Solving semi-algebraic systems with the `RegularChains` library in MAPLE

Changbo Chen, James H. Davenport, François Lemaire,
Marc Moreno Maza, Nalina Phisanbut, Bican Xia,
Rong Xiao and Yuzhen Xie

It is well known that multivariate nonlinear polynomial systems are genuinely harder to solve than linear systems. This is intrinsic, as the solutions display features that do not manifest themselves in the linear case. For instance, the solution set of a nonlinear polynomial system may consist of components of different dimensions. Moreover, even if all components have the same dimension, say dimension zero, they may not be glued into a single component without loosing finer properties such as *equi-projectability*, which is important in the design of algorithms for solving polynomial systems.

To illustrate this latter property without defining it formally, consider the following system of three equations, two unknowns and with 750,000 complex solutions:

$$\{x^{1000} - 1, (x^{500} - 1)(y^{500} - 1), y^{1000} - 1\} \tag{1}$$

The algorithm presented in [9] and implemented by the `Triangularize` command of the `RegularChains` library produces the following *triangular decomposition*

$$\{(x, y) : x^{500} - 1 = 0, y^{1000} - 1 = 0\} \cup \{(x, y) : x^{500} + 1 = 0, y^{500} - 1 = 0\} \tag{2}$$

which consists of two equi-projectable components, each of them represented by a *regular chain*. The triangular set

$$\{x^{1000} - 1, (x^{500} - 1)(y^{500} - 1) + (x^{500} + 1)(y^{1000} - 1)\}$$

has the same solution set as our input system. However this is not a well-behaved algebraic object since the leading coefficient in y of the second polynomial is a zero-divisor modulo the first polynomial. And, for this reason, this triangular set is not a regular chain. This algebraic observation has a geometrical interpretation clearly visible in (2): for each x-value, the first component admits 1000 y-values while for each x-value, the second component has only 500 y-values.

Since the pioneer work of Wen Tsün Wu [24], the notion of a regular chain and algorithms for computing triangular decomposition of algebraic systems have

been well studied, leading to various software packages [18, 23, 22]. This type of decomposition reveals geometrical information of practical interest while providing compact (in the sense of space-efficiency) representation of solution sets and supporting various meanings (more or less explicit and complete) for "solving". Similarly to methods based on Gröbner basis computations, triangular decomposition algorithms solve the input system over the algebraic closure of its field of coefficients.

What of $\mathbb{R}$? Here it is natural to allow both (strict) inequalities, i.e. $f_j(x_1, \ldots, x_n) > 0$, and inequations $f_k(x_1, \ldots, x_n) \neq 0$, as well as equations. Combinations of these give rise to *semi-algebraic* systems. In the paper [5], the authors adapt the notion of a regular chain to this context, leading to that of a *regular semi-algebraic system*. They also propose algorithms for computing triangular decompositions of semi-algebraic systems. In a second article [6], the authors show how semi-algebraic sets (that is the solution sets of semi-algebraic systems) represented by triangular decompositions can be manipulated and, for instance, how set-theoretic operations on those sets can be performed with this representation.

The first objective of the present work is to explain how the notions and algorithms introduced in [5, 6] have been implemented in the `RegularChains` library. As a byproduct, the notion of a regular chain supports most end-user needs for solving algebraic and semi-algebraic systems. By "solving", we mean here describing the locus of the zeros of the input system. The question of preserving the multiplicity structure of the zero set is work in progress and is not yet supported in our library. The library's current design and implementation strategies are discussed in Section 2.

Among the specifications for "solving" semi-algebraic systems, *cylindrical algebraic decomposition* (CAD) and *point sampling*[1] play a fundamental role in practice. Both specifications can be obtained by algorithms based on regular chains. This strengthens our observation that the notion of a regular chain is at the core of many practical tools for polynomial system solving. In Section 3, we report on an experimental comparison of those algorithms against their counterparts based on Collins' CAD method.

Section 4 illustrates the `RegularChains` library semi-algebraic tools with two applications. One deals with branch cut analysis of elementary functions, the other shows how the output of a "real solver" can be verified.

1. Representations based on regular chains

The notion of a regular chain, introduced independently in [17] and [26], is closely related to that of a triangular decomposition of a polynomial system. Broadly speaking, a *triangular decomposition*[2] of a polynomial system S, such as (1), is a

[1] By this term, we mean computing (at least) one point per connected component of the zero set of the input semi-algebraic system.

