Non-linear Interpolant Generation and Its Applications to Program Verification

Liyun Dai[1,4] Ting Gan [1] Bow-Yaw Wang [2] Bican Xia[1]
Naijun Zhan [3] Hengjun Zhao [3]

[1] LMAM & School of Mathematical Sciences, Peking University

[2] Institute of Information Science, Academia Sinica

[3] State Key Laboratory of Computer Science, Institute of Software, CAS

[4] Beijing International Center for Mathematical Research, Peking University

IPrA, July 13-14, 2013

Part I: Non-linear Interpolant Generation

(Based on CAV paper, the detail will be talked during the conference)

Motivation

- Current program verification techniques suffer from scalability.
- Compositional way has been thought as an effective solution to the problem.
- Interpolation-based techniques are inherently local and modular, which can be used to scale up these techniques of program verification:
  - Theorem proving: Nelson-Oppen method, SMT
  - Model-checking: BMC, CEGAR
  - Abstraction interpretation
  - Machine learning based approaches
- Synthesizing Craig interpolants is the cornerstone of interpolation based techniques

Interpolants

Given two formulae ϕ and ψ of $\mathcal{T}$ with $\vdash_{\mathcal{T}} (\phi \wedge \psi) \Rightarrow \bot$, then we say a formula Θ is an interpolant of ϕ and ψ, if $\vdash_{\mathcal{T}} \phi \Rightarrow \Theta$, $\vdash_{\mathcal{T}} (\psi \wedge \Theta) \Rightarrow \bot$, and Θ contains only symbols that ϕ and ψ share.

Semi-algebraic system

A semi-algebraic system (SAS) $\mathcal{T}(\mathbf{x})$ is of the form $\bigwedge_{j=0}^{k} f_j(\mathbf{x}) \rhd_j 0$, where f_j are polynomials in $\mathbb{R}[\mathbf{x}]$ and $\rhd_j \in \{=, \neq, \geq\}$.

Problem description

Given $\mathcal{T}_1 \wedge \mathcal{T}_2 \models \bot$, where

$$\mathcal{T}_1 = \begin{cases} f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, \ldots, g_{t_1}(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, \ldots, h_{u_1}(\mathbf{x}) = 0 \end{cases} \qquad \mathcal{T}_2 = \begin{cases} f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0, \\ g_{t_1+1}(\mathbf{x}) \neq 0, \ldots, g_t(\mathbf{x}) \neq 0, \\ h_{u_1+l}(\mathbf{x}) = 0, \ldots, h_u(\mathbf{x}) = 0 \end{cases}$$

we consider how to synthesize another polynomial formula I such that $\mathcal{T}_1 \vdash I$ and $\mathcal{T}_2 \wedge I \vdash \bot$.

Interpolants

Given two formulae ϕ and ψ of $\mathcal{T}$ with $\vdash_{\mathcal{T}} (\phi \wedge \psi) \Rightarrow \bot$, then we say a formula Θ is an interpolant of ϕ and ψ, if $\vdash_{\mathcal{T}} \phi \Rightarrow \Theta$, $\vdash_{\mathcal{T}} (\psi \wedge \Theta) \Rightarrow \bot$, and Θ contains only symbols that ϕ and ψ share.

Semi-algebraic system

A semi-algebraic system (SAS) $\mathcal{T}(\mathbf{x})$ is of the form $\bigwedge_{j=0}^{k} f_j(\mathbf{x}) \rhd_j 0$, where f_j are polynomials in $\mathbb{R}[\mathbf{x}]$ and $\rhd_j \in \{=, \neq, \geq\}$.

Problem description

Given $\mathcal{T}_1 \wedge \mathcal{T}_2 \models \bot$, where

$$\mathcal{T}_1 = \begin{cases} f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, \ldots, g_{t_1}(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, \ldots, h_{u_1}(\mathbf{x}) = 0 \end{cases} \qquad \mathcal{T}_2 = \begin{cases} f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0, \\ g_{t_1+1}(\mathbf{x}) \neq 0, \ldots, g_t(\mathbf{x}) \neq 0, \\ h_{u_1+1}(\mathbf{x}) = 0, \ldots, h_u(\mathbf{x}) = 0 \end{cases}$$

we consider how to synthesize another polynomial formula I such that $\mathcal{T}_1 \vdash I$ and $\mathcal{T}_2 \wedge I \vdash \bot$.

