Cylindrical Algebraic Decomposition in the RegularChains Library

Changbo Chen[1] and Marc Moreno Maza[2]

[1] Chongqing Key Laboratory of Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, China
changbo.chen@hotmail.com,
http://www.orcca.on.ca/∼cchen
[2] ORCCA, University of Western Ontario, Canada
moreno@csd.uwo.ca,
http://www.csd.uwo.ca/∼moreno

Abstract. Cylindrical algebraic decomposition (CAD) is a fundamental tool in computational real algebraic geometry and has been implemented in several software. While existing implementations are all based on Collins' projection-lifting scheme and its subsequent ameliorations, the implementation of CAD in the RegularChains library is based on triangular decomposition of polynomial systems and real root isolation of regular chains. The function in the RegularChains library for computing CAD is called CylindricalAlgebraicDecompose. In this paper, we illustrate by examples the functionality, the underlying theory and algorithm, as well the implementation techniques of CylindricalAlgebraicDecompose. An application of it is also provided.

Keywords: Cylindrical algebraic decomposition, triangular decomposition, RegularChains

1 Introduction

Cylindrical Algebraic Decomposition (CAD) was introduced by Collins [6] for solving the real quantifier elimination (QE) problem. A CAD is a partition of the real space $\mathbb{R}^n$ into finitely many connected semi-algebraic subsets, called cells, such that any two cells are cylindrically arranged, that is the projection of them onto any low dimensional space are either disjoint or identical. Let F be a set of polynomials with real number coefficients in n variables. A CAD is called F-invariant if any polynomial of F is sign-invariant on any cell of the CAD. The rich properties of CAD make it become a fundamental tool in studying real algebraic geometry. Despite of the doubly exponential running time complexity in the worst case, the practical performance of CAD has been improved by many researchers [1]. Accompanying with these improvements, many software have been implemented to compute CADs, among which the best known are QEPCAD, Mathematica and Reduce.

Most of the implementations for computing CADs are based on the original projection-lifting framework of Collins. In the projection phase, starting from an

input set F of polynomials in n variables, one applies a pre-defined projection operator P to F and obtains a set $P(F)$ of polynomials in $n-1$ variables. This process is recursively done for $P(F)$ until a set of univariate polynomials are computed. Let A be the set of all polynomials generated in such process. The projection phase guarantees that the zero sets of polynomials in A naturally defines a CAD of $\mathbb{R}^n$. The work of the lifting phase is to compute an explicit representation from such an implicitly defined CAD. In the base case, the real zeros of univariate polynomials in A are isolated, which divides the real line into disjoint open intervals. The real zeros and the intervals together form a CAD of $\mathbb{R}^1$. Assume a CAD of $\mathbb{R}^{n-1}$ is computed. For each cell C of it, one evaluates the polynomials of A in n variables at a sample point of the cell and obtains a set of univariate polynomials. Isolating the real roots of them allows one to deduce all the cells of the CAD of $\mathbb{R}^n$ whose projection are C.

In [4], a different method for computing CADs was proposed. It first produces a cylindrical decomposition of the complex space (CCD) through the computation of regular GCDs, and then refines the CCD into a CAD of the real space by isolating real roots of univariate polynomials with real algebraic number coefficients encoded by regular chains and isolating boxes. The efficiency of it was greatly improved in [2], where the computation of CCD is replaced by a new incremental algorithm. Both algorithms are based on triangular decomposition of polynomial systems and real root isolation of regular chains. For this reason, we call the CAD as computed in [4, 3] RC-CAD. The algorithm of [4] was firstly implemented in the `RegularChains` library of Maple 14. The implementation was revised in Maple 16 and has remained the same in the subsequent versions of Maple. The algorithm of [3] was implemented in the `RegularChains` library, but not shipped with Maple. Any update of the implementation of both algorithms are now available through the `RegularChains` library (`http://www.regularchains.org`).

