Towards Incremental Cylindrical Algebraic Decomposition in Maple

Cowen-Rivers, A. & England, M.

ESSN 1613-0073
Publisher: CEUR Workshop Proceedings

Towards Incremental Cylindrical Algebraic Decomposition in Maple

Alexander I. Cowen-Rivers[1] and Matthew England[2]

[1] University College London, London, UK
a.cowen-rivers.17@ucl.ac.uk
[2] Coventry University, Coventry, UK
Matthew.England@coventry.ac.uk

Abstract. Cylindrical Algebraic Decomposition (CAD) is an important tool within computational real algebraic geometry, capable of solving many problems for polynomial systems over the reals. It has long been studied by the Symbolic Computation community and has found recent interest in the Satisfiability Checking community.

The present report describes a proof of concept implementation of an Incremental CAD algorithm in Maple, where CADs are built and then refined as additional polynomial constraints are added. The aim is to make CAD suitable for use as a theory solver for SMT tools who search for solutions by continually reformulating logical formula and querying whether a logical solution is admissible.

We describe experiments for the proof of concept, which clearly display the computational advantages compared to iterated re-computation. In addition, the project implemented this work under the recently verified Lazard projection scheme (with corresponding Lazard valuation).

1 Introduction

We aim to adapt Cylindrical Algebraic Decomposition (CAD) for use with SMT-solvers [2], as part of the SC^2 Project which seeks to build collaborations between researchers in Symbolic Computation and Satisfiability Solving [1]. We report on an implementation of incremental CAD in Maple which can build a CAD and then refine it by incrementally adding polynomials. The implementation is restricted to Open CAD (full dimensional cells) and the addition of constraints (an SMT solver would also want the ability to remove them). While a proof of concept implementation, experiments show clear savings on offer.

Another minor contribution of the present work is an implementation of the Lazard projection operator (and corresponding valuation for lifting). The operator was proposed in 1994 [8], but shortly after a flaw was found in its proof of correctness (see [10] for details). However, recent work [11] has given an alternative proof (which necessitates some changes to the lifting stage). It is now the smallest known complete CAD projection operator.

1.1 Terminology

We work over n-dimension real space $\mathbb{R}^n$ in which there is a variable ordering.

Definition 1 A *decomposition* of the space $\mathcal{X} \subset \mathbb{R}^n$ is a finite collection of disjoint regions, called *cells*, whose union is $\mathcal{X}$.

Definition 2 A set is *semi-algebraic* if it can be constructed by finitely many applications of *union*, *intersection* and *complementation* operations on sets of the form $\{x \in \mathbb{R}^n \mid \mathbf{f}(x) \geq 0\}$ where $\mathbf{f} \in \mathbb{R}[x_1, \cdots, x_n]$.

Definition 3 A decomposition $\mathcal{D}$ is *algebraic* if each of its components $x \in \mathcal{D}$ is a semi-algebraic set.

Definition 4 A finite partition of $\mathcal{D}$ of $\mathbb{R}^n$ is called a *cylindrical decomposition* of $\mathbb{R}^n$ if the projections of any two cells onto any lower dimensional coordinate space with respect to the variable ordering are either equal or disjoint.

Thus a *cylindrical algebraic decomposition* (CAD) satisfies Definitions $1-4$.

Definition 5 A CAD is *sign-invariant* with respect to a set of input polynomials if each polynomial has a constant sign (positive, negative or zero) on each cell.

CADs may be produced with other invariant properties (see for example [5], [3]) but we assume sign-invariance in the present work. Each CAD cell is equipped with: a *cell index* which is a list of integers that defines the position of a cell in the decomposition; and a *sample point* of the cell. The cells we produce also come with a *cell description*: a cylindrical formula, that is, a description of the cell as a sequence of conditions on ordered variables of the form $\ell(x_1, \ldots, x_k) < x_{k+1} < u(x_1, \ldots, x_k)$, where ℓ and u may be $\pm\infty$.

CADs are traditionally produced through a two-stage process: first projection identifies polynomials of importance for the invariance property and then lifting incrementally builds CADs of $\mathbb{R}^k$ for $k = 1, \ldots, n$ according to these polynomials. Decompositions are performed by working at a sample point of a cell, reducing multivariate polynomials to univariate and then decomposing according to the output of real root isolation. For a fuller introduction see the lecture notes [7].

1.2 Example

We give a visual example[3]. The gingerbread face in Figure 1 is formed by four closed curves, each of which defined by a bi-variate polynomial equation. A corresponding sign-invariant CAD of $\mathbb{R}^2$ is visualised in Figure 2. We label the 37 open cells (those of two dimensions). There are a further 28 partially open (1-dimensional line segments) and 28 closed cells (isolated points) giving 93 CAD cells in total. Of course, in many industrial and SMT applications the polynomials will not form such aesthetically pleasing geometric shapes.

