On the Representation of Constructible Sets

Changbo Chen, Liyun Li, Marc Moreno Maza, Wei Pan and Yuzhen Xie
{cchen, liyun, moreno, panwei, yxie}@orcca.on.ca

Abstract

The occurrence of redundant components is a natural phenomenon when computing with constructible sets. We present different algorithms for computing an irredundant representation of a constructible set or a family thereof. We provide a complexity analysis and report on an experimental comparison.

1 Introduction

Constructible sets appear naturally when solving systems of equations, in particular in presence of parameters. Our MAPLE implementation of comprehensive triangular decompositions has led us to dedicate a module of the `RegularChains` library, `ConstructibleSetTools`, to computing with constructible sets. In this paper, we discuss the representation used in our software and the implementation of fundamental operations, such as the set theoretical operations of difference, union and intersection. The problems faced there are representative of the usual dilemma of symbolic computation: choosing between canonical representation and lazy evaluation.

We represent a constructible set C by a list $[[T_1, h_1], \ldots, [T_e, h_e]]$ of so-called regular systems, where a regular system is a pair $[T, h]$ consisting of a regular chain T and a polynomial h regular w.r.t. the saturated ideal of T. Then the points of C are formed by the points that belong to at least one quasi-component $W(T_i)$ without canceling the associated polynomial h_i.

Example 1 *The constructible set C given by the conjunction of the conditions $s-(y+1)x = 0, s-(x+1)y = 0, s-1 \neq 0$ can be represented by two regular systems $R_1 = [T_1, h_1]$ and $R_2 = [T_2, h_2]$, where*

$$\left\{ \begin{array}{lcl} T_1 & = & [(y+1)x - s, y^2 + y - s] \\ h_1 & = & s - 1 \end{array} \right. , \quad \left\{ \begin{array}{lcl} T_2 & = & [x+1, y+1, s] \\ h_2 & = & 1 \end{array} \right.$$

and $x > y > s$; recall that $W(T_1)$ is defined by $y+1)x - s = y^2 + y - s = 0$ and $y + 1 \neq 0$ whereas $W(T_2)$ is defined by $x + 1 = y + 1 = s = 0$. In the representation of constructible sets, two levels of redundancy need to be considered. A first one appears in representing a single constructible set with regular systems. This problem exists when computing the complement of a constructible set, the union, the intersection or the difference of two constructible sets. For instance, one of the central operations is the union of two constructible sets C_1 and C_2. The lazy evaluation point of view suggests to represent $C_1 \cup C_2$ by concatenating the lists of regular systems representing C_1 and C_2. Of course, a `simplify` function is needed, at least in order to remove duplicated regular systems. A canonical representation could be achieved via a decomposition into irreducible components as in [10], but this could be very expensive for our usage of the union of two constructible sets. Alternatively, we remove the redundancy by making the regular systems pairwise disjoint (MPD). In the `ConstructibleSetTools` module of the `RegularChains` library in MAPLE 12, this operation is implemented in the function `MakePairwiseDisjoint`. The fundamental algorithm in support of MPD is the `Difference` of two regular systems $[T, h]$ and $[T', h']$, i.e. $Z([T, h]) \setminus Z([T', h'])$, which will be described in Section 2.

A second level of redundancy can occur in a family of constructible sets. Our point of view is to provide an *intersection-free* representation of these constructible sets at an acceptable cost and to remove the redundancy as well. More precisely, let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be a set of constructible sets, each of which is represented by a series of regular systems. For $\mathcal{C}$, redundancy occurs while some C_i intersects a C_j for $i \neq j$. Like the coprime factorization for integers, $\mathcal{C}$ can be refined to an intersection-free basis $\mathcal{D} = \{D_1, \ldots, D_n\}$, that is, $\mathcal{D}$ is a set of constructible sets such that

(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 finite union of some of the D_j's.

This simplification operation is called *Symmetrically Make Pairwise Disjoint* (SMPD) and it has been implemented as `RefiningPartition` in the `ConstructibleSetTools` module. The input constructible sets of `RefiningPartition` are assumed to be represented by regular systems. For any other forms, one should use triangular decompositions [3] to obtain such a representation. The work in [7] suggests that the techniques from multiple-valued logic minimization could help with simplifying the problems before triangular decomposition.

