Simplification of Cylindrical Algebraic Formulas

Changbo Chen

Joint work with Marc Moreno Maza

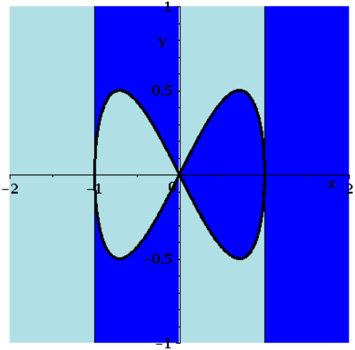
October 20, 2015

Outline

Cylindrical Algebraic Decomposition (CAD) of $\mathbb{R}^n$

A CAD of $\mathbb{R}^n$ is a partition of $\mathbb{R}^n$ s.t. each cell of it is a connected semi-algebraic subset of $\mathbb{R}^n$ and all the cells are cylindrically arranged.

Two subsets A and B of $\mathbb{R}^n$ are called cylindrically arranged if for any $1 \leq k < n$, the projections of A and B on $\mathbb{R}^k$ are either equal or disjoint.



- Introduced by G. E. Collins in 1975 and improved by many others.
- Implementation available in different software, such as QEPCAD, Mathematica, Redlog, SyNRAC, `RegularChains` (www.regularchains.org).

A CAD is naturally described by a tree

The following is a sign-invariant CAD w.r.t. $y^2 + x$.

$$T := \begin{cases} x < 0 & \begin{cases} y < -\sqrt{|x|} \\ y = -\sqrt{|x|} \\ y > -\sqrt{|x|} \wedge y < \sqrt{|x|} \\ y = \sqrt{|x|} \\ y > \sqrt{|x|} \end{cases} \\ \\ x = 0 & \begin{cases} y < 0 \\ y = 0 \\ y > 0 \end{cases} \\ \\ x > 0 & \text{any } y \end{cases}$$

Cylindrical Algebraic Formula (CAF)

A CAF associated with a CAD cell c, denoted by ϕ_c, is defined recursively.

- $n = 1$
  - If $c = \mathbb{R}$, then $\phi_c := true$.
  - If c is a point α, then define $\phi_c := x_1 = \alpha$.
  - If c is an open interval $(\alpha, \beta) \neq \mathbb{R}$, then $\phi_c := x_1 > \alpha \wedge x_1 < \beta$.
- $n > 1$. Let c_{n-1} be the projection of c onto $\mathbb{R}^{n-1}$.
  - If $c = c_{n-1} \times \mathbb{R}$, then define $\phi_c := \phi_{c_{n-1}}$.
  - If c is an θ_i-section, then $\phi_c := \phi_{c_{n-1}} \wedge x_n = \theta_i$.
  - If c is an (θ_i, θ_{i+1})-sector, then $\phi_c := \phi_{c_{n-1}} \wedge x_n > \theta_i \wedge x_n < \theta_{i+1}$.

Let S be a set of disjoint cells in a CAD. If $S = \emptyset$, $\phi_S := false$.
Otherwise, a CAF associated with S is defined as $\phi_S := \vee_{c \in S} \phi_c$.

Example

A CAF associated with the closed unit disk $x^2 + y^2 \leq 1$ is as below.

$$
\begin{aligned}
(x = -1 \wedge y = 0) \quad & \vee \quad (-1 < x \wedge x < 1 \wedge y = -\sqrt{1 - x^2}) \\
& \vee \quad (-1 < x \wedge x < 1 \wedge -\sqrt{1 - x^2} < y \wedge y < \sqrt{1 - x^2}) \\
& \vee \quad (-1 < x \wedge x < 1 \wedge y = \sqrt{1 - x^2}) \\
& \vee \quad (x = 1 \wedge y = 0)
\end{aligned}
$$

Pros and Cons of the CAF representation

Pros

- The projection of a CAF onto any lower-dimensional space can be easily read off from the CAF itself.
- For CAD-based QE, the CAF representation saves the cost of introducing augmented projection factors.
- A CAF naturally exhibits case distinctions.
- Each atomic formula of a CAF has the convenient format $x \ \sigma \ E$, and thus provides the specific value of each coordinate.