Interpolants

Given two formulae ϕ and ψ of $\mathcal{T}$ with $\vdash_{\mathcal{T}} (\phi \wedge \psi) \Rightarrow \bot$, then we say a formula Θ is an interpolant of ϕ and ψ, if $\vdash_{\mathcal{T}} \phi \Rightarrow \Theta$, $\vdash_{\mathcal{T}} (\psi \wedge \Theta) \Rightarrow \bot$, and Θ contains only symbols that ϕ and ψ share.

Semi-algebraic system

A semi-algebraic system (SAS) $\mathcal{T}(\mathbf{x})$ is of the form $\bigwedge_{j=0}^{k} f_j(\mathbf{x}) \triangleright_j 0$, where f_j are polynomials in $\mathbb{R}[\mathbf{x}]$ and $\triangleright_j \in \{=, \neq, \geq\}$.

Problem description

Given $\mathcal{T}_1 \wedge \mathcal{T}_2 \models \bot$, where

$$\mathcal{T}_1 = \begin{cases} f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, \ldots, g_{t_1}(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, \ldots, h_{u_1}(\mathbf{x}) = 0 \end{cases} \qquad \mathcal{T}_2 = \begin{cases} f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0, \\ g_{t_1+1}(\mathbf{x}) \neq 0, \ldots, g_t(\mathbf{x}) \neq 0, \\ h_{u_1+1}(\mathbf{x}) = 0, \ldots, h_u(\mathbf{x}) = 0 \end{cases}$$

we consider how to synthesize another polynomial formula I such that $\mathcal{T}_1 \vdash I$ and $\mathcal{T}_2 \wedge I \vdash \bot$.

Interpolants

Given two formulae ϕ and ψ of $\mathcal{T}$ with $\vdash_{\mathcal{T}} (\phi \wedge \psi) \Rightarrow \bot$, then we say a formula Θ is an interpolant of ϕ and ψ, if $\vdash_{\mathcal{T}} \phi \Rightarrow \Theta$, $\vdash_{\mathcal{T}} (\psi \wedge \Theta) \Rightarrow \bot$, and Θ contains only symbols that ϕ and ψ share.

Semi-algebraic system

A semi-algebraic system (SAS) $\mathcal{T}(\mathbf{x})$ is of the form $\bigwedge_{j=0}^{k} f_j(\mathbf{x}) \triangleright_j 0$, where f_j are polynomials in $\mathbb{R}[\mathbf{x}]$ and $\triangleright_j \in \{=, \neq, \geq\}$.

Problem description

Given $\mathcal{T}_1 \wedge \mathcal{T}_2 \models \bot$, where

$$\mathcal{T}_1 = \begin{cases} f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, \ldots, g_{t_1}(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, \ldots, h_{u_1}(\mathbf{x}) = 0 \end{cases} \qquad \mathcal{T}_2 = \begin{cases} f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0, \\ g_{t_1+1}(\mathbf{x}) \neq 0, \ldots, g_t(\mathbf{x}) \neq 0, \\ h_{u_1+l}(\mathbf{x}) = 0, \ldots, h_u(\mathbf{x}) = 0 \end{cases}$$

we consider how to synthesize another polynomial formula I such that $\mathcal{T}_1 \vdash I$ and $\mathcal{T}_2 \wedge I \vdash \bot$.

A running example

```
1  if (x*x + y*y < 1)
2  {    /* initial values */
3     while (x*x + y*y < 3)
4     {    x := x*x + y - 1;
5          y := y + x*y + 1;
6          if (x*x - 2*y*y - 4 > 0)
7             /* unsafe area */
8             error();    } }
```

$g_1 = 1 - x^2 - y^2 > 0$

$g_2 = 3 - x^2 - y^2 > 0$

$f_1 = x^2 + y - 1 - x' = 0$

$f_2 = y + x'y + 1 - y' = 0$

$g_3 = x'^2 - 2y'^2 - 4 > 0$

- The property to be verified is that **error()** procedure will never be executed.
- Suppose there is an execution segment $1 \to 3 \to 4 \to 5 \to 6 \to 8$.
- Let $\phi \triangleq g_1 > 0 \wedge f_1 = 0 \wedge f_2 = 0$ and $\psi \triangleq g_3 > 0$.
- The execution segment is infeasible iff $\phi \wedge \psi$ is unsatisfiable.