Relying on the traditional Euclidean algorithm for computing GCDs [6] and the augment refinement method by Bach, Driscoll and Shallit in [1], this paper introduces efficient algorithms for MPD and SMPD by exploiting the triangular structure of the regular system representation. Then, we give a complexity analysis of our algorithms under some realistic assumptions. We also report an experimental comparison among the implementations of three algorithms for SMPD: the one following the approach of Bach et al. (BachSMPD), one using a divide-and-conquer approach (DCSMPD) and the one of [3] (OldSMPD) where we introduced this operation. All the tested examples are well known problems on parametric polynomial systems [3].

2 Background

The starting point of our work is an algorithm for computing the set theoretical difference of the zero sets of two regular systems $Z([T, h])$ and $Z([T', h'])$. We introduced this algorithm in [3] as a building block for simplifying the representations of constructible sets by means of MPD and SMPD, defined in the previous section.

To be brief, we restrict ourselves to the case where $h = h' = 1$ and the initials of T and T' are all equal to 1, too. The complete algorithm has a similar structure but with more branches for case discussion. A sketch of this algorithm is given below and is illustrated by Figure 1.

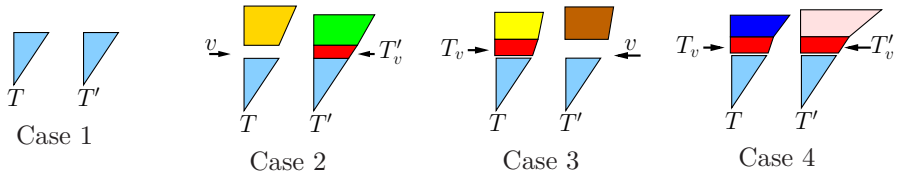

Figure 1: Compute $Z([T, h]) \setminus Z([T', h'])$ with $h = h' = 1$ by exploiting the triangular structure level by level.

Case 1: If T and T' generate the same ideal, which can be tested by pseudo-division, then the difference is empty.

If not, there exists a variable v such that below v the two sets generate the same ideal while at the level of v they disagree. This leads to the following case discussion.

Case 2: Assume that there is a polynomial in T' with main variable v and no such a polynomial in T. We then have two groups of points:

- those from $V(T)$ (the zero set of T) that do not cancel T'_v.
- those from $V(T)$ that cancel T'_v but which are outside of $V(T')$, which leads to a recursive call.

Case 3: Assume that there is a polynomial in T with main variable v and no such a polynomial in T'. Then, it suffices to exclude from $V(T)$ the points of $V(T')$ that cancel T_v, leading to a recursive call.

Case 4: Now we assume that both T_v and T'_v exist. By assumption, they are different modulo $T_{<v}$, which is the regular chain below the level v. Let g be their GCD modulo $T_{<v}$. To be simple, we assume that no splitting is needed and that the initial of g is 1. Three sub-cases arise:

Case 4.1: If g is a constant then the ideals generated by T and T' are relatively prime, hence $V(T)$ and $V(T')$ are disjoint and we just return $[T, 1]$.

Case 4.2: If g is non-constant but its main variable is less than v we have two groups of solution points:

- those from $V(T)$ that do not cancel g,
- those from $V(T)$ that cancel g but are still outside of $V(T')$, which leads to a recursive call.

Case 4.3: Finally, if g has main variable v, we just split T following the D5 principle philosophy [5] and we make two recursive calls.

From the above algorithm, we can see that the main cost comes from GCD computation modulo regular chains. By means of evaluation/interpolation techniques, these GCDs can be performed modulo zero-dimensional regular chains [8]. Thus if all the regular chains in the regular systems representing a constructible set have the same dimension and the same set of algebraic variables, one can reduce all computations to dimension zero; see Proposition 1.12 in [2] for a justification of this point. Therefore, in the present study, we restrict ourselves to regular systems $[T, h]$ such that the saturated ideal of T is zero-dimensional. We shall relax this restriction in future work. Under this zero-dimensional assumption, the ideal $\langle T \rangle$ is equal to the ideal generated by T; moreover h is invertible modulo T and thus can be assumed to be 1, or equivalently, can be ignored. Finally, we shall assume that the base field $\mathbb{K}$ is perfect and that $\langle T \rangle$ is radical. The latter assumption is easily achieved by squarefree factorization, since $\langle T \rangle$ is zero-dimensional.