[3] Inspired by `http://planning.cs.uiuc.edu/node296.html`

Fig. 1. Gingerbread face formed by 4 bivariate polynomials

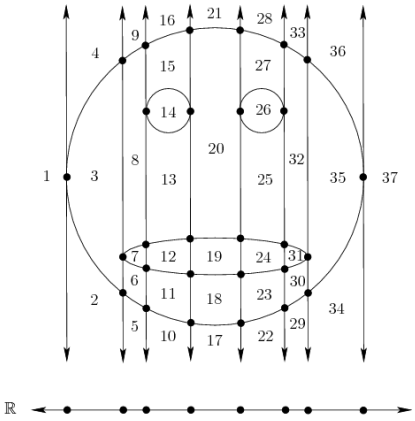

Fig. 2. A CAD of Fig 1. Numbers correspond to open (full dimensional) cells.

1.3 Report plan

We aim to work with CADs that change incrementally by constraint. I.e. given a CAD of $\mathbb{R}^n$ sign-invariant for polynomials $\{f_1, \ldots, f_m\}$ we aim to adapt this to one sign-invariant for $\{f_1, \ldots, f_m, f_{m+1}\}$ or $\{f_1, \ldots, f_{m-1}\}$. The present report deals only with the first problem. Such incrementality is needed to use CAD in SMT, and could also benefit CAD directly as a way of reducing the search space. We proceed by considering the changes required in Projection (Section 2) and Lifting (Section 3) alongside the issues in using the Lazard projection operator. We finish with a summary and plans for future work in Section 4. A larger report with additional details we do not have room for here is available on arXiv [4].

2 Projection

2.1 Lazard projection

The present project built upon code from the ProjectionCAD package [6] for Maple. This implemented the McCallum family of projection operators [9], [3] and our first step was to adapt this for the Lazard projection scheme. All projection operators take a set of polynomials and produce another set in one less variable. The Lazard operator is essentially a subset of the McCallum operator. Both take discriminants and cross-resultants of the input polynomials. The Lazard operator then takes in addition leading and trailing coefficients while the McCallum operator takes all coefficients. Thus adaptations to the projection algorithms were fairly minimal here (see [4] for algorithms). The main computational differences occur in the lifting stage as discussed later.

2.2 Worked example

We describe a worked example to illustrate what projection does and to use later to illustrate incremental projection. We work with a system of two polynomials $\mathbf{F}_1$ and seek a sign-invariant CAD:

$$\mathbf{F}_1 = \{\underbrace{x_1^2 + x_2^2 - 1}_{f_1}, \underbrace{x_1^3 - x_2^2}_{f_2}\}. \tag{1}$$

Since the problem involves only two variables, we need only a single projection (which we do with respect to x_2). The leading coefficients are constant, but the trailing coefficients clearly identify the points on the x_1 line at 0 and ± 1. The discriminants do not identify anything more but the resultant

$$\texttt{Resultant}(f_1, f_2, x_2) = (x_1^3 + x_1^2 - 1)^2 \tag{2}$$

identifies $\pm\alpha_1 =\approx \pm 0.7549$[4]. Thus the real line is decomposed into 11 cells according to these 5 points.

Figure 3 plots the graphs of these functions along with the real roots isolated. We see they mostly correspond to geometrically relevant features ($-\alpha_1$ corresponds to an intersect in $\mathbb{C}^2$).

2.3 Incremental Lazard projection

Our second step was to adapt the Lazard Projection algorithms to calculate new projection polynomials incrementally.

We continue our example to illustrate the incremental working. Suppose we take $\mathbf{F}_1$ from above and also the polynomial forming a line, $f_3 = x_2 - x_1$ to give

$$\mathbf{F}_2 = \{\underbrace{x_1^2 + x_2^2 - 1, x_1^3 - x_2^2}_{\mathbf{F}_1}, \underbrace{x_2 - x_1}_{f_3}\}. \tag{3}$$

We will compute all the projection polynomials discussed above and some additional ones. The discriminant and leading coefficient of f_3 are constant, and the trailing coefficient identifies $x_1 = 0$ which was already present from the trailing coefficient of f_2. Similarly, the resultant of f_2 and f_3 is $x_1^2(x_1 - 1)$ identifies two more roots we saw already. However $\texttt{Resultant}(f_1, f_3, x_2) = 2x_1^2 - 1$ identifies two new points, $\pm\alpha_2 = \pm 1/\sqrt{2} \approx \pm 0.7071$.

Figure 4 shows that the two new roots correspond to the two new intersections of the straight line with the circle.

[4] We give a decimal approximation but emphasise that CAD would use the full algebraic number representation: that α_1 is the sole real root of (2) in $(0, 1)$.

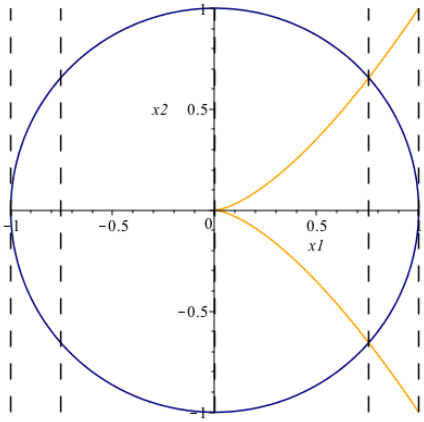

Fig. 3. The blue curve is f_1 and the orange f_2. Dotted lines show the projection roots $\in \mathcal{R}$.