[2] http://en.wikipedia.org/wiki/Triangular_decomposition

set of simpler (in a precise sense) polynomial systems $S_1, \ldots, S_e$ (as in (2)) such that

$$p \text{ is a solution of } S \Leftrightarrow \exists i : p \text{ is a solution of } S_i. \tag{3}$$

When the purpose is to describe the solution set of S in the algebraic closure of its coefficient field, those simpler systems are regular chains. If the coefficients of S are real numbers, then the real solutions of S can be obtained by a triangular decomposition into *regular semi-algebraic systems*, a notion introduced in [5]. In both cases, each of these simpler systems has a triangular shape and remarkable properties, which justifies the terminology.

A regular semi-algebraic system is a triple $[T, Q, P]$ where T is a regular chain, Q is a quantifier-free formula involving only the free variables of T and P is a set of polynomial inequalities; moreover $[T, Q, P]$ must satisfy the following properties.

(i) Q defines a non-empty open set in the space of the free variables of T,
(ii) $[T, P]$ specializes well at any point[3] defined by Q,
(iii) At any point α defined by Q, the specialized system $[T_\alpha, P_\alpha]$ admits at least one real solution β, in the sense that every polynomial in T_α is zero at β, and every polynomial in P_α is positive at β.

A consequence of (3) is that, if all the T_i have the same free variables, i.e. parameters, then $Q_1 \vee \cdots \vee Q_e$ defines the set of parameter values for which the input system possesses solutions. However, it is not necessary for all the T_i to have the same free variables and the example below given by (4) illustrates this fact.

An important property of any regular semi-algebraic system $[T, Q, P]$ is the fact that it is a parametrization of its zero set. Therefore, a triangular decomposition of a semi-algebraic system S decomposes the zero set of S into components, with each of them given by a parametric representation.

The need to split semi-algebraic (or indeed algebraic) systems into components may be unfamiliar to those who are focusing on linear equations and have never seen non-equi-projectable systems such as

$$(x-1)(x-2) = (x-1)y + (x-2)(y^2 - t) = 0, \tag{4}$$

which admits one solution point for $t(x-2) = 0$ and two for $x = 1,\ 0 < t$, as computed by the `RealTriangularize` command of the `RegularChains` library.

```
> RealTriangularize([(x-1)*(x-2),(x-1)*y + (x-2)*(y^2-t)], PolynomialRing([y,x,t]));
      {  2                  {   y = 0
      { y  - t = 0          {                    {   y = 0
      {                   , { x - 1 = 0        , {
      { x - 1 = 0           {                    { x - 2 = 0
      {                     {   t = 0
      {   0 < t
```

Triangular decompositions into regular semi-algebraic systems are an interesting representation of semi-algebraic sets for the following reasons. First, triangular decompositions into regular chains are a space-efficient encoding of algebraic

[3]This means that at any point u defined by Q, the specialized set $T(u)$ is a squarefree regular chain with the same rank as T and each specialized polynomial $P_i(u)$ is invertible modulo $\langle T(u) \rangle$.

sets. This fact is formally established in [12] and experimentally verified with the `RegularChains` library in [9]. Secondly, triangular decompositions into regular chains (or regular semi-algebraic systems) reveal important geometrical properties (dimensions of the irreducible components, fibration structure, etc.) of the input algebraic sets (or semi-algebraic sets). These can be used to design efficient algorithms for the operations manipulating those sets. For instance, performing set theoretic operations (in particular set theoretic difference) can be done very efficiently on both constructible sets and semi-algebraic sets as reported in [8] and [6], respectively. Last but not least, triangular decompositions and related techniques, such as dynamic evaluation, are well suited for supporting weaker solving specifications. Two examples of that are triangular decomposition of algebraic systems in the sense of Kalkbrener [17] and lazy triangular decomposition of semi-algebraic systems [5]. These types of decompositions provide a description of the "generic solutions" plus a continuation mechanism for obtaining the other solutions, if necessary.

2. The `RegularChains` library

The `RegularChains` library in MAPLE provides a collection of tools for dealing with systems of polynomial equations, inequations and inequalities. These tools include isolating and counting the real solutions of zero-dimensional systems, describing the real solutions of positive dimensional systems, classifying the number of real roots of parametric systems, finding sample points (thus determining emptiness) of semi-algebraic sets, performing set theoretical operations on semi-algebraic sets as well as computing cylindrical algebraic decompositions. The theory and algorithms underlying these tools are described in [25, 27, 10, 1, 5, 6]. Most commands implementing these tools are part of the `SemiAlgebraicSetTools` module while the others can be found in the `ParametricSystemTools` module or at the top level of the `RegularChains` library itself. All of these commands are already present in MAPLE 15 (that is, in the current version of MAPLE) except those for set theoretical operations which will appear in a future release.

One important design feature of the `RegularChains` library is the use of types for a few key algebraic structures such as *regular chains*, *constructible sets*, *semi-algebraic sets*, etc. This feature, unusual for a MAPLE package, forces the user to provide command input in an unambiguous manner and eases the manipulation of complicated output values. Let us illustrate this design feature with two examples.