Based on the above approach, we employ the augment refinement method in [1] in order to implement MPD for a list of regular systems or SMPD for a list of constructible sets. According to this method, given a set C_r and a list of pairwise disjoint sets $[C_1, \ldots, C_{r-1}]$, an intersection-free basis of $C_1, \ldots, C_{r-1}, C_r$ is computed by the following natural principle. First, we consider the pair of C_r and C_1; let $G_{C_r C_1}$ be their intersection. Second, we put $G_{C_r C_1}$ and $C_1 \setminus G_{C_r C_1}$ in the result list. Third, we need to consider the pair of $C_r \setminus G_{C_r C_1}$ and C_2 and continue similarly. In broad terms and in summary, only the "remaining part" of C_r from the first "pair refinement" needs to be considered for the rest of the "original refinement".

3 A complexity analysis

Our objective is to analyze the complexity of algorithms implementing MPD and SMPD operations. We rely on classical, thus quadratic, algorithms for computing GCDs modulo zero-dimensional regular chains [9]. Our motivation is practical: we aim at handling problem sizes to which the asymptotically fast GCDs of [4] are not likely to apply. Even if they would, we do not have yet implementations for these fast GCDs.

Let $T = [T_1, \ldots, T_n]$ be a zero-dimensional regular chain in $\mathbb{K}[X_1 < \cdots < X_n]$. We assume that T generates a radical ideal. The residue class ring $\mathbb{K}(T) := \mathbb{K}[X_1, \ldots, X_n]/\langle T \rangle$ is thus a direct product of fields (DPF). We denote by $\deg_i T$ the degree of T_i in X_i for $1 \leq i \leq n$. The degree of T is defined to be $\prod_{i=1}^{n} \deg_i(T)$.

We first adapt the extended Euclidean algorithm with coefficients in a field [6] to coefficients in a DPF defined by a regular chain. Then we use the augment refinement method of [1] to compute a polynomial GCD-free basis over a DPF. Following the inductive process applied in [4], we achieve the complexity result of Theorem 1. 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 $\mathbb{K}(T)$, counting operations in $\mathbb{K}$; see [4] for details.

Theorem 1 *There exists a constant C such that, writing*

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

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

Therefore, 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 $\mathbb{K}$. Moreover, 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 extend the augment refinement method in [1] to compute a GCD-free basis of F modulo T in $O(\sum_{1 \le i < j \le m}(d_i d_j))\mathsf{A}_n(T)$ operations in $\mathbb{K}$.

To estimate the time complexity of MPD and SMPD in zero-dimensional case where a regular system can be regarded as a regular chain, we start from the base operation RCPairRefine, presented in Algorithm 1. Given two zero-dimensional, monic and squarefree regular chains T and T' in $\mathbb{K}[X_1, \ldots, X_n]$, RCPairRefine produces three constructible sets D, I and D' such that $Z(D) = V(T) \setminus V(T')$, $Z(I) = V(T) \cap V(T')$ and $Z(D') = V(T') \setminus V(T)$. In other words, $\{D, I, D'\}$ is an intersection-free basis of the zero sets defined by T and T'. In the worst case, for each v in $X_1, \ldots, X_n$, a GCD of T_v and T'_v modulo $T_{<v}$ and two divisions are performed. The GCD operation used in Algorithm 1 is specified in [9]. If the degrees of regular chains T and T' are d and d' respectively, then RCPairRefine costs $O(C^{n-1}dd')$ operations in $\mathbb{K}$.

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** $(g, G) \in \mathsf{GCD}(T_v, T'_v, T_{<v})$ **do**
7: **if** $g \in \mathbb{K}$ **or** $\mathrm{mvar}(g) < v$ **then**
8: $T_q \leftarrow G \cup \{T_v\} \cup T_{>v};\ T'_q \leftarrow G \cup \{T'_v\} \cup T'_{>v};\ D \leftarrow D \cup T_q;\ D' \leftarrow D' \cup T'_q$
9: **else**
10: $q \leftarrow \mathrm{pquo}(T_v, g, G);\ q' \leftarrow \mathrm{pquo}(T'_v, g, G);\ E \leftarrow G \cup \{g\} \cup T_{>v};\ E' \leftarrow G \cup \{g\} \cup T'_{>v}$
11: **if** $\mathrm{mvar}(q) = v$ **then**
12: $T_q \leftarrow G \cup \{q\} \cup T_{>v};\ D \leftarrow D \cup T_q$
13: **end if**
14: **if** $\mathrm{mvar}(q') = v$ **then**
15: $T'_q \leftarrow G \cup \{q'\} \cup T'_{>v};\ D' \leftarrow D' \cup T'_q$
16: **end if**
17: $W, J, W' \leftarrow \mathsf{RCPairRefine}(E, E');\ D \leftarrow D \cup W;\ I \leftarrow I \cup J;\ D' \leftarrow D' \cup W'$
18: **end if**
19: **end for**
20: **return** D, I, D'
21: **end if**