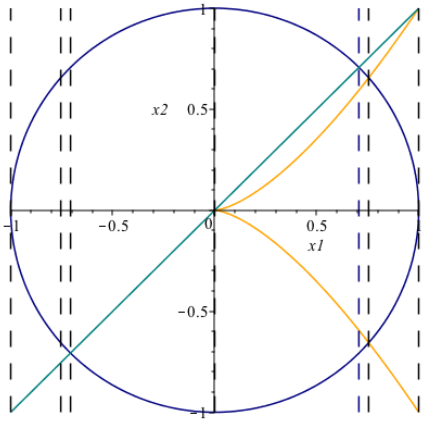

Fig. 4. As Fig. 3 but with the additional teal curve f_2 and the additional roots identified.

We developed Algorithms 1 and 2 to implement such an incremental projection. The pseudo code is closely linked to the MAPLE implementation. The former computes all the projection polynomials via calls to the latter which performs one projection. The adjustments (highlighted in blue) required to make the original projection code (see [4]) incremental were:

1. To output from `ProjectionPolysAdd` the full table of projection polynomials organised by the main variable and re-input this with incremental calls.
2. The process and pass polynomials separately into `ProjectionAdd`.
3. To calculate only the new projection polynomials and store appropriately.

2.4 Incremental projection experimental results

There were seven elements of the projection set to calculate in the original projection system `ProjL(`F_1`)` and after the addition of the extra polynomial `ProjL(`F_2`)` had 12. By avoiding full recomputation, we had to compute only the extra five. At the cell level the savings are more significant: on the real line there were 5 original roots (11 cells), and two new ones (making 15 cells).

We performed experiments to see how such savings transferred into computation time. We created examples using the random polynomial function `randpoly` in MAPLE. Testing was conducted through an external bash script creating new MAPLE instances to avoid any result caching. The testing code is available online[5].

[5] `https://github.com/acr42/InCAD.git`

Algorithm 1 ProjectionPolysAdd

Input: Set of calculated projection polynomials $prev \in \mathbb{R}[x_n, \ldots, x_1]$ and set of new polynomials $new \in \mathbb{R}[x_n, \ldots, x_1]$ (Adjustment 1)
Output: All projection polynomials $\in \mathbb{R}[x_n, \ldots, x_1]$
Procedure `ProjectionPolysAdd` (Compute all projection polynomials)
$dim \leftarrow$ Number of variables $+1$
$pset[0] = table()$
$pset[0] \leftarrow$ Primitive set from *new*, wrt variable x_n (Adjustment 2)
$pset[0] \leftarrow$ Square free basis set from $pset[0]$, wrt variable x_n
$pset[0] \leftarrow$ Set of factors from $pset[0]$, wrt variable x_n
$cont \leftarrow$ Set of contents of *new*, wrt x_n (Adjustment 2)
for `i from 1 to dim-1` **do**
    $out \leftarrow$ `ProjectionAdd`$(prev[i-1], pset[i-1])$ (Adjustment 2)
    $pset[i] \leftarrow (out \cup cont)$
    $cont \leftarrow$ Content set from $pset[i]$, wrt variable x_{n-i}
    $pset[i] \leftarrow$ Prime set from $pset[i]$, wrt variable x_{n-i}
    $pset[i] \leftarrow$ Square free basis from set $pset[i]$, wrt variable x_{n-i}
    $pset[i] \leftarrow$ Set factors from $pset[i]$, wrt variable x_{n-i}
end for
$pset[dim-1] \leftarrow$ Remove constant multiples from $pset[[dim-1]$
$ret \leftarrow pset[[dim-1]$
return *pset* (Adjustment 1)

Bivariate polynomials We created 60 pairs of bivariate polynomials, considered first finding the projection of one and then incrementing the projection by including the other. On average it was 16% faster to increment compared to computing the projection for both polynomials together. However, there was a large variance: the cases which were faster were on average 55% faster, while a small number of cases were slower, by as much as 87%.