Based on the algorithms of [5], the `RealTriangularize` command decomposes an input semi-algebraic system into finitely many *regular semi-algebraic systems* (See Figure 1 for an example.). An object of type `regular_semi_algebraic_system` consists of a regular chain, a quantifier-free formula and positive inequalities. The `RegularChains` library provides types for the former two while inequalities are a MAPLE primitive type.

The fact that RealTriangularize decomposes any semi-algebraic system into finitely many regular semi-algebraic systems leads to a convenient representation of semi-algebraic sets. As mentioned in Section 1, regular semi-algebraic systems enjoy remarkable properties, which enable efficient implementation of set theoretical operations on semi-algebraic sets, like Difference and Intersection.

```
> with(RegularChains):
> R := PolynomialRing([z, y, x]);
```

$$R := polynomial\_ring \tag{1}$$

```
> Limao := x^2-y^3*z^3;
  Zylinder := y^2+z^2-1;
```

$$Limao := x^2 - y^3\,z^3$$

$$Zylinder := y^2 + z^2 - 1 \tag{2}$$

```
> RealTriangularize([Zylinder, Limao], R);
```

$$[regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system] \tag{3}$$

```
> Display(%, R);
```

$$\left[\left[\begin{array}{c} (-y^3+y^5)\,z + x^2 = 0 \\ y^{12} - 3\,y^{10} + 3\,y^8 - y^6 + x^4 = 0 \\ 8\,x^2 < 1 \text{ and } x \neq 0 \end{array}\right], \left[\begin{array}{c} 2\,y\,z - 1 = 0 \\ 2\,y^2 - 1 = 0 \\ 8\,x^2 - 1 = 0 \end{array}\right], \left[\begin{array}{c} z = 0 \\ y - 1 = 0 \\ x = 0 \end{array}\right], \left[\begin{array}{c} z + 1 = 0 \\ y = 0 \\ x = 0 \end{array}\right], \left[\begin{array}{c} z - 1 = 0 \\ y = 0 \\ x = 0 \end{array}\right], \left[\begin{array}{c} z = 0 \\ y + 1 = 0 \\ x = 0 \end{array}\right]\right] \tag{4}$$

FIGURE 1. Output of RealTriangularize

Based on the algorithms of [10], the CylindricalAlgebraicDecompose command takes a list of semi-algebraic systems as input and returns the CAD cells satisfying at least one of the input semi-algebraic systems. (See Figure 2 for an example.). A cell is encoded by the type cad_cell, which consists of informations related to this cell, such as its index, a sample point and its semi-algebraic representation. A semi-algebraic representation of a CAD cell of the n-dimensional Euclidean space with coordinates $x_1 < \cdots < x_n$ is defined recursively as follows.

- For $n = 1$ it is either a point or an interval; in the first case x_1 is set to an algebraic expression while in the second case x_1 is strictly bounded between two algebraic expressions; in both cases each of these algebraic expressions is a MAPLE RootOf expression, where the defining polynomial is univariate over the rational numbers.
- For $n > 1$ it is a CAD cell of the $(n-1)$-dimensional space together with a constraint on x_n of type section or sector. In the first case x_n is set to an algebraic expression and in the second x_n is strictly bounded between two algebraic expressions; in both cases, each algebraic expression is a RootOf expression where the defining polynomial is univariate with coefficients that are polynomials in the previous coordinates.

The sample point of a CAD cell is stored during the computation as a field of the type cad_cell. If one wants to retrieve the sample point of an object of type cad_cell, the command SamplePoints can be applied to such an object and

output the sample point without any computational cost. Thus, the use of type here provides a convenient way for manipulating the computed objects.

The `CylindricalAlgebraicDecompose` command admits various output formats. Among them, the `rootof` format is meant to support end-users' needs for solving semi-algebraic systems. In fact, this format is very similar to that of MAPLE's `solve` command for solving polynomial systems.

```
> R := PolynomialRing([x, c]);
```

$$R := \mathit{polynomial\_ring} \tag{1}$$

```
> cad := CylindricalAlgebraicDecompose([[x^2+x-c>0]], R, output=cadcell);
```

$$cad := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell] \tag{2}$$

```
> Display(cad,R);
```

$$\left[\left\{\begin{array}{l} x = x \\ c < -\frac{1}{4} \end{array}\right., \left\{\begin{array}{l} x < -\frac{1}{2} \\ c = -\frac{1}{4} \end{array}\right., \left\{\begin{array}{l} -\frac{1}{2} < x \\ c = -\frac{1}{4} \end{array}\right., \left\{\begin{array}{l} x < -\frac{1}{2} - \frac{1}{2}\sqrt{4\,c+1} \\ -\frac{1}{4} < c \end{array}\right., \left\{\begin{array}{l} -\frac{1}{2} + \frac{1}{2}\sqrt{4\,c+1} < x \\ -\frac{1}{4} < c \end{array}\right.\right] \tag{3}$$

```
> sp := SamplePoints(cad[4],R); Display(sp,R);
```