A running example

```
1  if (x*x + y*y < 1)
2  {    /* initial values */
3     while (x*x + y*y < 3)
4     {    x := x*x + y - 1;
5          y := y + x*y + 1;
6          if (x*x - 2*y*y - 4 > 0)
7              /* unsafe area */
8              error();    } }
```

$g_1 = 1 - x^2 - y^2 > 0$

$g_2 = 3 - x^2 - y^2 > 0$

$f_1 = x^2 + y - 1 - x' = 0$

$f_2 = y + x'y + 1 - y' = 0$

$g_3 = x'^2 - 2y'^2 - 4 > 0$

- The property to be verified is that **error()** procedure will never be executed.
- Suppose there is an execution segment $1 \to 3 \to 4 \to 5 \to 6 \to 8$.
- Let $\phi \triangleq g_1 > 0 \wedge f_1 = 0 \wedge f_2 = 0$ and $\psi \triangleq g_3 > 0$.
- The execution segment is infeasible iff $\phi \wedge \psi$ is unsatisfiable.

A running example

```
1  if (x*x + y*y < 1)
2  {    /* initial values */
3     while (x*x + y*y < 3)
4     {    x := x*x + y - 1;
5          y := y + x*y + 1;
6          if (x*x - 2*y*y - 4 > 0)
7             /* unsafe area */
8             error();    } }
```

$g_1 = 1 - x^2 - y^2 > 0$

$g_2 = 3 - x^2 - y^2 > 0$

$f_1 = x^2 + y - 1 - x' = 0$

$f_2 = y + x'y + 1 - y' = 0$

$g_3 = x'^2 - 2y'^2 - 4 > 0$

- The property to be verified is that **error()** procedure will never be executed.
- Suppose there is an execution segment $1 \to 3 \to 4 \to 5 \to 6 \to 8$.
- Let $\phi \triangleq g_1 > 0 \wedge f_1 = 0 \wedge f_2 = 0$ and $\psi \triangleq g_3 > 0$.
- The execution segment is infeasible iff $\phi \wedge \psi$ is unsatisfiable.

Sketch of our approach

Firstly, reduction by Positivestellensatz

If there exist $\delta_1, \ldots, \delta_7$ s.t.

$$g_1\delta_2 + f_1(\delta_3 - \delta_4) + f_2(\delta_5 - \delta_6) + g_3\delta_7 + \delta_1 + 1 \equiv 0,$$

where $\delta_1, \ldots, \delta_6 \in \mathbb{R}[x, y, x', y'], \delta_7 \in \mathbb{R}[x', y']$ are sums of squares, i.e., of the form $f_1^2 + \ldots + f_n^2$, then $g_3\delta_7 + \frac{1}{2} \leq 0$ is an interpolant for ϕ and ψ.

Then finding $\delta_1, \ldots, \delta_7$ by SDP

- Set $\deg(\delta_1) = \deg(\delta_2) = \cdots = \deg(\delta_7) = 4$. Then $\delta_1 = (Z_2^4)^T Q_1 (Z_2^4)$, $\delta_2 = (Z_2^4)^T Q_2 (Z_2^4), \ldots, \delta_7 = (Z_2^4)^T Q_7 (Z_2^4)$, where
  - $Q_1, \ldots, Q_7$ are 15×15-symmetric matrices, and all entries of $Q_1, \ldots, Q_7$ are parameters.
  - $Z_2^4 = \left[1, x, y, x', y', xy, xx', xy', x'y, x'y', yy', x^2, y^2, x'^2, y'^2\right]$.

