Non-linear Interpolant Generation and Its Applications to Program Verification

Liyun Dai\textsuperscript{1,4} Ting Gan \textsuperscript{1} Bow-Yaw Wang \textsuperscript{2} Bican Xia\textsuperscript{1}
Naijun Zhan \textsuperscript{3} Hengjun Zhao \textsuperscript{3}

\textsuperscript{1} LMAM & School of Mathematical Sciences, Peking University
\textsuperscript{2} Institute of Information Science, Academia Sinica
\textsuperscript{3} State Key Laboratory of Computer Science, Institute of Software, CAS
\textsuperscript{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 \( \phi \) and \( \psi \) of \( \mathcal{T} \) with \( \Rightarrow_\mathcal{T} (\phi \land \psi) \Rightarrow \bot \), then we say a formula \( \Theta \) is an interpolant of \( \phi \) and \( \psi \), if \( \Rightarrow_\mathcal{T} \phi \Rightarrow \Theta \), \( \Rightarrow_\mathcal{T} (\psi \land \Theta) \Rightarrow \bot \), and \( \Theta \) contains only symbols that \( \phi \) and \( \psi \) share.

Semi-algebraic system

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

Problem description

Given \( \mathcal{T}_1 \land \mathcal{T}_2 \models \bot \), where

\[
\begin{align*}
\mathcal{T}_1 &= \left\{ \begin{array}{l}
f_1(x) \geq 0, \ldots, f_{s_1}(x) \geq 0, \\
g_1(x) \neq 0, \ldots, g_{t_1}(x) \neq 0, \\
h_1(x) = 0, \ldots, h_{u_1}(x) = 0
\end{array} \right. \\
\mathcal{T}_2 &= \left\{ \begin{array}{l}
f_{s_1+1}(x) \geq 0, \ldots, f_s(x) \geq 0, \\
g_{t_1+1}(x) \neq 0, \ldots, g_t(x) \neq 0, \\
h_{u_1+1}(x) = 0, \ldots, h_u(x) = 0
\end{array} \right.
\end{align*}
\]

we consider how to synthesize another polynomial formula \( I \) such that \( \mathcal{T}_1 \vdash I \) and \( \mathcal{T}_2 \land I \vdash \bot \).
Interpolants

Given two formulae \( \phi \) and \( \psi \) of \( T \) with \( \vdash_T (\phi \land \psi) \Rightarrow \bot \), then we say a formula \( \Theta \) is an interpolant of \( \phi \) and \( \psi \), if \( \vdash_T \phi \Rightarrow \Theta \), \( \vdash_T (\psi \land \Theta) \Rightarrow \bot \), and \( \Theta \) contains only symbols that \( \phi \) and \( \psi \) share.

Semi-algebraic system

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

Problem description

Given \( T_1 \land T_2 \models \bot \), where

\[
T_1 = \begin{cases} 
  f_1(x) \geq 0, \ldots, f_{s_1}(x) \geq 0, \\
  g_1(x) \neq 0, \ldots, g_{t_1}(x) \neq 0, \\
  h_1(x) = 0, \ldots, h_{u_1}(x) = 0 
\end{cases} \quad T_2 = \begin{cases} 
  f_{s_1+1}(x) \geq 0, \ldots, f_s(x) \geq 0, \\
  g_{t_1+1}(x) \neq 0, \ldots, g_t(x) \neq 0, \\
  h_{u_1+l}(x) = 0, \ldots, h_u(x) = 0 
\end{cases}
\]

we consider how to synthesize another polynomial formula \( I \) such that \( T_1 \vdash I \) and \( T_2 \land I \vdash \bot \).
Problem Description

Interpolants

Given two formulae $\phi$ and $\psi$ of $T$ with $\vdash_T (\phi \land \psi) \Rightarrow \bot$, then we say a formula $\Theta$ is an interpolant of $\phi$ and $\psi$, if $\vdash_T \phi \Rightarrow \Theta$, $\vdash_T (\psi \land \Theta) \Rightarrow \bot$, and $\Theta$ contains only symbols that $\phi$ and $\psi$ share.