The purpose of this paper is to lift the veil of the implementation of RC-CAD. In the `RegularChains` library, the function for computing CCD and CAD are respectively CylindricalDecompose and CylindricalAlgebraicDecompose. In Section 2, we illustrate by examples how to use the two functions. In Section 3, we explain the underlying theory and algorithms of RC-CAD. The technical challenges for implementing the algorithms and our solutions are also discussed. Finally, in Section 4, we report on an application of our software.

2 Functionality

A cylindrical decomposition of the complex space, or complex cylindrical decomposition (CCD) is a partition of the complex space into cylindrically arranged constructible sets, each of which is the zero set of a regular system. Figure 1 shows a CCD represented in a piecewise format. Here the variable order is $x < y$. The "1's" in the formula are placeholders having no meanings. Such format can be interpreted as a tree shown in Figure 2, where each branch in the piecewise format corresponds to a path of the tree. The constraints on a path of the tree

```
> R := PolynomialRing([y, x]):
  F := [y^2-x]:
  CylindricalDecompose(F, R, output=piecewise);
```

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

Fig. 1. Compute complex cylindrical decomposition by CylindricalDecompose.

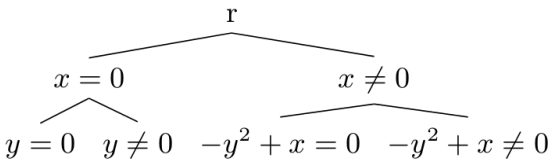

Fig. 2. A complex cylindrical tree.

form a regular system. Such a CCD is sign-invariant w.r.t. $f := x^2 + y^2 - 1$, that is for a given path of the tree from the root to a leaf, either f vanishes at all points of the path or f vanishes at none of the points of the path.

An F-sign invariant CAD is depicted in Figure 3. The CylindricalAlgebraicDecompose command supports several different input and output formats. The input can be a list of polynomials, as shown in Figure 1, as well as a list of polynomial constraints, as shown in Figure 5. The format 'output'='cadcell' allows only true cells satisfying the input constraints are displayed. To get a sample point of a CAD cell, the function SamplePoints can be called. Here no cost occurs since sample points are computed along the computation of the CAD and are stored in the type cad_cell. A sample point is encoded by the type box, which is represented by a regular chain and an isolation cube. Such a representation allows one to easily test if the sign of a polynomial at the sample point by calling the function SignAtBox.

Due to the intrinsic doubly exponential complexity of CAD, it is not uncommon that the number of CAD cells is numerous. To get a compact output for the purpose of "solving" the input semi-algebraic system, the option 'output'='rootof' can be used. In this case, the solver will try to merge the adjacent CAD cells as much as possible in order to get a simple formula. See Figure 4 for an example.

```
> R := PolynomialRing([y, x]):
  F := [y^2-x]:
  CylindricalAlgebraicDecompose(F,R,output=piecewise);
```

$$\begin{cases} 1 & x < 0 \\ \begin{cases} 1 & y < 0 \\ 1 & y = 0 \\ 1 & 0 < y \end{cases} & x = 0 \\ \begin{cases} 1 & y < -\sqrt{x} \\ 1 & y = -\sqrt{x} \\ 1 & \mathrm{And}\left(-\sqrt{x} < y, y < \sqrt{x}\right) \\ 1 & y = \sqrt{x} \\ 1 & \sqrt{x} < y \end{cases} & 0 < x \end{cases}$$

Fig. 3. Compute CAD by CylindricalAlgebraicDecompose.

```
> R := PolynomialRing([y, x]):
  CylindricalAlgebraicDecompose([x^2+y^2-1<=0],R,output=
  rootof);
```

$$\left[\left[\mathrm{And}(-1 \le x, x \le 1), \mathrm{And}\left(-\sqrt{-x^2+1} \le y, y \le \sqrt{-x^2+1}\right)\right]\right]$$