Projection	Results			
	Classical	Incremental		
`Variance`	0.0004660s	0.0006425s	**27.46%**	Larger
`Mean`	0.03743s	0.0315s	**15.85%**	Faster
`Lower Quartile`	0.024s	0.008s	**66.66%**	Faster
`Median`	0.0285s	0.015s	**47.39%**	Faster
`Upper Quartile`	0.03675s	0.05525s	**50.34%**	Slower

Algorithm 2 ProjectionAdd

Input: Sets of polynomials $new = \{f_1, \ldots, f_m\} \in \mathbb{R}[x_n, \ldots, x_1]$, $old \in \mathbb{R}[x_{n-1}, \ldots, x_1]$, and variable ordering (Adjustment 2)
Output: Set of polynomials $Pset = \{p_1, \ldots, p_q\} \in \mathbb{R}[x_{n-1}, \ldots, x_1]$
Procedure `ProjectionAdd`
$Polys \leftarrow$ Primitive set from new, wrt variable x_n (Adjustment 2)
$Cont \leftarrow$ Content set from new, wrt variable x_n (Adjustment 2)
$Polys \leftarrow$ Square free basis set from $Polys$, wrt variable x_n
$Pset1 = table()$:
for `i from 1 to number of elements of` $Polys$ **do**
  $Pol \leftarrow Polys[i]$
  $clist \leftarrow$ Lazard coefficient set from Pol, wrt to x_n
  $temp \leftarrow$ Discriminant set from Pol, wrt x_n
  $temp \leftarrow$ Remove constant multiples from $temp$
  $Pset1[i] \leftarrow$ union $temp$ & $clist$
end for
$Pset2 = table()$:
for `i from 1 to number of` $Polys$ **do**
  for `j from i+1 to number of` $Polys$ **do**
    $Pset2[i, j] \leftarrow$ Resultant of $Polys[i]$ and $Polys[j]$, wrt to variable var
    $Pset2[i, j] \leftarrow$ Remove constant multiples from $Pset2[i, j]$
  end for
end for
$oldset \leftarrow old$
$Pset3 = table()$:
for `i from 1 to number of` $Polys$ **do**
  for `j from 1 to size`$(oldset)$ **do**
    $Pset3[i, j] \leftarrow$ Resultant of $Polys[i]$ and $oldset[j]$, wrt to variable var
    $Pset3[i, j] \leftarrow$ Remove constant multiples from $Pset3[i, j]$
  end for
end for(Adjustment 3)
$Pset \leftarrow$ union $(cont, Pset1, Pset2, Pset3)$
$Pset \leftarrow$ Remove constant multiples from $Pset$
return $Pset$

Trivariate polynomials We next created 80 further examples with pairs of trivariate polynomials, in this case restricting to 4 terms per polynomial. Here there were greater savings, on average 29% faster to increment than to compute altogether. There was also a smaller variance in the timings of the example set, although it was still the case that a few examples were slower to increment than recompute.

Projection	Results			
	Classical	Incremental		
`Variance`	0.002743s	0.002205s	**24.39%**	Smaller
`Mean`	0.06739s	0.04809s	**28.64%**	Faster
`Lower Quartile`	0.02475s	0.013s	**47.47%**	Faster
`Median`	0.0625s	0.035s	**44.00%**	Faster
`Upper Quartile`	0.09425s	0.07525s	**20.16%**	Faster

We suggest that the extra overheads of the incremental approach will become less important in comparison to the savings as the number of variables increase: indeed, this follows from the well-known complexity results on CAD.

3 Lifting

3.1 Lifting after Lazard projection

We had to make changes to the lifting code in ProjectionCAD, not just to allow for incrementality but also to validate the use of the Lazard projection operator [11]. The McCallum projection operator [9] is known to be incomplete if it occurs that a projection polynomial is nullified over the sample point of a cell. For example, the polynomial $(y^2 - 2)w + z(y - x^2 + x + 2)$ is nullified over a cell in (x, y, z)-space with sample point $(\sqrt{2}, -\sqrt{2}, 1)$. When lifting after McCallum projection, one must check for this situation and warn the user that above the cell in question we are not guaranteed sign-invariance.

With the Lazard operator (as proved valid in [11]) we can avoid such checks and warnings but we must do some additional work during lifting to recover information lost by nullification, as outlined in Algorithm 3. With the previous example we would first substitute for $x = \sqrt{2}$ to get $(y^2 - 2)w + z(y + \sqrt{2})$; but then before substituting for y we must divide by $y + \sqrt{2}$ to give $(y - \sqrt{2})w + z$. We can only then substitute for y to give $-2\sqrt{2}w + z$ and finally z to give $-2\sqrt{2}w + 1$. We thus must lift with respect to this univariate polynomial in w, creating necessary cell divisions that would have been lost by nullification.

This process is difficult when involving irrational sample points. However, since our prototype implementation lifts only over open cells, we have avoided such difficulties for now. Our implementation produces Open CADs, avoiding costly algebraic number calculations, but still getting a good understanding of the solution set.

Definition 6 An **Open-CAD** is produced by lifting over open intervals only[6].

Algorithm 3 Lazard valuation

Input: A polynomial $f \in \mathbb{R}[x_1, \ldots, x_d]$, and $\mathtt{sp} = [r_1, \ldots, r_{d-1}] \in \mathbb{R}^{d-1}$.
Output: List of roots.
Procedure `LazardValuation`:
Set `roots` to be an empty list
for j `from` 1 `to` $d-1$ **do**
  while $f(r_1, \ldots, r_j) = 0$ **do**
    $f \leftarrow f/(x_j - r_j)$
    substitute $x_j = r_j$ into f.
  end while
end for
return f:

We will now discuss our approach for incremental lifting, which can be thought of graphically as a form of acyclic tree merge, as shown later.