- $g_1\delta_2+f_1(\delta_3-\delta_4)+f_2(\delta_5-\delta_6)+g_3\delta_7+\delta_1+1=\sum_{i+j+k+m\leq 4} c^{i,j,k,m}x^i y^j x'^k y'^m$,

  where $c_{i,j,k,m}=\langle A_{i,j,k,m}, X\rangle$, $X=diag\{Q_1,Q_2,\ldots,Q_7\}$, entries of $A_{i,j,k,m}$ comes from coefficients of g_1, f_1, f_2, g_3, e.g.,

$$A_{0,0,0,0}=\begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & -1 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots\cdots & -4 \end{pmatrix}$$

- Resulted SDP: $\inf_{X\in Sym_{105}} \langle C, X\rangle$ s.t. $X\succeq 0, \langle A_{i,j,k,m}, X\rangle = 0$ $(i,j,k,m=1,\ldots,4)$.
- Solving the SDP, obtain $\delta_1,\ldots\delta_7$ with tool `AiSat`. Thus, $g_3\delta_7+\frac{1}{2}\geq 0$ is an interpolant.
- In addition, we can verify that it is an inductive invariant by QE.

- $g_1\delta_2+f_1(\delta_3-\delta_4)+f_2(\delta_5-\delta_6)+g_3\delta_7+\delta_1+1=\sum_{i+j+k+m\leq 4} c^{i,j,k,m}x^i y^j x'^k y'^m$,

  where $c_{i,j,k,m}=\langle A_{i,j,k,m}, X\rangle$, $X=diag\{Q_1,Q_2,\ldots,Q_7\}$, entries of $A_{i,j,k,m}$ comes from coefficients of g_1, f_1, f_2, g_3, e.g.,

$$A_{0,0,0,0}=\begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & -1 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots\cdots & -4 \end{pmatrix}$$

- Resulted SDP: $\inf_{X\in Sym_{105}} \langle C, X\rangle$ s.t. $X\succeq 0, \langle A_{i,j,k,m}, X\rangle = 0$ $(i,j,k,m=1,\ldots,4)$.
- Solving the SDP, obtain $\delta_1,\ldots\delta_7$ with tool `AiSat`. Thus, $g_3\delta_7+\frac{1}{2}\geq 0$ is an interpolant.
- In addition, we can verify that it is an inductive invariant by QE.

- $g_1\delta_2+f_1(\delta_3-\delta_4)+f_2(\delta_5-\delta_6)+g_3\delta_7+\delta_1+1=\sum_{i+j+k+m\leq 4} c^{i,j,k,m}x^i y^j x'^k y'^m$,

  where $c_{i,j,k,m}=\langle A_{i,j,k,m}, X\rangle$, $X=diag\{Q_1,Q_2,\ldots,Q_7\}$, entries of $A_{i,j,k,m}$ comes from coefficients of g_1, f_1, f_2, g_3, e.g.,

$$A_{0,0,0,0}=\begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & -1 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots\cdots & -4 \end{pmatrix}$$

- Resulted SDP: $\inf_{X\in Sym_{105}} \langle C, X\rangle$ s.t. $X\succeq 0, \langle A_{i,j,k,m}, X\rangle = 0$ $(i,j,k,m=1,\ldots,4)$.
- Solving the SDP, obtain $\delta_1,\ldots\delta_7$ with tool `AiSat`. Thus, $g_3\delta_7+\frac{1}{2}\geq 0$ is an interpolant.
- In addition, we can verify that it is an inductive invariant by QE.

- $g_1\delta_2+f_1(\delta_3-\delta_4)+f_2(\delta_5-\delta_6)+g_3\delta_7+\delta_1+1=\sum_{i+j+k+m\leq 4} c^{i,j,k,m}x^i y^j x'^k y'^m$,

  where $c_{i,j,k,m}=\langle A_{i,j,k,m}, X\rangle$, $X=diag\{Q_1,Q_2,\ldots,Q_7\}$, entries of $A_{i,j,k,m}$ comes from coefficients of g_1, f_1, f_2, g_3, e.g.,

$$A_{0,0,0,0}=\begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & -1 & \cdots\cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots\cdots & -4 \end{pmatrix}$$

- Resulted SDP: $\inf_{X\in Sym_{105}} \langle C, X\rangle$ s.t. $X\succeq 0, \langle A_{i,j,k,m}, X\rangle = 0$ $(i,j,k,m=1,\ldots,4)$.
- Solving the SDP, obtain $\delta_1,\ldots\delta_7$ with tool `AiSat`. Thus, $g_3\delta_7+\frac{1}{2}\geq 0$ is an interpolant.
- In addition, we can verify that it is an inductive invariant by QE.