Fig. 4. Solve semi-algebraic systems by CylindricalAlgebraicDecompose.

```
> R := PolynomialRing([y, x]):
  cad := CylindricalAlgebraicDecompose([x^2+y^2-1=0, x*y-1/2=0],R,output=cadcell);
  Display(cad, R);
  sp := map(SamplePoints, cad, R);
  Display(sp, R);
  s := SignAtBox(x^2+y^2-2, sp[1], R);
```

$$cad := [cad\_cell, cad\_cell]$$

$$\left[\left\{\begin{array}{l} y = \frac{1}{2x} \\ x = -\frac{1}{2}\sqrt{2} \end{array}\right., \left\{\begin{array}{l} y = \frac{1}{2x} \\ x = \frac{1}{2}\sqrt{2} \end{array}\right.\right]$$

$$sp := [box, box]$$

$$\left[\left\{\begin{array}{l} y = \left[-\frac{46341}{65536}, -\frac{1482907}{2097152}\right] \\ x = \left[-\frac{46341}{65536}, -\frac{741455}{1048576}\right] \end{array}\right., \left\{\begin{array}{l} y = \left[\frac{185363}{262144}, \frac{741457}{1048576}\right] \\ x = \left[\frac{741455}{1048576}, \frac{46341}{65536}\right] \end{array}\right.\right]$$

$$s := -1$$

Fig. 5. Compute CAD of a semi-algebraic system.

3 Underlying theory and technical contribution

In this section, we explain the algorithms and theory underlying RC-CAD as well as a universe tree data structure for implementing RC-CAD. There are two important ingredients in RC-CAD, namely a routine for computing CCD, and a routine for turning a CCD into a CAD.

Two algorithms have been proposed for computing CCD. The first one was proposed in [4]. It has two phases: InitialPartition and MakeCylindrical. Let F be a set of polynomials in n variables. InitialPartition partitions $\mathbb{C}^n$ into a family $\mathcal{C}$ of constructible sets, called cells, each of which is the zero set of a regular system. Moreover, for a given cell C and any polynomial $f \in F$, either f vanishes at all points of C or f vanishes at no points of C. The cells in the output of InitialPartition are not necessarily cylindrically arranged. The cylindricity is achieved in a top-down fashion. The collection $\mathcal{C}$ of constructible sets is refined into a new family $\mathcal{D}$ of disjoint constructible sets, such that the projection of any two cells of $\mathcal{D}$ onto $\mathbb{C}^{n-1}$ are either identically equal or disjoint. By making a recursive call to MakeCylindrical on the projection of $\mathcal{D}$ on $\mathbb{C}^{n-1}$, one finally deduces a collection of cylindrically arranged cells. If the option 'method'='recursive' is enabled, CylindricalDecompose will use such an algorithm.

A second one was proposed in [3]. Let F be a set of m polynomials in n variables. The algorithm first builds an initial CCD $\mathcal{C}_0$, which consists of only one cell $\mathbb{C}^n$. It then pops a polynomial f_1 from F and refines $\mathcal{C}_0$ into $\mathcal{C}_1$ such that $\mathcal{C}_1$ is sign-invariant w.r.t. f_1. If F is not empty, a new polynomial f_2 is popped and $\mathcal{C}_1$ is refined w.r.t. f_2. This process is repeated until F gets empty. Making a CCD sign-invariant w.r.t. a polynomial is reduced to making every cell of the CCD sign-invariant w.r.t. a polynomial. This process has to preserve cylindricity, which is achieved by a refinement operation called IntersectPath in [3].