3.2 Worked example

We will describe lifting the projection polynomial system defined previously `ProjL`($\mathbf{F}_1$). Recall that we identified four points on the real line: $\{-1, 0, \alpha_1, 1\}$. Thus, we need to choose a sample value from the 9 cells in the decomposition:

$$\begin{array}{lll} a_1 = \{x_1 < -1\}, & a_2 = \{x_1 = -1\}, & a_3 = \{-1 < x_1 < 0\}, \\ a_4 = \{x_1 = 0\}, & a_5 = \{0 < x_1 < \alpha_1\}, & a_6 = \{x_1 = \alpha_1\}, \\ a_7 = \{\alpha_1 < x_1 < 1\}, & a_8 = \{x_1 = 1\}, & a_9 = \{1 < x_1\} \end{array} \tag{4}$$

We are forced to pick non-rational sample points for cell a_6 but the others can be rational. We choose sample points: $(-2, -1, -\frac{1}{2}, 0, \frac{1}{2}, \alpha_1, \frac{9}{10}, 1, 2)$.

We lift over each cell at the designated sample point by isolating real roots of the univariate polynomials in x_2 we get by from the Lazard valuation at the sample point in x_1. Below, $p_{i,j}$ denotes the polynomial acquired after applying the Lazard valuation method to the i'th sample point, on the j'th polynomial from F_1. For example, if we use sample point $-\frac{1}{2}$ for cell a_3 then polynomial f_1 becomes $\frac{1}{4} + x_2^2 - 1$ which has two real roots at $\pm\beta_0 = \pm\sqrt{3}/2 \approx \pm 0.8660$. We proceed this way to generate our CAD cells in $\mathbb{R}^2$. The structure is as shown in Figure 5. For a full list of the new cell descriptions see [4].

[6] Not actually a decomposition of $\mathbb{R}^n$ as missing boundaries of the n-dimensional cells.

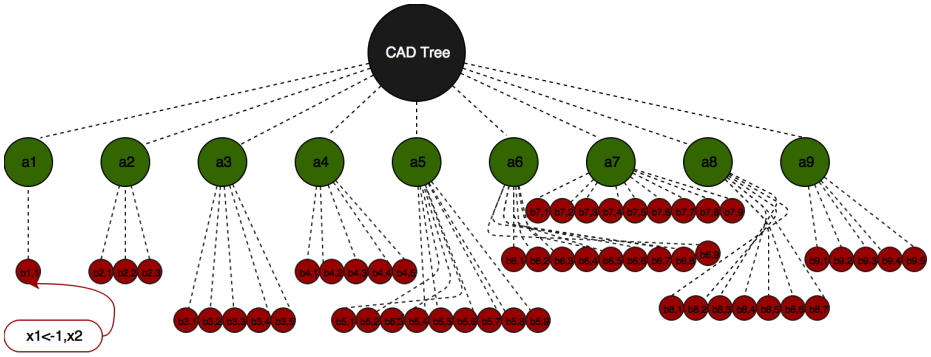

Fig. 5. CAD tree of F_1. Green nodes are in the first dimension and red the second.

3.3 Incremental Lazard lifting

The general concept of how we solved this stage of the problem was to think of it as solving a graph (tree) attachment/detachment problem. One should think of the old CAD as having a tree structure which we save: where nodes are cells; and branches link a cell to its parent (cell it projects onto), or child (decomposition in a cylinder above) cells. At each depth of the CAD/tree, say depth p, are all the cells within $\mathbb{R}^p$ before we lifted to $\mathbb{R}^{p+1}$. We go through a worked example.

We perform an incremental lift on the polynomial system $\mathbf{F}_1$, incremented by a new polynomial $f_4 = x_1^3 + x_2^2$, forming the new system (5).

$$\mathbf{F}_3 = \{\underbrace{x_1^2 + x_2^2 - 1, x_1^3 - x_2^2}_{\mathbf{F}_1}, \underbrace{x_1^3 + x_2^2}_{f_4}\} \tag{5}$$

The new system is symmetrical about the y axis as you can see in Figure 8.

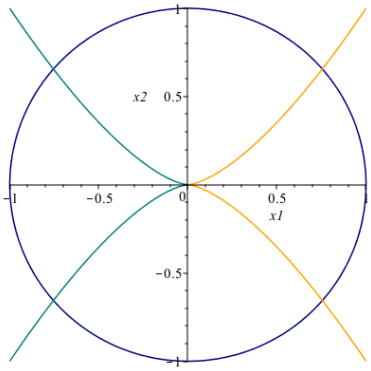

Fig. 6. The blue curve is f_1, the orange f_2 and the teal f_4.