Part II: Applications to Program Verification

Invariant

Inductive invariant of loop

Given a Hoare triple $\{Pre\}$ **while** B **do** P $\{Post\}$, we say a formula θ is an invariant of the loop, if

- $\theta \Rightarrow Pre$
- $\{\theta \wedge B\}\ P\ \{\theta\}$
- $\theta \wedge \neg B \Rightarrow Post$

General framework

- To negate $Post$;
- For each single execution of P, generating an interpolant between Pre and $\neg Post$;
- Using QE to verify the disjunct of the obtained interpolants form an inductive invariant of the loop.

Invariant

Inductive invariant of loop

Given a Hoare triple $\{Pre\}$ **while** B **do** P $\{Post\}$, we say a formula θ is an invariant of the loop, if

- $\theta \Rightarrow Pre$
- $\{\theta \wedge B\}\ P\ \{\theta\}$
- $\theta \wedge \neg B \Rightarrow Post$

General framework

- To negate $Post$;
- For each single execution of P, generating an interpolant between Pre and $\neg Post$;
- Using QE to verify the disjunct of the obtained interpolants form an inductive invariant of the loop.

Invariant

Inductive invariant of loop

Given a Hoare triple $\{Pre\}$ **while** B **do** P $\{Post\}$, we say a formula θ is an invariant of the loop, if

- $\theta \Rightarrow Pre$
- $\{\theta \wedge B\}\ P\ \{\theta\}$
- $\theta \wedge \neg B \Rightarrow Post$

General framework

- To negate $Post$;
- For each single execution of P, generating an interpolant between Pre and $\neg Post$;
- Using QE to verify the disjunct of the obtained interpolants form an inductive invariant of the loop.

Example

Source code

```
1   vc := 0;
2     /* the initial veclocity  */
3   fr := 1000;
4     /* the initial force */
5   ac := 0.0005 * fr;
6     /* the initial acceleration */
7   while ( 1 )
8   {   fa := 0.5418 * vc * vc;
9         /* the force control  */
10    fr := 1000 − fa;
11    ac := 0.0005 * fr;
12    vc := vc + ac;
13     assert(vc < 49.61);
14        /* the safety velocity  */ }
```

- Safety property is that the velocity of the car cannot surpass $49.61m/s$.
- Suppose $(vc < 49.61) \rightarrow 8 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 13$ $(vc \geq 49.61)$.
- By applying `AiSat`, we can obtain an interpolant $-1.3983vc + 69.358 > 0$, which guarantees $vc < 49.61$.
- It is easy to verify $-1.3983vc + 69.358 > 0$ is an inductive invariant.