$$sp := box$$

$$\left\{\begin{array}{l} x = -3 \\ c = 1 \end{array}\right. \tag{4}$$

```
> cad := CylindricalAlgebraicDecompose([[x^2+x-c>0]], R, output=rootof);
```

$$cad := \left[\left[c < -\frac{1}{4}, x = x\right], \left[c = -\frac{1}{4}, x < -\frac{1}{2}\right], \left[c = -\frac{1}{4}, -\frac{1}{2} < x\right], \left[-\frac{1}{4} < c, x < -\frac{1}{2} - \frac{1}{2}\sqrt{4\,c+1}\right], \left[-\frac{1}{4} < c, -\frac{1}{2} + \frac{1}{2}\sqrt{4\,c+1} < x\right]\right] \tag{5}$$

FIGURE 2. Output of `CylindricalAlgebraicDecompose`

Another design feature is the use of MAPLE `piecewise` structure [7] for formatting the output of commands producing a set of "components" (for instance regular semi-algebraic systems) (See Figure 3 for an example.). This has at least two advantages. First, this highlights the relations between components. Secondly, this supports lazy evaluation in the form of unevaluated recursive calls, see [5] for details.

3. Experimentation

As mentioned in the introduction, among the specifications for "solving" semi-algebraic systems *cylindrical algebraic decomposition* (CAD) and *point sampling*

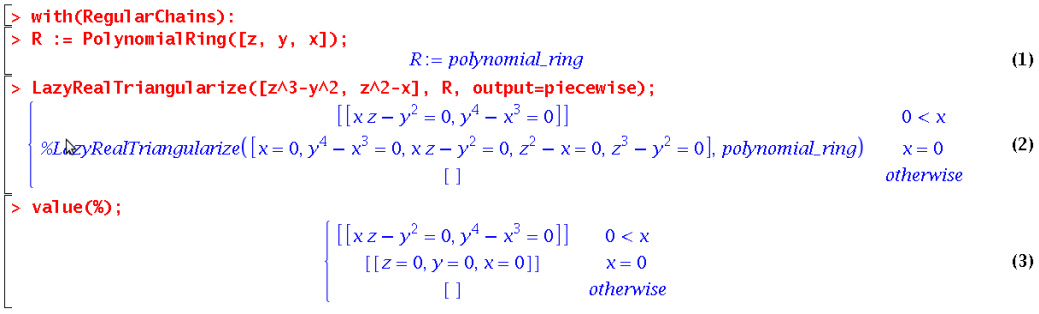

FIGURE 3. Output of `LazyRealTriangularize`

play a fundamental role in practice. Both specifications can be obtained by algorithms based on regular chains. See the papers [10] and [5] for details. In this section, we report on an experimental comparison of those algorithms against their counterparts based on Collins' CAD method as implemented in QEPCAD-B [16, 3] and in Mathematica's SemialgebraicComponentInstances command [21].

3.1. Cylindrical algebraic decomposition

Our CAD algorithm ([10]) is completely different from that of Collins and the numerous improvements to it: we do not perform repeated projections. The fact that we have a different algorithm does not, of course, break through the theoretical complexity barriers [2, 13]. Our practical observations to date (in the context of simplification, see [19]) seem to show the following.

- For problems in two variables, our implementation behaves fairly similarly to QEPCAD [16, 3], which is one of the best descendants of [11].
- The two tend to agree, with only slight variations, on the best and worst variable orderings for constructing a CAD.
- For problems in more variables, with the best variable ordering, the two are again fairly similar, with QEPCAD probably having a slight edge (comparisons are difficult as QEPCAD only constructs a partial CAD, and ours constructs a complete one).
- For bad variable orderings, QEPCAD can produce dramatically more cells, e.g. by a factor of over 50 (see Table 1), than Maple.
- Although some heuristics are known [4, 15], the choice of variable ordering is still an important unsolved issue. While QEPCAD and our implementation roughly agree on what are good orderings, they don't agree precisely, and more research is needed.

TABLE 1. Extract from [19, Table 8-11].