Cons

- Indexed root expressions are not globally defined.
- A CAD-based QE solver usually outputs very lengthy CAFs.
- For the application of automatic loop parallelization, too many case distinctions might increase the arithmetic cost of evaluation.

Thus, simplification of CAFs is needed!

Related work

- Simplification of Tarski formulas (A. Dolzmann & T. Sturm, 1995; H. Hong 1992; C. Brown & A. Strzeboński, 2010; C. Brown 2012; H. Iwane & H. Higuchi & H. Anai, 2013).
- Simplification of extended Tarski formulas (Chapter 8 of C. Brown's PhD thesis, Mathematica).
- Our goal is to reduce as much as possible the number of conjunctive CAF clauses while still maintaining the feature of case distinctions.

Generalized cylindrical algebraic formula (GCAF)

A GCAF is a "combination" of "nearby" CAFs.

GCAF

Let S be a set of disjoint cells in a CAD of $\mathbb{R}^n$. A GCAF associated with S, denoted by Φ, is of the form $\Phi = \vee_{i=1}^{s} \wedge_{j=1}^{s_i} \phi_{i,j}$ such that

- Each $\phi_{i,j}$ is of the form $w\ \sigma\ \mathrm{Root}_{w,k}(f)$, where $\sigma \in \{=, \neq, >, <, \geq, \leq\}$ and $w \in \{x_1, \ldots, x_n\}$.
- Let $v(\phi_{i,j})$ be the biggest variable appearing in $\phi_{i,j}$. Then for any ϕ_{i,j_1} and ϕ_{i,j_2}, where $j_1 < j_2$, we have $v(\phi_{i,j_1}) \leq v(\phi_{i,j_2})$.
- Let $w \in \{x_1, \ldots, x_n\}$. Denote by $\Phi_i^{<w} := \wedge_{v(\phi_{i,j})<w} \phi_{i,j}$. If $\phi_{i,j} = w\ \sigma\ \mathrm{Root}_{w,k}(f)$, then $\mathrm{Root}_{w,k}(f(\alpha))$ is defined for all α satisfying $\Phi_i^{<w}$.
- For every $w \in \{x_1, \ldots, x_n\}$, we have $\pi_{<w} Z_{\mathbb{R}}(\Phi_i^{\leq w}) = Z_{\mathbb{R}}(\Phi_i^{<w})$.
- The zero set of $\Phi_i := \wedge_{j=1}^{s_i} \phi_{i,j}$ is a union of some cells in S.
- The zero sets of Φ_i and Φ_j are disjoint for $1 \leq i < j \leq s$.
- The zero set of Φ is exactly $\cup_{c \in S} c$.

Example

Example

A CAF associated with the closed unit disk $x^2 + y^2 \leq 1$ is as below.

$$\begin{array}{rcl}(x = -1 \wedge y = 0) & \vee & (-1 < x \wedge x < 1 \wedge y = -\sqrt{1 - x^2}) \\ & \vee & (-1 < x \wedge x < 1 \wedge -\sqrt{1 - x^2} < y \wedge y < \sqrt{1 - x^2}) \\ & \vee & (-1 < x \wedge x < 1 \wedge y = \sqrt{1 - x^2}) \\ & \vee & (x = 1 \wedge y = 0)\end{array}$$

Example

Both $(-1 \leq x \wedge x \leq 1 \wedge -\sqrt{1 - x^2} \leq y \wedge y \leq \sqrt{1 - x^2})$ and

$$\begin{array}{rcl}(x = -1 \wedge y = 0) & \vee & (-1 < x \wedge x < 1 \wedge -\sqrt{1 - x^2} \leq y \wedge y \leq \sqrt{1 - x^2}) \\ & \vee & (x = 1 \wedge y = 0)\end{array}$$

are GCAFs equivalent to the CAF.