Semi-algebraic system

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

Problem description

Given $T_1 \land T_2 \models \bot$, where

$T_1 = \left\{ \begin{array}{l} f_1(x) \geq 0, \ldots, f_{s_1}(x) \geq 0, \\ g_1(x) \neq 0, \ldots, g_{t_1}(x) \neq 0, \\ h_1(x) = 0, \ldots, h_{u_1}(x) = 0 \end{array} \right\}$

$T_2 = \left\{ \begin{array}{l} f_{s_1+1}(x) \geq 0, \ldots, f_s(x) \geq 0, \\ g_{t_1+1}(x) \neq 0, \ldots, g_{t}(x) \neq 0, \\ h_{u_1+i}(x) = 0, \ldots, h_u(x) = 0 \end{array} \right\}$

we consider how to synthesize another polynomial formula $I$ such that $T_1 \vdash I$ and $T_2 \land I \vdash \bot$. 
Interpolants

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

Semi-algebraic system

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

Problem description

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

$$\mathcal{T}_1 = \left\{ \begin{array}{l} f_1(x) \geq 0, \ldots, f_{s_1}(x) \geq 0, \\ g_1(x) \neq 0, \ldots, g_{t_1}(x) \neq 0, \\ h_1(x) = 0, \ldots, h_{u_1}(x) = 0 \end{array} \right\} \quad \mathcal{T}_2 = \left\{ \begin{array}{l} f_{s_1+1}(x) \geq 0, \ldots, f_{s}(x) \geq 0, \\ g_{t_1+1}(x) \neq 0, \ldots, g_{t}(x) \neq 0, \\ h_{u_1+l}(x) = 0, \ldots, h_{u}(x) = 0 \end{array} \right\}$$

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

1. if \((x \times x + y \times y < 1)\)  
2. \{
3. \hspace{1em} /* initial values */
4. \hspace{1em} while \((x \times x + y \times y < 3)\)
5. \hspace{1em} \{
6. \hspace{2em} x := x \times x + y - 1;
7. \hspace{2em} y := y + x \times y + 1;
8. \hspace{1em} if \((x \times x - 2 \times y \times y - 4 > 0)\)
9. \hspace{1em} /* unsafe area */
10. \hspace{1em} error();
11. \}\}

\(g_1 = 1 - x^2 - y^2 > 0\)
\(g_2 = 3 - x^2 - y^2 > 0\)
\(f_1 = x^2 + y - 1 - x' = 0\)
\(f_2 = y + x'y + 1 - y' = 0\)
\(g_3 = x'^2 - 2y'^2 - 4 > 0\)

- The property to be verified is that \texttt{error()} procedure will never be executed.
- Suppose there is an execution segment \(1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8\).
- Let \(\phi \triangleq g_1 > 0 \land f_1 = 0 \land f_2 = 0\) and \(\psi \triangleq g_3 > 0\).
- The execution segment is infeasible iff \(\phi \land \psi\) is unsatisfiable.
A running example

1 if \((x \times x + y \times y < 1)\)  
2 \(\{\) /* initial values */  
3 \(\) while \((x \times x + y \times y < 3)\)  
4 \(\) \{  
5 \(x := x \times x + y - 1;\)  
6 \(y := y + x \times y + 1;\)  
7 \(\) if \((x \times x - 2 \times y \times y - 4 > 0)\)  
8 \(\) /* unsafe area */  
9 \(\) 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 \text{error()} procedure will never be executed.
- Suppose there is an execution segment 1 → 3 → 4 → 5 → 6 → 8.
- Let \(\phi \triangleq g_1 > 0 \land f_1 = 0 \land f_2 = 0\) and \(\psi \triangleq g_3 > 0\).
- The execution segment is infeasible iff \(\phi \land \psi\) is unsatisfiable.
A running example

1 if \((x \times x + y \times y < 1)\)  
\[
g_1 = 1 - x^2 - y^2 > 0
\]
2 { /* initial values */
\[
g_2 = 3 - x^2 - y^2 > 0
\]
3 while \((x \times x + y \times y < 3)\)
\[
f_1 = x^2 + y - 1 - x' = 0
\]
4 {  
\[
f_2 = y + x'y + 1 - y' = 0
\]
5 \(x := x \times x + y - 1;\)
\[
g_3 = x'^2 - 2y'^2 - 4 > 0
\]
6 \(y := y + x \times y + 1;\)
\[
\]
7 if \((x \times x - 2 \times y \times y - 4 > 0)\)
\[
/* unsafe area */
\]
8 error();  

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

Firstly, reduction by Positivstellensatz

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 $\phi$ and $\psi$. 
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 $15 \times 15$-symmetric matrices, and all entries of $Q_1, \ldots, Q_7$ are parameters.
  - $Z_2^4 = [1, x, y, x', y', xy, xx', xy', x'y, x'y', y'y, x^2, y^2, x'^2, y'^2]$.
\[
g_1 \delta_2 + f_1(\delta_3 - \delta_4) + f_2(\delta_5 - \delta_6) + g_3 \delta_7 + \delta_1 + 1 = \sum_{i+j+k+m \leq 4} c_{i,j,k,m} x^i y^j x'^k y'^m,
\]