Rank according to QEPCAD cell count			
Variable order	**QEPCAD cells**	**Maple CAD's cells**	**ratio**
$v > y > u > x$ (G)	785	673	1.16
$y > v > x > u$ (G)	785	673	1.16
$y > v > u > x$ (G)	901	557	1.62
$v > y > x > u$ (G)	901	557	1.62
$v > u > y > x$	2049	989	2.07
$y > u > v > x$	2049	1781	1.15
$v > u > x > y$	2049	989	2.07
$v > x > u > y$	2049	1869	1.10
$y > x > v > u$	2049	989	2.07
$v > x > y > u$	2049	1781	1.15
$y > x > u > v$	2049	989	2.07
$y > u > x > v$	2049	1869	1.10
$u > x > v > y$	5985	557	10.74
$x > u > y > v$	5985	557	10.74
$x > u > v > y$	6597	673	9.80
$u > x > y > v$	6597	673	9.80
$u > v > y > x$	9101	989	9.20
$x > y > v > u$	9101	989	9.20
$u > y > v > x$	28821	1781	16.18
$x > v > y > u$	28821	1781	16.18
$u > v > x > y$	37957	989	36.38
$x > y > u > v$	37957	989	36.38
$x > v > u > y$	92829	1781	52.12
$u > y > x > v$	92829	1781	52.12

(G) = recommended by [15]'s Greedy Algorithm

3.2. Point sampling

To evaluate our SamplePoints command in the `RegularChains` library, we run it on some well-known semi-algebraic systems. We also use the SemialgebraicComponentInstances command in Mathematica 8.0.1.0 to compute sample points of the same systems. Both commands produce at least one point per connected component of the input semi-algebraic set.

In Table 2 we report the number of sample points computed by the two commands and the time used for each of these computations. The benchmark runs are carried out on an Intel Core i7 CPU 870 2.93GHz with 8 GB memory and 8 MB cache. The time limit of each benchmark run is set to 10 minutes and the memory limit is set to 6 GB. For each system, the "SysInfo" column lists the

TABLE 2. Maple SamplePoints vs Mathematica SemialgebraicComponentInstances

system	SysInfo	Maple SP		Mathematica SP	
	[ht, dim, deg]	time (s)	#	time (s)	#
BM05-1	[2, 2, 6]	0.020	2	0.010	6
BM05-2	[1, 2, 6]	0.120	10	0.040	27
DescartesFolium	[2, 1, 4]	0.039	2	0.000	2
EdgeSquare	[1, 3, 3]	9.840	80	0.150	153
Ellipse	[1, 5, 3]	1.810	60	0.060	96
EnneperSurface	[2, 2, 9]	0.010	1	0.010	1
IBVP	[1, 4, 2]	4.770	30	-	-
Jirstrand22	[1, 1, 5]	0.160	9	0.020	9
Jirstrand23	[9, 2, 72]	-	-	-	-
Jirstrand24	[1, 1, 9]	0.090	4	0.020	4
Jirstrand41	[3, 1, 3]	0.080	3	0.010	3
Jirstrand42	[1, 3, 8]	0.629	9	0.040	15
Lafferriere35	[3, 2, 7]	0.069	4	0.010	4
Lafferriere37	[2, 0, 15]	0.099	3	0.210	3
MPV89	[2, 3, 8]	0.360	3	0.060	7
p3p-isosceles	[1, 4, 18]	3.540	44	1.820	1876
p3p	[1, 5, 18]	5.019	128	6.590	8452
putnam	[7, 2, 4]	0.340	16	0.010	16
SEIT	[1, 8, 1]	6.050	1	0.000	2
Solotareff-4a	[2, 1, 21]	0.970	1	-	-
Solotareff-4b	[2, 1, 21]	0.970	1	-	-
Xia	[2, 3, 16]	0.419	1	11.430	97
8-3-config-Li	[1, 7, 1]	311.39	81	-	-
Cheaters-homotopy-easy	[1, 4, 13]	0.020	1	0.000	1
Cinquin_Demongeot-3-3	[1, 1, 55]	0.520	24	88.330	223
Cinquin_Demongeot-3-4	[1, 1, 133]	0.350	11	20.300	39
collins-jsc02	[2, 1, 21]	0.590	19	-	-
dgp29	[1, 2, 720]	0.010	1	0.010	1
dgp6	[1, 3, 33]	4.410	3	-	-
DonatiTraverso-rev	[1, 2, 3]	0.159	6	-	-
Hairer-2-BGK	[5, 2, 32]	2.689	36	-	-
hereman-8	[9, 5, 6]	1.140	21	-	-
Leykin-1	[4, 4, 15]	1.520	20	-	-
Lichtblau	[13, 1, 11]	0.010	1	0.010	1
L	[1, 9, 1]	27.850	198	-	-

maximum bit-size [4] of a coefficient among the input polynomials, followed by the

[4]more precisely, $\max\{\lfloor \log_2(\mid c \mid) \rfloor + 1 \mid \mid c \text{ is a non-zero coefficient in the system}\}$ taking into account the fact that coefficients are all integer numbers.

dimension and the degree of the variety defined by the input equations. The "-"s in Table 2 describes the cases where the function can not produce a result within the time and memory limits for the system under test.