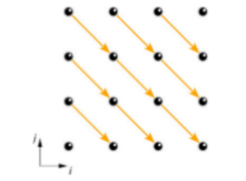
The main features of the simplification procedure

- Transform a CAF into a GCAF with less conjunctive clauses.
- Make use of the CAD data structure when applying the simplification.
- The procedure has four simplification levels.
- The first two levels merge adjacent or nearby CAD cells.
- The last two levels attempt to simplify a CAF into a single conjunctive clause, which is usually expected in the application of loop transformation.
- The first two levels are effective for general QE problems.
- The last two are effective for QE problems arising from loop transformation.

Automatic parallelization of polynomial multiplication

Serial dense univariate polynomial multiplication

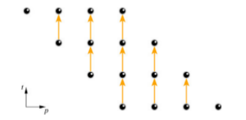
```
for(i=0; i<= 2*n; i++) c[i] = 0;
for(i=0; i<=n; i++) {
  for(j=0; j<=n; j++)
    c[i+j] += a[i] * b[j];
}
```



Dependence analysis suggests to set $t(i,j) = n - j$ and $p(i,j) = i + j$.

Synchronous parallel dense univariate polynomial multiplication

```
for (p=0; p<=2*n; p++) c[p]=0;
for (t=0; t<=n; t++)
  parallel_for (p=n-t; p<=2*n-t; p++)
    c[p] += a[t+p-n] * b[n-t];
}
```



A. Größlinger et al. Quantifier elimination in automatic loop parallelization. *J. Symb. Comput.*, 41(11):1206–1221, 2006.

The first simplification level (I)

```
ff := &E([i,j]), (0 <= i) &and (i <= n) &and (0 <= j) &and
                  (j <= n) &and (t = n - j) &and  (p = i + j);
R :=  PolynomialRing([i,j,p,t,n]);
sols := QuantifierElimination(ff, R, output=rootof, simplification=false);
`&or`(`&and`(n = 0,t = n,p = 0),
      `&and`(0 < n,t = 0,p = n),
      `&and`(0 < n,t = 0,n < p,p < 2*n),
      `&and`(0 < n,t = 0,p = 2*n),
      `&and`(0 < n,0 < t,t < n,p = -t+n),
      `&and`(0 < n,0 < t,t < n,-t+n < p,p < 2*n-t),
      `&and`(0 < n,0 < t,t < n,p = 2*n-t),
      `&and`(0 < n,t = n,p = 0),
      `&and`(0 < n,t = n,0 < p,p < n),
      `&and`(0 < n,t = n,p = n))
```

Observation 1: adjacent cells can be merged

Consider the following subformula:

$$\begin{array}{rcl}(0 < n \wedge t = 0 \wedge p = n) & \vee & (0 < n \wedge t = 0 \wedge n < p \wedge p < 2n) \\ & \vee & (0 < n \wedge t = 0 \wedge p = 2n)\end{array}.$$

It can be simplified to: $0 < n \wedge t = 0 \wedge n \leq p \wedge p \leq 2n$.

The first level (II)

Simplified result using Observation 1.

```
`&or`(`&and`(n = 0,t = n,p = 0),
       `&and`(0 < n,t = 0,n <= p,p <= 2*n),
       `&and`(0 < n,0 < t,t < n,-t+n <= p,p <= 2*n-t),
       `&and`(0 < n,t = n,0 <= p,p <= n)
```

Observation 2: specialization

If we specialize

$$-t + n \leq p \wedge p \leq 2n - t$$

at $t = 0$, we obtain

$$n \leq p \wedge p \leq 2n.$$

Similarly, at $t = n$, we obtain $0 \leq p \wedge p \leq n$.

Thus, the last three conjunctive clauses can be combined into one:

```
`&and`(0 < n,0 <= t,t <= n,-t+n <= p,p <= 2*n-t).
```

Applying this *specialization technique* again, we obtain the final output:

```
`&and`(0 <= n,0 <= t,t <= n,-t+n <= p,p <= 2*n-t).
```

The second simplification level

Main idea

The formula $n > 0 \wedge p < n$ and $n > 0 \wedge p > n$ can be combined as $n > 0 \wedge p \neq n$, even though the underlying CAD cells are not adjacent.