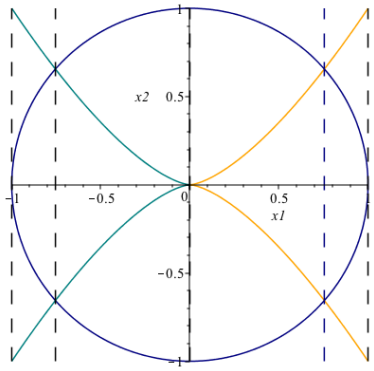

Fig. 7. Dotted lines show the projection roots.

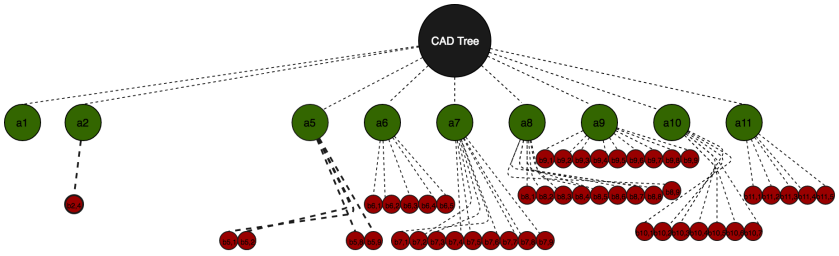

Fig. 8. CAD tree of **unchanged cells** from F_1 incremented by f_4.

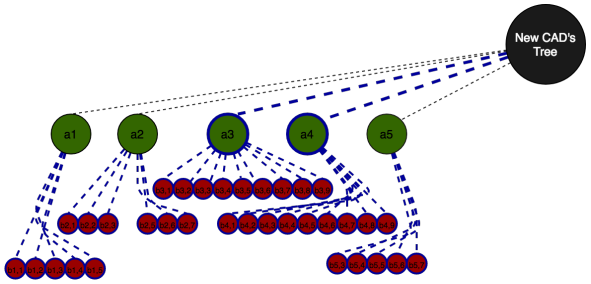

Fig. 9. CAD tree of **new cells** from F_1 incremented by f_4. Blue outlines around lines/ nodes represent new connections / cells.

We skip the projection steps[7]. The addition polynomials identify one further point on the real line: $-\alpha_1$. The decomposition of the real line is now:

$$a_1 = \{x_1 < -1\}, a_2 = \{x_1 = -1\}, a_3 = \{-1 < x_1 < -\alpha_1\},$$
$$a_4 = \{x_1 = -\alpha_1\}, a_5 = \{-\alpha_1 < x_1 < 0\}, a_6 = \{x_1 = 0\}, a_7 = \{0 < x_1 < \alpha_1\},$$
$$a_8 = \{x_1 = \alpha_1\}, a_9 = \{\alpha_1 < x_1 < 1\}, a_{10} = \{x_1 = 1\}, a_{11} = \{1 < x_1\}$$

and our sample points are: $-2, -1, -\frac{9}{10}, -\alpha_1, -\frac{1}{2}, 0, \frac{1}{2}, \alpha_1, \frac{9}{10}, 1, 2$.

We first test whether each sample point leads to new roots when lifting with f_4. If so we must refine the decomposition in the cylinder above. For example, on a_2 we have sample point $x_1 = -1$, and the Lazard valuation of f_4 is $x_2^2 - 1$ with real roots at ± 1. However, we had already identified these from other polynomials, so no change is required. However, on a_5 with sample point $-\frac{1}{2}$, f_4 evaluates to $x_2^2 - \frac{1}{8}$ and we find two new real roots. We thus refine the decomposition above.

We then lift over the new sample points with respect to all polynomials creating new decompositions. For example, at $x_1 = \frac{9}{10}$ we have two roots from the valuation of f_1 and another two from f_4 (so decomposition above into 9 cells). Figures 10-12 show the new CAD tree structure and its split into new and unchanged cells, illustrating potential savings. Full cell descriptions are in [4].

[7] See "Worked Examples" worksheet in: `https://github.com/acr42/InCAD.git`

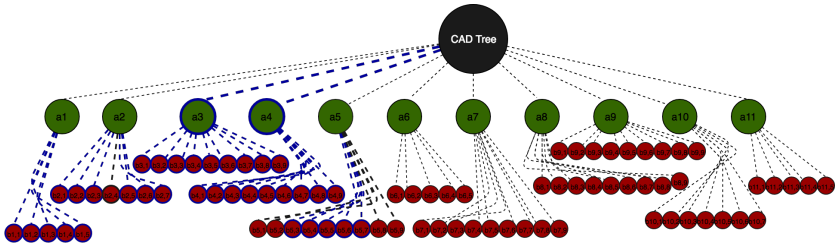

Fig. 10. CAD tree of F_1 incremented by f_4: a merger of Fig. 10 and 11.

3.4 Algorithms

The two new algorithms created for incremental lifting were `LiftSetupAdd` (Algorithm 4) concentrating on the base phase (CAD of the real line) and `LiftAdd` (Algorithm 5) which deals with the rest of the CAD. The non-incremental versions can be found in the report [4].