One should stress the fact that sample points are not encoded in the same way by the SamplePoints and SemialgebraicComponentInstances commands:

- in the former case, a sample point $(s_1, \ldots, s_n)$ is given by a zero-dimensional regular chain $T_1(x_1), \ldots, T_n(x_1, \ldots, x_n)$ and closed intervals with rational end-points $I_1, \ldots, I_n$ such that $T_j(s_1, \ldots, s_j) = 0$ and $s_j \in I_j$ both hold for all $j = 1 \cdots n$.
- in the latter case, a sample point $(s_1, \ldots, s_n)$ is given by univariate polynomials $p_1, \ldots, p_n \in \mathbb{Q}[x]$ and root indices $i_1, \ldots, i_n$ such that s_j is the i_j-th real root of the polynomial p_j, for all $j = 1 \cdots n$.

The results show that our SamplePoints command can generate results within the time and memory limit for all the 35 systems in our benchmark test except for one, namely Jirstrand23. There are 12 problems for which Mathematica's SemialgebraicComponentInstances command can not conclude, including Jirstrand23. Among the 23 problems that the SemialgebraicComponentInstances command can solve within the time and memory limits, there are 12 for which the numbers of sample points are more than those computed by our SamplePoints command, by a ratio of 2 to 66 times higher. In terms of timing, for many of the cases that the SemialgebraicComponentInstances function can solve, Mathematica's timings are better than the Maple's ones. However, there are a few systems for which the timings by the SemialgebraicComponentInstances function are dramatically higher. We have noticed that the variable ordering plays an important role in this computation. In the SamplePoints command code, we apply some heuristics for determining a variable ordering. The variable ordering used in the SemialgebraicComponentInstances function in Mathematica is not known to us.

4. Applications

In this section, we present two applications of the tools of the `RegularChains` library for manipulating semi-algebraic systems. Both are of high practical importance for computer algebra systems. Indeed, the first one aims at testing the validity of identity involving elementary functions over the complex plane such as $\sqrt{1-z}\sqrt{1+z} = \sqrt{1-z^2}$. The second one targets the verification of polynomial system solvers computing symbolic description of the real solutions.

4.1. Branch cut analysis

In analysis, a major challenge is the manipulation of "multivalued functions". Regarding them as single-valued functions requires the imposition of branch cuts, which are normally semi-algebraic sets in $\mathbb{C}^n = \mathbb{R}^{2n}$ across which the functions are not continuous. For example, the branch cut of $\log z = \log(x + iy)$ is normally taken to be $y = 0 \wedge x < 0$: a linked pair of an equation and an inequality. In [14], the author shows how the connectivity of the complement of the branch

cuts becomes the question of interest. Since cell adjacency in a cylindrical algebraic decomposition (CAD) is explicit, one way (the only practical one known to us) to explore these connectivity questions is to compute a CAD of $\mathbb{R}^{2n}$ induced by the branch cuts, and construct connected components from this. The *CAD* algorithm of [10] starts with a triangular decomposition of the set of polynomials occurring in the branch cuts, irrespective of how they are linked, whereas the QEPCAD approach [3] takes advantage of knowing how the equalities and inequalities are connected. Nevertheless, [20] shows that the approach of [10] often produces no more cells than QEPCAD, and indeed it is possible to "precondition" the inequality by the equality in some cases, to improve the performance of both. Figure 4 illustrates how to conduct branch cut analysis by our CAD, namely the `CylindricalAlgebraicDecompose` command in `RegularChains` library.

```
> R := PolynomialRing([y, x]);
> f1 := sqrt(z^2-1)-sqrt(z+1)*sqrt(z-1); f2 := sqrt(1-z^2)-sqrt(1+z)*sqrt(1-z);
```

$$f1 := \sqrt{z^2-1} - \sqrt{z+1}\,\sqrt{z-1}$$
$$f2 := \sqrt{1-z^2} - \sqrt{z+1}\,\sqrt{1-z} \qquad (1)$$

```
> bc1 := RegularChains:-branchcuts([f1]); bc2 := RegularChains:-branchcuts([f2]);
```

$$bc1 := [[y = 0, x < -1], [y = 0, x < 1], [2xy = 0, x^2 - y^2 < 1]]$$
$$bc2 := [[y = 0, x < -1], [-y = 0, -x < -1], [-2xy = 0, -x^2 + y^2 < -1]] \qquad (2)$$

```
> sp1 := CylindricalAlgebraicDecompose(bc1, R, output=cadcell); sp1 := map(BoxValues, map
  (SamplePoints, sp1, R), R);
  sp2 := CylindricalAlgebraicDecompose(bc2, R, output=cadcell); sp2 := map(BoxValues, map
  (SamplePoints, sp2, R), R);
```