A program termination analysis example. The non-simplified one has 246 conjunctive clauses.

```
R := PolynomialRing([v1,v2,v3,labda,a11,a21,a22,a33,b12,b22,b23]);
f:= &E([v1,v2,v3,labda]), (labda>0) &and (a11*v1=labda*v1) &and
(a21*v1+a22*v2=labda*v2)&and(a33*v3=labda*v3)&and(b12*v2>0)&and(b22*v2+b23*v3>0);
QuantifierElimination(f, R, output=rootof,partial=true,simplification='L2');
`&or`(`&and`(b23 <> 0, b22 < 0, b12 < 0, a22 <= 0, a21 <> 0, 0 < a11),
`&and`(b23 <> 0, b22 < 0, b12 < 0, 0 < a22),
`&and`(b23 <> 0, b22 < 0, 0 < b12, 0 < a33, a22 <> a33, a21 <> 0, a11 = a33),
`&and`(b23 <> 0, b22 < 0, 0 < b12, 0 < a33, a22 = a33),
`&and`(b23 <> 0, b22 = 0, b12 <> 0, 0 < a33, a22 <> a33, a21 <> 0, a11 = a33),
`&and`(b23 <> 0, b22 = 0, b12 <> 0, 0 < a33, a22 = a33),
`&and`(b23 <> 0, 0 < b22, b12 < 0, 0 < a33, a22 <> a33, a21 <> 0, a11 = a33),
`&and`(b23 <> 0, 0 < b22, b12 < 0, 0 < a33, a22 = a33),
`&and`(b23 <> 0, 0 < b22, 0 < b12, a22 <= 0, a21 <> 0, 0 < a11),
`&and`(b23 <> 0, 0 < b22, 0 < b12, 0 < a22),
`&and`(b23 = 0, b22 < 0, b12 < 0, a22 <= 0, a21 <> 0, 0 < a11),
`&and`(b23 = 0, b22 < 0, b12 < 0, 0 < a22),
`&and`(b23 = 0, 0 < b22, 0 < b12, a22 <= 0, a21 <> 0, 0 < a11),
`&and`(b23 = 0, 0 < b22, 0 < b12, 0 < a22))
```

The third simplification level

```
ff := &E([i,j]), (0 <= i) &and (i <= n) &and (0 <= j) &and (j <= n) &and
                 (t = n - j) &and  (p = i + j);
R :=  PolynomialRing([i,j,t,p,n]);
sols := QuantifierElimination(ff, R, output=rootof, simplification=false);
`&or`(`&and`(n = 0,p = 0,t = n), `&and`(0 < n,p = 0,t = n),
      `&and`(0 < n,0 < p,p < n,t = -p+n),
      `&and`(0 < n,0 < p,p < n,-p+n < t,t < n),`&and`(0 < n,0 < p,p < n,t = n),
      `&and`(0 < n,p = n,t = 0),`&and`(0 < n,p = n,0 < t,t < n),
      `&and`(0 < n,p = n,t = n),`&and`(0 < n,n < p,p < 2*n,t = 0),
      `&and`(0 < n,n < p,p < 2*n,0 < t,t < -p+2*n),
      `&and`(0 < n,n < p,p < 2*n,t = -p+2*n),`&and`(0 < n,p = 2*n,t = 0))
```

Result using simplification level 1 or 2

```
`&or`(`&and`(n = 0,p = 0,t = 0),
      `&and`(0 < n,0 <= p,p <= n,-p+n <= t,t <= n),
      `&and`(0 < n,n < p,p <= 2*n,0 <= t,t <= -p+2*n))
```

The third simplification level

The question is how to merge the following two conjunctive clauses

$$0 < n \wedge 0 \leq p \wedge p \leq n \wedge -p + n \leq t \wedge t \leq n \tag{1}$$

and

$$0 < n \wedge n < p \wedge p \leq 2n \wedge 0 \leq t \wedge t \leq -p + 2n. \tag{2}$$

