leq j \leq m$$

should be pair-wise coprime, which is called a *non-critical* triangular decomposition.

- Thus GCD-free basis computation modulo regular chains is needed. We extend Bach's augment refinement method to $\mathbb{K}(T)$ for this.

Complexity Analysis

- Following the inductive process applied in “On the complexity of D5 principle”, we obtain an arithmetic time for regular chains based on classical algorithms.

- Recall that an arithmetic time $T \mapsto \mathsf{A}_n(\deg_1 T, \ldots, \deg_n T)$ is an asymptotic upper bound for the cost of basic polynomial arithmetic operations in $\mathbb{K}(T)$ (counted in $\mathbb{K}$), i.e. $+, *$, quasi-inverse, projection and computation of non-critical decompositions.

An Arithmetic Time for Regular Chain based on Classical Algorithms

Theorem (An arithmetic time for regular chain)

There exists a constant C such that, writing

$$\mathrm{A}_n(d_1, \ldots, d_n) = C^n (d_1 \times \cdots \times d_n)^2,$$

the function $T \mapsto \mathrm{A}_n(\deg_1 T, \ldots, \deg_n T)$ is an arithmetic time for regular chain T in n variables, for all n.

Basic Complexity Result

- An extended GCD of f_1 and f_2 in $\mathbb{K}(T)[y]$ with degrees $d_1 \geq d_2$ can be computed in $O(d_1 d_2 \mathsf{A}_n(T))$ operations in $\mathbb{K}$.

- For a family of monic squarefree polynomials $F = \{f_1, \ldots, f_m\}$ in $\mathbb{K}(T)[y]$ with degrees $d_1, \ldots, d_m$, we can extend the augment refinement method to compute a GCD-free basis of F modulo T in $O(\sum_{1 \leq i < j \leq m}(d_i d_j)\mathsf{A}_n(T))$ operations in $\mathbb{K}$.

- With $\deg(T) = d$ and $\deg(T') = d'$, RCPairRefine or `Difference`(T, T') costs $O(C^{n-1}dd')$ operations in $\mathbb{K}$.

Main Complexity Result

Assume all regular chains are monic and squarefree in dimension zero.

Theorem (Complexity of MPD)

Let $L = \{U_1, \ldots, U_m\}$ be a set of regular chains and the degree of U_i be d_i for $1 \leq i \leq m$. Then a pairwise disjoint representation of L can be computed in $O(C^{n-1} \sum_{1 \leq i < j \leq m} d_i d_j)$ operations in $\mathbb{K}$.

Remark

Consider the case when $d_i = d_j = d$, then MPD can be computed in $O(C^{n-1}(md)^2)$ operations.

Main Complexity Result

Theorem (Complexity of SMPD)

Given a set $L = \{C_1, \ldots, C_m\}$ of constructible sets, each of which is given by some pairwise disjoint regular chains. Let D_i be the number of points in C_i for $1 \leq i \leq m$. An intersection-free basis of L can be computed in $O(C^{n-1} \sum_{1 \leq i < j \leq m} D_i D_j)$ operations in $\mathbb{K}$.

Remark

A special case is when $D_i = D_j = D$, then the number of operations needed to compute an SMPD is bounded by $O(C^{n-1}(mD)^2)$.

An Experimental Comparison

- OldSMPD:
  - the defining regular systems of each constructible set are made (symmetrically) pairwise disjoint;
  - the set theoretical differences and the intersections are computed separately.
- BachSMPD
- DCSMPD: combining a divide-and-conquer approach with the augment refinement method for the operation SMPD.

Timing(s) of 15 Examples Computed by 3 SMPD Algorithms

Sys	OldSMPD	BachSMPD	DCSMPD
1	12.302	3.494	0.786
2	0.303	0.103	0.062
3	1.123	0.259	0.271
4	2.407	1.184	0.703
5	0.574	0.091	0.159
6	0.548	0.293	0.300
7	0.733	0.444	0.211
8	3.430	0.584	0.633
9	25.413	8.292	9.530
10	1097.291	82.468	122.575
11	11.828	0.930	0.985
12	54.197	1.934	1.778
13	0.530	0.047	0.064
14	27.180	13.705	4.626
15	–	1838.927	592.554

An Experimental Comparison

- Improved OldSMPD: extending the RCPairRefine algorithm to positive dimension and manages to compute the difference and the intersection in one pass

- BachSMPD + MPD and DCSMPD + MPD: cleaning each constructible set after SMPD.

Timing(s) of 15 Examples Computed by 3 SMPD Algorithms

Sys	OldSMPD(improved)	BachSMPD(+MPD)	DCSMPD+MPD
1	3.949	3.766	0.914
2	0.118	0.103	0.062
3	0.271	0.259	0.271
4	1.442	1.449	0.927
5	0.116	0.100	0.173
6	0.257	0.300	0.290
7	0.460	0.444	0.211
8	0.607	0.584	0.633
9	9.291	8.347	9.592
10	95.690	82.795	125.286
11	0.912	0.930	1.784
12	12.330	1.934	2.900
13	0.065	0.047	0.065
14	16.792	16.280	6.323
15	2272.550	1876.061	624.679

Conclusions

- We provide different algorithms for removing the redundancies and distinguished two cases:

MPD: *removal* of redundant components within a constructible set

SMPD: *removal with refinement* among a family of constructible sets.

- We rely on quadratic arithmetic motivated by practical concerns.

- We give a complexity analysis of these algorithms in dimension zero:
  - following the work of Bach, Driscoll and Shallit, we have obtained *essentially quadratic time complexity* based on classical (=quadratic) polynomial arithmetic.
  - the GCD of polynomials modulo a zero-dimensional regular chain can be done in *essentially quadratic time complexity* based on quadratic arithmetic.

References


Eric Bach, James R. Driscoll, and Jeffrey Shallit. Factor refinement, 1990.


F. Boulier, F. Lemaire, and M. Moreno Maza.
Well known theorems on triangular systems and the D5 principle, 2006.


C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan.
Comprehensive Triangular Decomposition, 2007.


J. von zur Gathen and J. Gerhard. *Modern Computer Algebra, 1999.*


X. Dahan, M. Moreno Maza, É. Schost, and Y. Xie.
On the complexity of the D5 principle, 2006.


J. Della Dora, C. Dicrescenzo, and D. Duval.
About a new method for computing in algebraic number fields, 1985.


R. Rioboo and M. Moreno Maza.
Polynomial GCD computations over towers of algebraic extensions, 1995.


X. Li, M. Moreno Maza, and R. Rasheed.
Fast arithmetic and modular techniques for triangular decompositions, 2008.


M. Manubens and A. Montes.
Minimal canonical comprehensive Gröbner system.


J. O'Halloran and M. Schilmoeller.

Gröbner bases for constructible sets,2002