$$sp1 := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell]$$
$$sp1 := \left[[y = 0, x = -2], [y = 0, x = -1], \left[y = 0, x = -\frac{1}{2}\right], [y = -1, x = 0], [y = 0, x = 0], [y = 1, x = 0], \left[y = 0, x = \frac{1}{2}\right]\right]$$
$$sp2 := [cad\_cell, cad\_cell]$$
$$sp2 := [[y = 0, x = -2], [y = 0, x = 2]] \qquad (3)$$

```
> map(p->simplify(subs(z=rhs(p[2])+I*rhs(p[1]), f1)), sp1);
  map(p->simplify(subs(z=rhs(p[2])+I*rhs(p[1]), f2)), sp2);
```

$$[2\sqrt{3}, 0, 0, 2I\sqrt{2}, 0, 0, 0]$$
$$[0, 0] \qquad (4)$$

Note that $x = -2, y = 0$ (i.e. $z = -2$) is explicitly found as a counterexample to $\sqrt{z^2-1} = \sqrt{z+1}\sqrt{z-1}$.

FIGURE 4. Branch cut analysis by `CylindricalAlgebraicDecompose`

4.2. Verification of real solvers

On a given input polynomial system, two solving tools may produce correct results that look fairly different. Proving that these two results are equivalent can be a very complex task. Here's an example. Given a triangle with edge lengths a, b, c (denoting the respective edges a, b, c too) the following two conditions C_1, C_2 are both characterizing the fact that the external bisector of the angle of a, c intersects with b on the other side of a than the triangle: $C_1 = a > 0 \wedge b > 0 \wedge c > 0 \wedge a < b+c \wedge b < a+c \wedge c < a+b \wedge \left(b^2 + a^2 - c^2 \leq 0 \vee c(b^2 + a^2 - c^2)^2 < ab^2(2ac - (c^2 + a^2 - b^2))\right)$,

$C_2 = a > 0 \wedge b > 0 \wedge c > 0 \wedge a < b+c \wedge b < a+c \wedge c < a+b \wedge c-a > 0$. We verify the equivalence of C_1 and C_2 by computing the set-theoretical differences $C_1 \setminus C_2$ and $C_2 \setminus C_1$. The command `Difference` of the `SemiAlgebraicSetTools` module of the `RegularChains` library can be used for this purpose. Figure 5 shows how the computations are conducted.

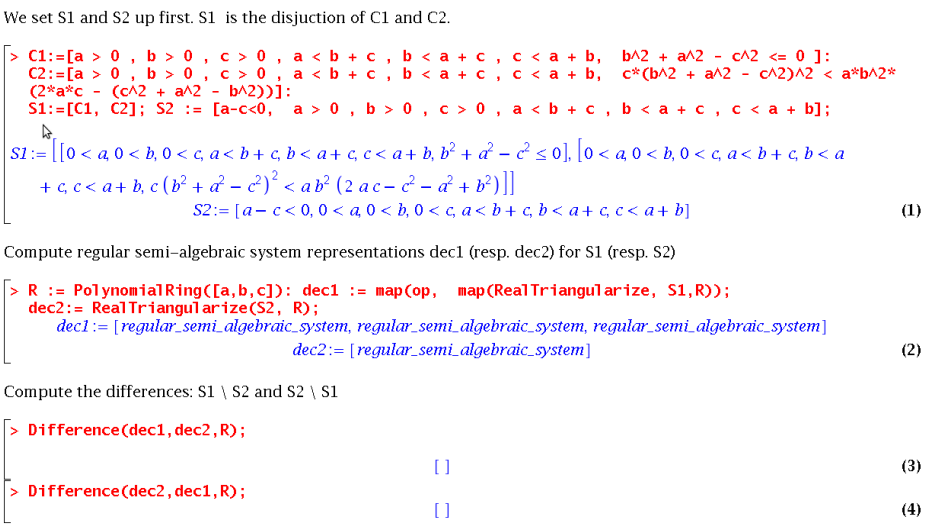

FIGURE 5. Testing the equivalence of two formulas by `Difference`

5. Conclusion

In this paper, we discussed how the notion of a regular chain is adapted and used to represent the solution sets of semi-algebraic systems. We then presented the new functionalities of the `RegularChains` library for computing with the real solutions of polynomial systems based on this representation. We compared our software tools with two related software packages by experimentation. The results of which illustrate the effectiveness of our tools. We have also successfully applied these new functionalities of the `RegularChains` library to branch cut analysis and the verification of "real solvers".