Observation

- The blue parts can be merged if the red parts are the same.
- ✗ If we force the red part to be equivalent, then $p = n$ must hold.
- ✓ Instead, to merge $A_1 \wedge B_1$ and $A_2 \wedge B_2$, we check if $A_1 \wedge B_1 \implies B_2$ and $A_2 \wedge B_2 \implies B_1$. If both are true, then we have

$$(A_1 \wedge B_1) \vee (A_2 \wedge B_2) \iff (A_1 \vee A_2) \wedge B_1 \wedge B_2.$$

By the above observation, the two clauses (**??**) and (**??**) are combined into

$$0 < n \wedge 0 \leq p \wedge p \leq 2n \wedge -p + n \leq t \wedge t \leq n \wedge 0 \leq t \wedge t \leq -p + 2n.$$

The fourth simplification level

```
R := PolynomialRing([i, j, t, p, u, b, B, n]);
ff := &E([i,j]), (0 < n) &and (0 <= i) &and (i <= n) &and (0 <= j) &and
                 (j <= n) &and  (t = n - j) &and  (p = i + j) &and
                 (b>=0) &and (0<=u) &and (u<B) &and (p=b*B+u);
```

Without simplification, there are 223 conjunctive clauses. With simplification level 1 or 2, there are 29 conjunctive clauses. Using level 3, the output consists of 5 conjunctive clauses.

The following are the second and the third one

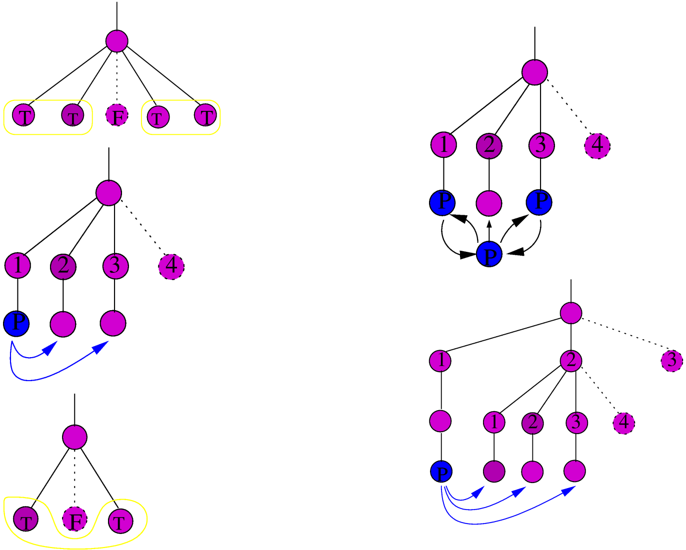
$$0 < n \wedge B = 2n \wedge b = 0 \wedge 0 \leq u \wedge u < 2n \wedge p = u \wedge n - u \leq t \wedge t \leq n \wedge 0 \leq t \wedge t \leq 2n - u)$$

$$0 < n \wedge B = 2n \wedge 0 < b \wedge b < 1/2 \wedge 0 \leq u \wedge u \leq -2bn + 2n \wedge p = 2bn + u$$
$$\wedge -2bn + n - u \leq t \wedge t \leq n \wedge 0 \leq t \wedge t \leq -2bn + 2n - u$$