Comparison with other approaches

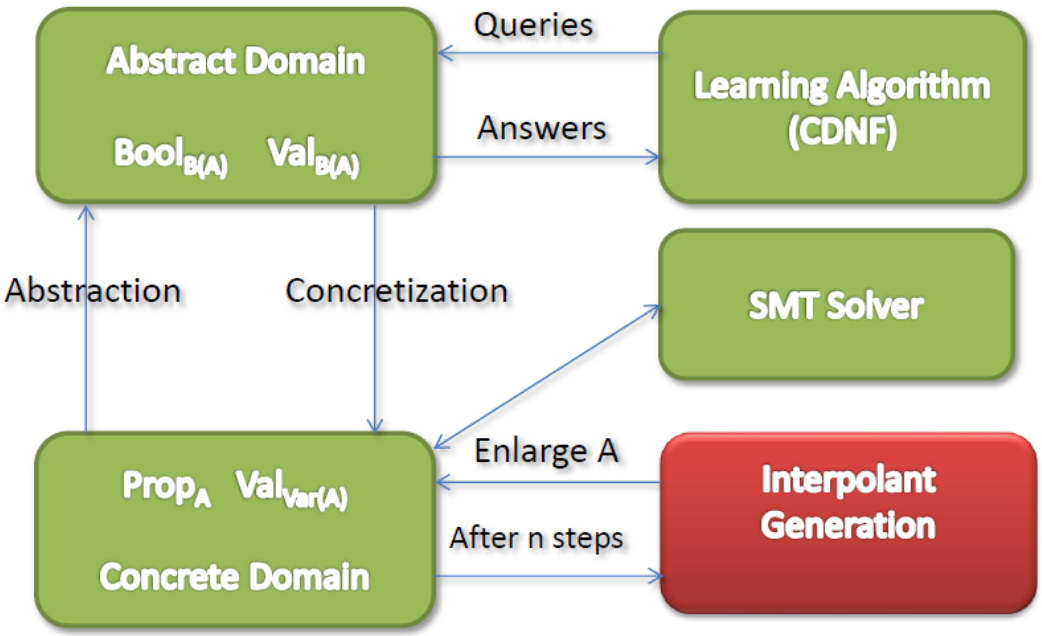
Interpolation based vs QE based

- Interpolation based is complete under *Archimedean condition*, while QE based is complete related to the predefined templates.
- Interpolation based is more efficient, its complexity is polynomial in the given degree, whose upper bound is at least triply exponential in the numbers of variables and constraints; while QE based is doubly exponential in the number of parameters and variables in the predefined templates in general.
- Interpolation based has to consider error issue because of numerical computation, while QE based does not need.

Combining interpolation generation with QE to invariant generation can improve efficiency, also makes error is controllable.