References

[1] F. Boulier, C. Chen, F. Lemaire, and M. Moreno Maza. Real root isolation of regular chains. In *Proc. of ASCM'09*, 2009.

[2] C. W. Brown and J. H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In *Proc. ISSAC'07*, pages 54–60, 2007.

[3] C.W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.*, 37(4):97–108, 2003.

[4] C.W. Brown. Tutorial handout: ISSAC 2004. `http://www.cs.usna.edu/~wcbrown/research/ISSAC04/handout.pdf`, 2004.

[5] C. Chen, J. H. Davenport, J. May, M. Moreno Maza, B. Xia, and R. Xiao. Triangular decomposition of semi-algebraic systems. *J. Symb. Comp*, 2011. To appear.

[6] C. Chen, J.H. Davenport, M. Moreno Maza, B. Xia, and R. Xiao. Computing with semi-algebraic sets represented by triangular decomposition. In *Proceedings ISSAC 2011*, 2011.

[7] J. Carette. A canonical form for piecewise defined functions. In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 77–84, 2007.

[8] C. Chen, O. Golubitsky, F. Lemaire, M. Moreno Maza, and W. Pan. Comprehensive triangular decomposition. In *Proc. of CASC'07*, volume 4770 of *Lecture Notes in Computer Science*, pages 73–101, 2007.

[9] C. Chen and M. Moreno Maza. Algorithms for computing triangular decompositions of polynomial systems. In *Proceedings of the 36th international symposium on symbolic and algebraic computation*, ISSAC '11, pages 83–90, New York, NY, USA, 2011. ACM.

[10] C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *ISSAC'09*, pages 95–102, 2009.

[11] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.

[12] X. Dahan, A. Kadri, and É. Schost. Bit-size estimates for triangular sets in positive dimension. *J. of Complexity*, 2011. To appear.

[13] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *J. Symbolic Comp.*, 5:29–35, 1988.

[14] J.H. Davenport. The geometry of $\mathbb{C}^n$ is important for the algebra of elementary functions. In M. Jowsig and N. Takayama, editors, *Algebra geometry and software systems*, pages 207–224. Springer Verlag, 2003.

[15] A. Dolzmann, A. Seidl, and T. Sturm. Efficient projection orders for CAD. In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 111–118, 2004.

[16] H. Hong *et al.* QEPCAD B, `www.usna.edu/Users/cs/qepcad/`.

[17] M. Kalkbrener. *Three contributions to elimination theory.* PhD thesis, Johannes Kepler University, Linz, 1991.

[18] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[19] N. Phisanbut. *Practical simplification of elementary functions using cylindrical algebraic decomposition.* PhD thesis, University of Bath, 2011.

[20] N. Phisanbut, R. J. Bradford, and J. H. Davenport. Geometry of branch cuts. *Communications in Computer Algebra*, 44:132–135, 2010.

[21] A. Strzeboński. Solving systems of strict polynomial inequalities. *J. Symb. Comput.*, 29(3):471–480, 2000.

[22] D. K. Wang. The `Wsolve` package. http://www.mmrc.iss.ac.cn/∼dwang/wsolve.txt.

[23] D. M. Wang. `Epsilon 0.618.` http://www-calfor.lip6.fr/∼wang/epsilon.

[24] W. T. Wu. A zero structure theorem for polynomial equations solving. *MM Research Preprints*, 1:2–12, 1987.

[25] L. Yang and B. Xia. Real solution classifications of a class of parametric semi-algebraic systems. In *Proc. of the A3L'05*, pages 281–289, 2005.

[26] L. Yang and J. Zhang. Searching dependency between algebraic equations: an algorithm applied to automated reasoning. Technical Report IC/89/263, International Atomic Energy Agency, Miramare, Trieste, Italy, 1991.

[27] T. Zhang and B. Xia. A new method for real root isolation of univariate polynomials. *Mathematics in Computer Science*, 1:305–320, 2007.

Changbo Chen
University of Western Ontario
Canada
e-mail: `cchen252@csd.uwo.ca`

James H. Davenport
University of Bath
United Kingdom
e-mail: `jhd@cs.bath.ac.uk`

François Lemaire
Université de Lille 1
France
e-mail: `Francois.Lemaire@lifl.fr`

Marc Moreno Maza
University of Western Ontario
Canada
e-mail: `moreno@csd.uwo.ca`

Nalina Phisanbut
University of Bath
United Kingdom
e-mail: `cspnp@bath.ac.uk`

Bican Xia
Peking University
China
e-mail: `xbc@math.pku.edu.cn`

Rong Xiao
University of Western Ontario
Canada
e-mail: `rong@csd.uwo.ca`

Yuzhen Xie
University of Western Ontario
Canada
e-mail: `yxie@csd.uwo.ca`