Note that Algorithm 1 suffices to compute solely `Difference`(T, T'), i.e. $V(T) \setminus V(T')$ when removing the lines for computing I and D'. Thus, the cost of `Difference`(T, T') is also bounded by $O(C^{n-1}dd')$.

Using RCPairRefine and adapting the augment refinement method in [1] to a list of regular systems or constructible sets, the operation `CSPairRefine` for computing an intersection-free basis of a pair of constructible sets can be deduced naturally. Based on these operations, we build Algorithms 2 and 3 to implement MPD and SMPD respectively. Their complexity results are stated in the theorems below.

Theorem 2 *Let $L = \{U_1, \ldots, U_m\}$ be a set of monic and square free regular chains in dimension zero and the degree of U_i be d_i for $1 \le i \le m$. Then a pairwise disjoint representation of L (that is, regular chains $S_1, \ldots, S_q$ such that $\{V(S_1), \ldots, V(S_q)\}$ forms a partition of the union of $V(U_1), \ldots, V(U_m)$) can be computed in $O(C^{n-1}\sum_{1 \le i < j \le m} d_i d_j)$ operations in $\mathbb{K}$.*

Theorem 3 *Given a set $L = \{C_1, \ldots, C_m\}$ of constructible sets, each of which is given by some monic squarefree and pairwise disjoint regular chains in dimension zero. Let D_i be the number of points in C_i for $1 \le i \le m$. An intersection-free basis of L can be computed in $O(C^{n-1}\sum_{1 \le i < j \le m} D_i D_j)$ operations in $\mathbb{K}$.*

Algorithm 2 MPD

Input: a list L of monic squarefree 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: $L^* \leftarrow$MPD$(L[1, \ldots, n-1])$
7: **for** $l' \in L^*$ **do**
8: $d \leftarrow \mathsf{Difference}(d, l')$
9: **end for**
10: **return** $d \cup L^*$
11: **end if**

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: $L^* \leftarrow \mathsf{BachSMPD}(L[1, \ldots, n-1])$
7: **for** $l' \in L^*$ **do**
8: $d, i, d' \leftarrow \mathsf{CSPairRefine}(d, l')$
9: $I \leftarrow I \cup i;\ D' \leftarrow D' \cup d'$
10: **end for**
11: **return** $d \cup I \cup D'$
12: **end if**

We have also combined a divide-and-conquer approach with the augment refinement method, leading to another algorithm, called DCSMPD, for the operation SMPD. Our analysis shows that its worst case complexity is the same as that BachSMPD; however it performs better for some tested examples.

The main operation in our divide-and-conquer algorithm merges two lists of pairwise disjoint constructible sets $[A_1, \ldots, A_s]$ and $[B_1, \ldots, B_t]$. We first consider A_1 and $[B_1, \ldots, B_t]$ following the principle of the augment refinement method described in Section 2. This will result in three parts: $[G_{A_1B_1}, \ldots, G_{A_1B_t}]$, $[B_1 \backslash G_{A_1B_1}, \ldots, B_t \backslash G_{A_1B_t}]$ and $A_1 \backslash G_{A_1B_1} \backslash \cdots \backslash G_{A_1B_t}$, where $G_{A_1B_1}, \ldots, G_{A_1B_t}$ are the respective intersections of A_1 and B_i for $1 \leq i \leq t$. Next we only need to consider A_2 with respect to $[B_1 \backslash G_{A_1B_1}, \ldots, B_t \backslash G_{A_1B_t}]$ since A_2 is disjoint from $G_{A_1B_i}$ for $1 \leq i \leq t$. The same rule applies to each of $A_3, \ldots, A_s$.

4 An experimental comparison

In this section we provide benchmarks on the implementation of three different algorithms for realizing the SMPD operation, respectively OldSMPD, BachSMPD and DCSMPD.