Combining with machine-learning

General framework

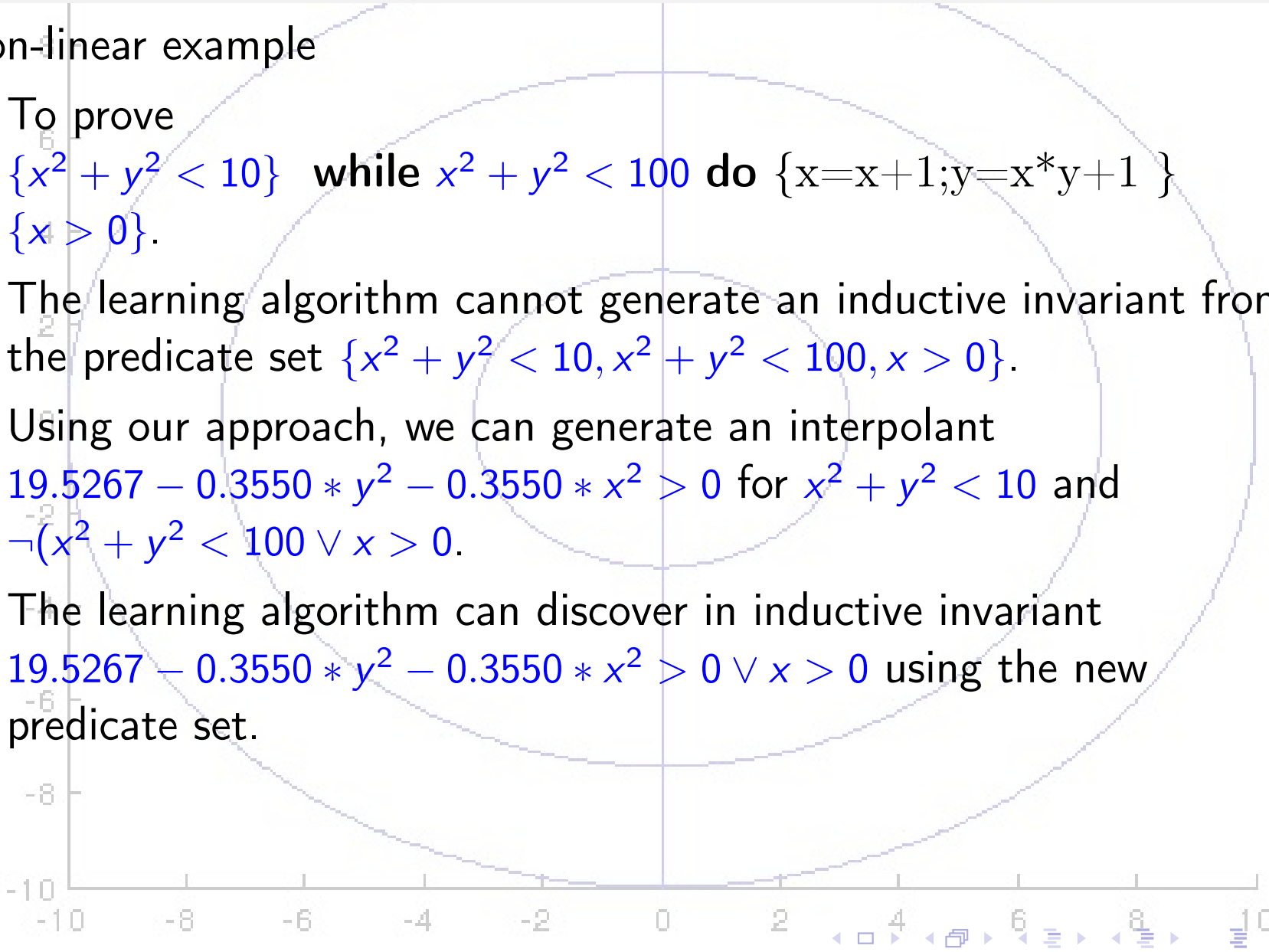


- Two parts: Learning + discovering predicate set.
- Learning=CDNF+predicate abstraction+SMT solver.
- Discovering predicate set is by interpolation.
- Learning part is incomplete: after n steps, if no invariant is synthesized, either **restart** or **enlarge** the predicate set by interpolation.
- Previous work is only applicable to linear cases.

Extending to non-linear cases

A non-linear example

- To prove
  $\{x^2 + y^2 < 10\}$ **while** $x^2 + y^2 < 100$ **do** {x=x+1;y=x*y+1 } $\{x > 0\}$.
- The learning algorithm cannot generate an inductive invariant from the predicate set $\{x^2 + y^2 < 10, x^2 + y^2 < 100, x > 0\}$.
- Using our approach, we can generate an interpolant $19.5267 - 0.3550 * y^2 - 0.3550 * x^2 > 0$ for $x^2 + y^2 < 10$ and $\neg(x^2 + y^2 < 100 \vee x > 0$.
- The learning algorithm can discover in inductive invariant $19.5267 - 0.3550 * y^2 - 0.3550 * x^2 > 0 \vee x > 0$ using the new predicate set.



Conclusion

- We first summarize our recent work on generating non-linear interpolants by semi-definite programming.
- Then we show how to apply the results to program verification, including invariant generation and combining with machine-learning based techniques.

Future work

- Some issues related to nonlinear interpolant generation, like
  - How to relax the Archimedean condition.
  - How to combine non-linear arithmetic with other well-established decidable first order theories.
  - To investigate errors caused by numerical computation in **SDP** is quite interesting.
- To investigate combining with other verification techniques, like CEGAR, BMC, SMT and so on.
- To investigate the possibility to the verification of hybrid systems.

Conclusion

- We first summarize our recent work on generating non-linear interpolants by semi-definite programming.
- Then we show how to apply the results to program verification, including invariant generation and combining with machine-learning based techniques.

Future work

- Some issues related to nonlinear interpolant generation, like
  - How to relax the Archimedean condition.
  - How to combine non-linear arithmetic with other well-established decidable first order theories.
  - To investigate errors caused by numerical computation in **SDP** is quite interesting.
- To investigate combining with other verification techniques, like CEGAR, BMC, SMT and so on.
- To investigate the possibility to the verification of hybrid systems.

Conclusion

- We first summarize our recent work on generating non-linear interpolants by semi-definite programming.
- Then we show how to apply the results to program verification, including invariant generation and combining with machine-learning based techniques.

Future work

- Some issues related to nonlinear interpolant generation, like
  - How to relax the Archimedean condition.
  - How to combine non-linear arithmetic with other well-established decidable first order theories.
  - To investigate errors caused by numerical computation in **SDP** is quite interesting.
- To investigate combining with other verification techniques, like CEGAR, BMC, SMT and so on.
- To investigate the possibility to the verification of hybrid systems.

References

1. L. Dai, B. Xia and N. Zhan: Generating non-linear interpolants by semi-definite programming. CAV 2013.
2. Y. Chen, B. Xia, L. Yang and N. Zhan: Generating polynomial invariants by DISCOVERER and QEPCAD. Formal Methods and Hybrid Real-time Systems.
3. Yungbum Jung, Wonchan Lee, Bow-Yaw Wang, Kwangkeun Yi: Predicate generation for learning-based quantifier-free loop invariant inference. TACAS 2011.
4. Soonho Kong, Yungbum Jung, Cristina David, Bow-Yaw Wang, Kwangkeun Yi: Automatically inferring quantified Loop Invariants by Algorithmic Learning from Simple Templates. APLAS 2010.

Thanks & Questions?