When incrementing the lift stage, we can think of it as starting at the root node of the old CAD tree and working our way down it, one depth level at a time, until we reach the leaves. In the process of working down the old tree, we will be creating subtrees, which will later be reconnected to the unchanged tree, to form the new incremented CAD tree.

One tree (`UnchangedCells`) is a strict subset of the old trees nodes and edges, discovered through analysing the old structure down and Lazard valuating on each cell not marked as *new*, at each depth p with new projection polynomials in R^{p+1}. Then, if there are new real roots discovered, we prune inspected cells children, and the cell is sent to the `NewCells` set for full re-computation. In the `NewCells` set, we will then use this cell to form a subtree, to later be reconnected with the `UnchangedCells` tree. Each cell in the `NewCells` list is a subtree, to later be reconnected via source indices.

When going through the old CAD tree structure, we have only two cases:

CASE 1: When a node has new children:
New real roots have been acquired from one of the new projection polynomials. We start by pruning all of the child branches in the old tree structure, by labelling them as *new*, then performing a full lift onto the set of all projection polynomials from $\mathbb{R}^k, \ldots, \mathbb{R}^n$, where k is the depth the new root was discovered. When we label a cell as *new*, effectively that halts tree growth in the `UnchangedCells` structure, so that later on we can attach the branch extensions, gained from the incremental lift. Such cells have an updated source index, as its source cell would now be saved in a new tree structure with a new index.

CASE 2: When a node has no new children.
The *new* flag is passed down to children from parents. Cells which are not *new* are stored in `UnchangedCells`. Cells here only have Lazard valuation at the new

projection polynomial (rather than all projection polynomials). If this leads to a new root, we move it over to the `NewCells` structure. Otherwise, we continue with its child cells.

When moving cells into `UnchangedCells`, we make sure that the indexing of each cell does not clash with that of the indexing in `NewCells` at each depth level in the tree. We then merge the `NewCells` and `UnchangedCells` trees, forming the full incremented CAD tree. At each stage of the lift, we merge-sort the list.

3.5 Incremental lifting experimental results

We conducted testing for the incremental lift method on the same examples used to test the incremental projection earlier.

Bivariate polynomials On average 30% faster than recomputing.

Lift	Results		
	Classical	Incremental	
`Variance`	0.003734s	0.002903s	**28.63%** Smaller
`Mean`	0.1778s	0.1240s	**30.25%** Faster
`Lower Quartile`	0.1328s	0.089s	**32.96%** Faster
`Median`	0.163s	0.1255s	**23.01%** Faster
`Upper Quartile`	0.226s	0.163s	**27.87%** Faster

Trivariate polynomials On average only 7% faster; one example 28% slower.

Lift	Results		
	Classical	Incremental	
`Variance`	0.05541s	0.06838s	**18.96%** Larger
`Mean`	0.2880s	0.2687s	**6.707%** Faster
`Lower Quartile`	0.1275s	0.0995s	**21.96%** Faster
`Median`	0.207s	0.164s	**20.77%** Faster
`Upper Quartile`	0.3605s	0.3523s	**2.29%** Faster

So the lifting code shows opposite results to projection with the savings decreasing with input size: indicating that the overheads required for the incremental work grow faster than the savings (at least for the size of examples studied).

Of course, we can also experiment with combined projection and lifting. Unsurprisingly the lifting costs dominate. For the bivariate polynomials incremental code was 37% faster but for the trivariate only 12% faster (see [4] for details). We think the reason for such drops in performance was due to poor choices of Maple's data-structure: in particular Maple lists which are implemented as immutable types meaning our edits of them caused separate lists to be created each time. Further, the project may benefit from a fully object-oriented approach. It is likely further progress could come through code re-factoring.

4 Summary and Future Work

The presented work acts as a proof of concept that incremental CAD construction in Maple is possible with savings on offer. Care needs to be given to the datatypes used. Beyond that the main areas of further work are moving out of the open case (our implementation restricted lifting to open cells), considering what happens if the new polynomial has a variable not already represented in the system, and considering incremental reduction of constraints. To remove a polynomial from the CAD means finding all those projection polynomials created from only that source polynomial (or as a resultant of that with another) and removing them and the cylinder splits their real roots caused.

Acknowledgements This work was funded by the EU's H2020 programme under grant No H2020-FETOPEN-2015-CSA 712689 (SC^2). We thank C. Brown for a tutorial on the Lazard valuation; S. Timms, J.H. Davenport and S. Forrest for useful discussions; and the organisers of the SC^2 2017 Summer School where these took place.