Let's illustrate this incremental algorithm by an example. Let $F := \{y^2 - x, x^2 + y^2 - 1\}$, where $f_1 = y^2 - x$, $f_2 = x^2 + y^2 - 1$. The evolution of the CCD tree during the computation is depicted by Figure 6. The first one is the initial tree. In the second tree, the node "any x" splits into two nodes to make the discriminant of f_1 w.r.t. y sign-invariant. In the third tree, the nodes "any y" split to make f_1 sign-invariant. Moreover, when $x = 0$, f_1 is replaced by its squarefree part modulo $x = 0$. In the fourth tree, the node $y \neq 0$ splits to make f_2 sign-invariant. In the fifth tree, the node $x \neq 0$ splits to make the resultant of f_1 and f_2 w.r.t. y sign-invariant. In the sixth tree, the node $x(x^2 + x - 1) \neq 0$ splits to make the discriminant of f_2 w.r.t. y sign-invariant. Finally the nodes $f_1 \neq 0$ splits to make f_2 sign-invariant.

The operation turning a CCD into a CAD is called MakeSemiAlgebraic. It is implemented in a recursive manner. For the base case $n = 1$, it collects all the polynomials in the equational nodes of the CCD tree, does univariate real root isolation, and picks sample points. Let $\mathcal{C}_{n-1}$ be a CAD of $\mathbb{R}^{n-1}$ derived from a CCD $\mathcal{T}$ of $\mathbb{C}^{n-1}$. Let C be a cell of $\mathcal{C}_{n-1}$ derived from a cell D of $\mathcal{T}$. Let s be a sample point of C. Let P be the set of polynomials appearing in the equational children of D. To compute the cells of a CAD of $\mathbb{R}^n$ whose projection onto $\mathbb{R}^{n-1}$ is C (these cells form a stack over C), one isolates the real

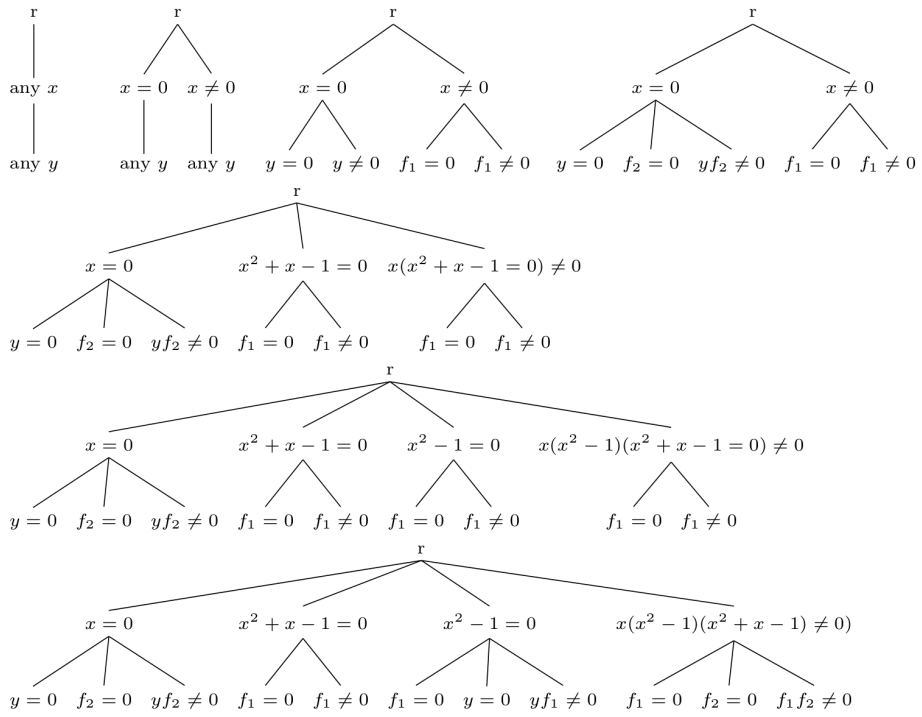

Fig. 6. The process of computing a CCD.