Given a list $\mathcal{C}$ of constructible sets, the algorithm OldSMPD first collects all their defining regular systems into a list, then computes its intersection-free basis $\mathcal{G}$ which consists of regular systems, and finally one can easily group $\mathcal{G}$ into an intersection-free basis of $\mathcal{C}$. In this manner the defining regular systems of each constructible set are made (symmetrically) pairwise disjoint, though sometimes this is unnecessary. As reported in [3], OldSMPD is expensive and sometimes can be a bottleneck. After replacing OldSMPD in the comprehensive triangular decomposition (CTD) algorithm respectively by BachSMPD and DCSMPD, we rerun the CTD algorithm for twenty examples selected from [3] (all examples are in positive dimension).

The second column named OldSMPD in Table 1 is the timing from [3] where SMPD was first implemented. The third column OldSMPD (improved) extends the RCPairRefine algorithm to positive dimension and manages to compute the difference and the intersection in one pass, whereas in OldSMPD the set theoretical differences and the intersections are computed separately. The fourth column and the sixth column present the timings of computing an SMPD with BachSMPD and DCSMPD respectively. The fifth column and the seventh column show the timings for cleaning each constructible set with an MPD operation on each of the constructible set in the output. In this way, we remove the redundancy both among a series of constructible sets and in their defining regular systems.

Table 1 shows that BachSMPD and DCSMPD are more efficient than OldSMPD. We know from the previous section that Algorithms BachSMPD and DCSMPD have the same complexity in the worst case. However, experimentation shows that DCSMPD performs more than 3 times faster than BachSMPD for some examples, which need to be investigated in the future.

Sys	OldSMPD	OldSMPD (improved)	BachSMPD	BachSMPD (+MPD)	DCSMPD	DCSMPD (+MPD)
9	3.817	0.871	0.818	0.877	1.112	1.435
10	1.138	0.154	0.223	0.223	0.281	0.344
11	12.302	3.949	3.494	3.766	0.786	0.914
12	10.114	0.551	0.383	0.383	0.318	0.318
13	1.268	0.348	0.318	0.318	0.362	0.363
14	0.303	0.118	0.103	0.103	0.062	0.062
15	1.123	0.271	0.259	0.259	0.271	0.271
16	2.407	1.442	1.184	1.449	0.703	0.927
17	0.574	0.116	0.091	0.100	0.159	0.173
18	0.548	0.257	0.293	0.300	0.283	0.290
19	0.733	0.460	0.444	0.444	0.211	0.211
20	0.020	0.013	0.013	0.013	0.013	0.013
21	3.430	0.607	0.584	0.584	0.633	0.633
22	25.413	9.291	8.292	8.347	9.530	9.592
23	1097.291	95.690	82.468	82.795	122.575	125.286
24	11.828	0.912	0.930	0.930	0.985	1.784
25	54.197	12.330	1.934	1.934	1.778	2.900
26	0.530	0.065	0.047	0.047	0.064	0.065
27	27.180	16.792	13.705	16.280	4.626	6.323
28	–	2272.550	1838.927	1876.061	592.554	624.679

Table 1 Timing(s) of 20 examples computed by 3 algorithms

References

[1] E. Bach, J. R. Driscoll, and J. Shallit. Factor refinement. In *SODA*, pages 201–211, 1990.

[2] F. Boulier, F. Lemaire, and M. Moreno Maza. Well known theorems on triangular systems and the D5 principle. In *Proc. of* Transgressive Computing, Spain, 2006.

[3] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. *Comprehensive Triangular Decomposition*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101. Springer Verlag, 2007.

[4] X. Dahan, M. Moreno Maza, É. Schost, and Y. Xie. On the complexity of the D5 principle. In *Proc. of* Transgressive Computing, Spain, 2006.

[5] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Proc. EUROCAL 85 Vol. 2*, volume 204 of *LNCS*, pages 289–290. Springer-Verlag, 1985.

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

[7] H. Hong. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In *Proc. ISSAC'92*, pages 177–188.

[8] X. Li, M. Moreno Maza, and R. Rasheed. Fast arithmetic and modular techniques for polynomial gcds modulo regular chains, 2008. Preprint.

[9] R. Rioboo and M. Moreno Maza. Polynomial GCD computations over towers of algebraic extensions. In *Proc. AAECC-11*, pages 365–382, 1995.

[10] M. Manubens and A. Montes. Minimal canonical comprehensive Gröbner system, 01-12-06. arXiv:math.AC/0611948.