References

1. E. Ábrahám, J. Abbott, B. Becker, A.M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J.H. Davenport, M. England, P. Fontaine, S. Forrest, A. Griggio, D. Kroening, W.M. Seiler, and T. Sturm. SC^2: Satisfiability checking meets symbolic computation. In: Intelligent Computer Mathematics (LNCS 9791), pages 28–43. Springer International Publishing, 2016.
2. A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability (Vol. 185 Frontiers in Artificial Intelligence and Applications)*. IOS Press, 2009.
3. R. Bradford, J.H. Davenport, M. England, S. McCallum, and D. Wilson. Truth table invariant cylindrical algebraic decomposition. *J. Symb. Comp.*, 76:1–35, 2016.
4. A.I. Cowen-Rivers and M. England. Summer research report: Towards incremental Lazard cylindrical algebraic decomposition. *arXiv:1804.08564*, 2018.
5. M. England, R. Bradford, and J.H. Davenport. Improving the use of equational constraints in cylindrical algebraic decomposition. In *Proc. ISSAC '15*, pages 165–172. ACM, 2015.
6. M. England, D. Wilson, R. Bradford, and J.H. Davenport. Using the Regular Chains Library to build cylindrical algebraic decompositions by projecting and lifting. In *Proc. ICMS '14*, (LNCS 8592), pages 458–465. Springer, 2014.
7. M. Jirstrand. Cylindrical algebraic decomposition: An introduction. Course notes from Linköping University, 1995.
8. D. Lazard. An improved projection for cylindrical algebraic decomposition. *Algebraic Geometry and its Applications*, page 467–476, 1994.
9. S. McCallum. An improved projection operation for cylindrical algebraic decomposition. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts & Monographs in Symbolic Computation, pages 242–268. Springer-Verlag, 1998.
10. S. McCallum and H. Hong. On using Lazard's projection in CAD construction. *J. Symb. Comp.*, 72:65–81, 2016.

11. S. McCallum, A. Parusiński, and L. Paunescu. Validity proof of Lazard's method for CAD construction. In Press: *J. Symb. Comp.*, 2017.

Algorithm 4 LiftSetupAdd

Input: A sets of new projection polynomials new, an $oldcad$, a table of sets of all projection polynomials $psetfull$, and a variable ordering
Output: $[NewCells, OldCad, OldRoots, UnchangedCells]$ where NewCells are in the last variable $LiftIncf_2$ is information for the next lift, OldCad contains the previous CAD tree, and UnchangedCells is a subset.
Procedure `LiftSetup`
$cad \leftarrow table()$
$NewRoots \leftarrow [\,]$
$NewCells \leftarrow table()$
$UnchangedCells \leftarrow table()$
$LiftIncf_2 \leftarrow [\,]$
for i `from 1 to size`$(new[1])$ **do**
  Append to $NewRoots$ output of $RealRoots(new[1][i])$
  $NewRoots \leftarrow$ Sort in ascending order and remove duplicates
end for
$NewCells[1], UnchangedCells[1] =$ `Split`$(OldRoots, NewRoots, OldCad)$
for i `from 1 to size`$(oldcad[1])$ **do**
  for j `from 1 to size`$(new[2])$ **do**
    Add $oldcad[1][i]$ to $NewCells$
  end for
end for
for i `from 1 to size`$(NewCells[1])$ **do**
  for j `from 1 to size`$(psetfull[2])$ **do**
    Set $roots$ to real roots of `LazardValuation`$(oldcad[1][i], new[2][j]$
    Append $[[i], [roots]]$ to $LiftIncf_2$
  end for
end for
for i `from 1 to size`$(oldcad[1])$ **do**
  If cell $OldCad[1][i]$'s flag is not equal to new, then add cell to $UnchangedCells[1]$ and update index accordingly.
end for
return list of variables as in Output

Algorithm 5 LiftAdd

Input: A sets of new projection polynomials new, an $oldcad$, a table of sets of all projection polynomials $psetfull$, and a variable ordering
Output: Incremented CAD
Procedure `LiftAdd`
$[NewCells, OldCad, LiftIncf_2, Unchanged] \leftarrow$ `LiftSetupAdd`$(pset, vars)$
$dim \leftarrow$ Number of elements in $vars$
for d `from` 2 `to` $dim - 1$ **do**
  $LiftIncf_{d+1} \leftarrow [\,]$
  $NewCells[d] \leftarrow Lift(NewCells_{d-1}, LiftIncf_d\ )$
  for i `from` 1 `to size`$(OldCad[d])$ **do**
    for j `from` 1 `to size`$(new[d+1])$ **do**
      Add $oldcad[d][i]$ to $NewCells$
    end for
  end for
  for i `from` 1 `to size`$(NewCells[d])$ **do**
    for j `from` 1 `to size`$(psetfull[d+1])$ **do**
      Set $roots$ to real roots of `LazardValuation`$(OldCad[d][i], new[d+1][j]$
      Append $[[i], [roots]]$ to $LiftIncf_{d+1}$
    end for
  end for
  for i `from` 1 `to size`$(OldCad[d])$ **do**
    If cell $OldCad[d][i]$'s flag is not equal to new, then add cell to $Unchanged[d]$ and update index accordingly.
  end for
end for
$FinalUnchangedCells \leftarrow []$
$FinalUnchangedCells$ Union $Unchanged[d][f]$ for all cells f with their corresponding flags not equal to new.
$IncCAD \leftarrow FinalUnchangedCells$ Union $NewCells[d]$
return IncCAD