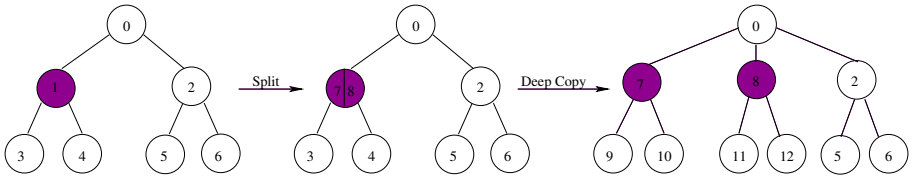

Fig. 7. The universe tree and the Split operation.

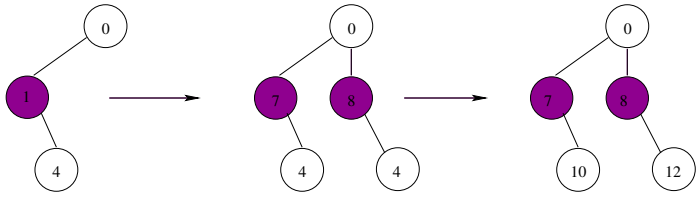

Fig. 8. A sub-tree evolves with the universe tree.

roots of univariate polynomials $\tilde{p} := p([x_1, \ldots, x_{n-1}] = s, x_n)$, $p \in P$. Here the substitution is carried with interval arithmetic since the coordinate of s may be irrational real algebraic numbers. As a result, the coefficients of the univariate polynomials $\tilde{p}$ are approximated by intervals whose width can be reduced to arbitrarily small. If the width of the intervals are sufficiently small, the real roots of $\tilde{p}$ can be deduced from its sleeve polynomials, which are univariate polynomials with rational number coefficients, thanks to the fact that s is encoded by a regular chain and a box [8, 9, 5].

The data structure supporting the implementations of CCD and CAD is a *universe* tree [3]. It is a tree data structure equipped with a Split operation (7). The Split operation is frequently used in the incremental algorithm [3] for computed CCDs, where the nodes in a complex cylindrical tree are split to make new added or generated polynomials sign-invariant and maintain the tree cylindrical. This process is illustrated by Figure 6. As a result, the universe tree is always kept to be updated. Suppose now we'd like to do several operations on a sub-tree. In order to maintain data consistency, the sub-tree has to been updated according to the universe. Note that it is fine to only update the node to be immediately worked on. The update of the sub-tree is illustrated by Figure 8.

4 Application

In this section, we show the application of RC-CAD on solving two challenges [7].

Challenge 1 *Demonstrate automatically the truth of Formula 1 over reals.*

$$\begin{aligned}\forall x_1 \forall x_2 \forall y_1 \forall y_2\ (x_1^2 + y_1^2 > 1 \wedge x_2^2 + y_2^2 > 1 \wedge x_1 + \tfrac{x_1}{x_1^2+y_1^2} = x_2 + \tfrac{x_2}{x_2^2+y_2^2} \wedge \\ y_1 - \tfrac{y_1}{x_1^2+y_1^2} = y_2 - \tfrac{y_2}{x_2^2+y_2^2} \implies (x_1 = x_2 \wedge y_1 = y_2))\end{aligned} \tag{1}$$

Challenge 2 *Demonstrate automatically the truth of Formula 2 over reals.*

$$\begin{aligned}\forall x_1 \forall x_2 \forall y_1 \forall y_2\ (y_1 > 0 \wedge y_2 > 0 \wedge x_1 + \tfrac{x_1}{x_1^2+y_1^2} = x_2 + \tfrac{x_2}{x_2^2+y_2^2} \wedge \\ y_1 - \tfrac{y_1}{x_1^2+y_1^2} = y_2 - \tfrac{y_2}{x_2^2+y_2^2} \implies (x_1 = x_2 \wedge y_1 = y_2))\end{aligned} \tag{2}$$