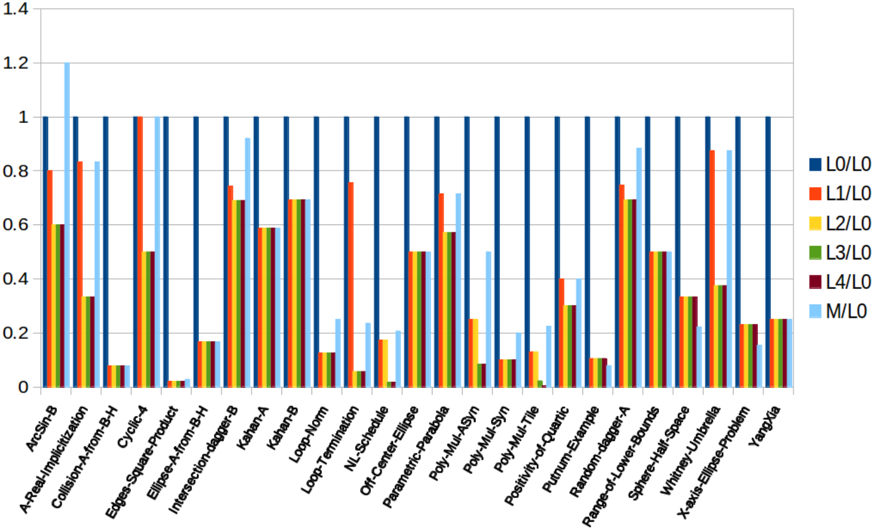
Note that direct specialization cannot merge the two together. A better pivot subformula simplifies the output as

```
`&and`(0 < n,0 < B,0 <= b,b <= 2*n/B,0 <= u,u < B,u <= -B*b+2*n,
       p = B*b+u, -B*b+n-u <= t, t <= n, 0 <= t, t <= -B*b+2*n-u)
```

The fourth simplification level



Benchmark: effect of simplification



A tiled parallel polynomial multiplication

```
for(i=0; i<=n; i++) {
  for(j=0; j<=n; j++)
    c[i+j] += a[i] * b[j];
}
```

Quantifier elimination with simplification level 4

```
R := PolynomialRing([i, j, t, p, u, b, B, n]);
ff := &E([i,j]), (0 < n) &and (0 <= i) &and (i <= n) &and (0 <= j) &and
                 (j <= n) &and  (t = n - j) &and  (p = i + j) &and
                 (b>=0) &and (0<=u) &and (u<B) &and (p=b*B+u);
QuantifierElimination(ff,R,partial=true,output=rootof,simplification='L4');
`&and`(0 < n,0 < B,0 <= b,b <= 2*n/B,0 <= u,u < B,u <= -B*b+2*n,
       p = B*b+u, -B*b+n-u <= t, t <= n, 0 <= t, t <= -B*b+2*n-u)
```

The parallel code

```
parallel_for (b=0; b<= 2 n / B; b++) {
    for (u=0; u<=min(B-1, 2*n - B * b); u++) {
       p = b * B + u;
       for (t=max(0,n-p); t<=min(n,2*n-p) ;t++)
           c[p] = c[p] + a[t+p-n] * b[n-t];
   }
}
```

```
b = blockIdx.x; u = threadIdx.x;
if (u <= 2*n - B * b) {
    p = b * B + u;
    for (t=max(0,n-p); t<=min(n,2*n-p) ;t++)
        c[p] = c[p] + a[t+p-n] * b[n-t];
}
```

More application examples on generating parametric CUDA kernels

The following data are obtained by calling QuantifierElimination of the RegularChains library with the following option

```
precondition='AP', output='rootof', simplification='L4'.
```

Here the first options enables the function to call a special QE procedure (preprocessing the input by FM before calling CAD).

Example	#constraints	#variables	Timing
Array reversal	6	5	0.072
1D Jacobi	6	5	0.948
2D Jacobi	14	9	7.735
LU decomposition	16	10	4.416
matrix transposition	14	9	1.314
matrix addition	14	9	1.314
matrix vector multiplication	6	5	0.072
matrix matrix multiplication	21	13	2.849

Table: Timings of quantifier elimination

Conclusion

- Introduce the notion of generalized cylindrical algebraic formula (GCAF).
- Propose a multi-level heuristic algorithm for simplifying CAFs into GCAFs.
- Make use of the CAD data structure when applying the simplification.
- The method has been implemented and new options are added to both the CylindricalAlgebraicDecompose and QuantifierElimination commands of the `RegularChains` library.
- The effectiveness of this algorithm is illustrated by examples coming from the application of automatic loop transformation as well as from other application domains.
- The running time overhead of simplification compared to the running time of the quantifier elimination procedure itself is negligible in the first two levels and acceptable in the advanced levels of the proposed heuristics.