The first challenge is solved by RC-CAD within one minute on a laptop (Intel i7, 8Gb RAM, Ubuntu), as shown by Figure 9. The second challenge can be solved in a similar way in about 20 seconds, which is not shown here limited to space. Both answers are true. To achieve this, a universal quantifier elimination problem is converted to an existential one using the following equivalence:

$$\forall \mathbf{x}(A \implies (B \wedge C)) \text{ iff } \neg\exists \mathbf{x} \neg(A \implies (B \wedge C)) \text{ iff } \neg\exists \mathbf{x}\,((A \wedge \neg B) \vee (A \wedge \neg C))$$

As a result, Formula 1 is true if and only if none of the two semi-algebraic systems $sys1$ and $sys2$ in Figure 9 has solutions. To solve a semi-algebraic system, the command CylindricalAlgebraicDecompose is called with three options. The option 'precondition'='TD' allows to precondition the input system by means of triangular decomposition. The option 'partial'='true' uses partial lifting.

Acknowledgements Supported by NSFC (11301524) and CSTC (cstc2013jjys0002).

```
> t0 := time():
  f_1 := x_1+x_1/(x_1^2+y_1^2)-(x_2+x_2/(x_2^2+y_2^2)):
  f_2 := y_1-y_1/(x_1^2+y_1^2)-(y_2-y_2/(x_2^2+y_2^2)):
  g_1 := numer(normal(f_1)): g_2 := numer(normal(f_2)):

  sys := &not( (x_1^2+y_1^2-1>0) &and (x_2^2+y_2^2-1>0) &and (g_1=0) &and (g_2=0)
  &implies  (  (x_1=x_2) &and (y_1=y_2)   )  ):
  lsas := RegularChains:-TRDcadLogicFormulaToLsas(sys):
  nops(lsas); sys1 := lsas[1]: sys2 := lsas[2]:

  vars := SuggestVariableOrder(sys1): R := PolynomialRing(vars):
  out1 := CylindricalAlgebraicDecompose(sys1, R, output=rootof, precondition='TD',
  partial='true');

  vars := SuggestVariableOrder(sys2): R := PolynomialRing(vars):
  out2 := CylindricalAlgebraicDecompose(sys2, R, output=rootof, precondition='TD',
  partial='true');

  evalb( nops(out1)=0 and nops(out2)=0 ); t1 := time() - t0;
                                          2
                                    out1 := [ ]
                                    out2 := [ ]
                                       true
                                    t1 := 55.197
```

Fig. 9. Use CAD to solve Problem Joukowski-a.

References

1. B. Caviness and J. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation*. Springer, 1998.
2. C. Chen and M. Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *Journal of Symbolic Computation*, 47(6):610 – 642, 2012.
3. C. Chen and M. Moreno Maza. An incremental algorithm for computing cylindrical algebraic decompositions. *Proc. ASCM '12*, Oct. 2012.
4. C. Chen, M. Moreno Maza, B. Xia, and L. Yang. Computing cylindrical algebraic decomposition via triangular decomposition. In *ISSAC'09*, pages 95–102, 2009.
5. J. S. Cheng, X. S. Gao, and C. K. Yap. Complete numerical isolation of real zeros in zero-dimensional triangular systems. In *ISSAC*, pages 92–99, 2007.
6. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Springer Lecture Notes in Computer Science*, 33:515–532, 1975.
7. J. H. Davenport, R. J. Bradford, M. England, and D. J. Wilson. Program verification in the presence of complex numbers, functions with branch cuts etc. In *SYNASC*, pages 83–88, 2012.
8. Z. Y. Lu, B. He, Y. Luo, and L. Pan. An algorithm of real root isolation for polynomial systems. In D. M. Wang and L. Zhi, editors, *Proceedings of Symbolic Numeric Computation 2005*, pages 94–107, 2005.
9. B. Xia and T. Zhang. Real solution isolation using interval arithmetic. *Comput. Math. Appl.*, 52(6-7):853–860, 2006.