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

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

Resulted SDP: \( \inf_{X \in \text{Sym}_{105}} \langle C, X \rangle \) 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 \( c_{i,j,k,m} = \langle A_{i,j,k,m}, X \rangle \), \( X = \text{diag}\{Q_1, Q_2, \ldots, Q_7\} \), entries of \( A_{i,j,k,m} \) comes from coefficients of \( g_1, f_1, f_2, g_3 \), e.g.,

\[
A_{0,0,0,0} = \begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & 1 & \cdots & 0 & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & 0 & \cdots & -1 & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & 0 & \cdots & 0 & \cdots & \cdots & \cdots & -4
\end{pmatrix}
\]

Resulted SDP: \( \inf_{X \in \text{Sym}_{105}} \langle C, X \rangle \) 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 \( c_{i,j,k,m} = \langle A_{i,j,k,m}, X \rangle \), \( X = \text{diag}\{Q_1, Q_2, \ldots, Q_7\} \), entries of \( A_{i,j,k,m} \) comes from coefficients of \( g_1, f_1, f_2, g_3 \), e.g.,

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

Resulted SDP:
\[
\inf_{X \in \text{Sym}_{105}} \langle C, X \rangle \text{ s.t. } X \succeq 0, \langle A_{i,j,k,m}, X \rangle = 0 \\
(i, j, k, m = 1, \ldots, 4).
\]

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 \( c_{i,j,k,m} = \langle A_{i,j,k,m}, X \rangle \), \( X = \text{diag}\{Q_1, Q_2, \ldots, Q_7\} \), entries of 
\( A_{i,j,k,m} \) comes from coefficients of \( g_1, f_1, f_2, g_3 \), e.g.,
\[
A_{0,0,0,0} = \begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 1 & \cdots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & -1 & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & \cdots & \cdots & \cdots & -4
\end{pmatrix}
\]

Resulted SDP: \( \inf_{X \in \text{Sym}_{105}} \langle C, X \rangle \text{ s.t. } X \succeq 0, \langle A_{i,j,k,m}, X \rangle = 0 \)

\((i, j, k, m = 1, \ldots, 4)\).

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 \{\textit{Pre}\} while \textit{B} do \textit{P} \{\textit{Post}\}, we say a formula \( \theta \) is an invariant of the loop, if

- \( \theta \Rightarrow \text{Pre} \)
- \( \{\theta \land B\} \ P \ \{\theta\} \)
- \( \theta \land \neg B \Rightarrow \text{Post} \)

General framework

- To negate \textit{Post};
- For each single execution of \textit{P}, generating an interpolant between \textit{Pre} and \textit{\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 \( \{\text{Pre}\} \text{ while } B \text{ do } P \{\text{Post}\} \), we say a formula \( \theta \) is an invariant of the loop, if

- \( \theta \Rightarrow \text{Pre} \)
- \( \{\theta \land B\} \ P \{\theta\} \)
- \( \theta \land \neg B \Rightarrow \text{Post} \)

General framework

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

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

\[
\begin{align*}
\theta & \Rightarrow Pre \\
\{\theta \land B\} & P \{\theta\} \\
\theta \land \neg B & \Rightarrow Post
\end{align*}
\]

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

```c
1 vc := 0;
2 /* the initial velocity */
3 fr := 1000;
4 /* the initial force */
5 ac := 0.0005 * fr;
6 /* the initial acceleration */
7 while ( 1 )
8 {
9     fa := 0.5418 * vc * vc;
10    /* the force control */
11     fr := 1000 - fa;
12     ac := 0.0005 * fr;
13     vc := vc + ac;
14    assert(vc < 49.61);
15    /* the safety velocity */
16 }
```

- Safety property is that the velocity of the car cannot surpass 49.61 m/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.

Diagrams:
- Abstract Domain: $\text{Bool}_{B(A)}$, $\text{Val}_{B(A)}$ to Queries, Answers, and Concrete Domain: $\text{Prop}_A$, $\text{Val}_{\text{Var}(A)}$ via Concretization.
- Learning Algorithm (CDNF) processes Queries, Answers, and Interpolant Generation.
- SMT Solver inputs Enlarge A and After n steps.

Extending to non-linear cases

A non-linear example

- To prove
  \[
  \{ x^2 + y^2 < 10 \} \text{ while } x^2 + y^2 < 100 \text{ do } \{ x = x + 1; y = x \cdot y + 1 \} \text{ } \{ x > 0 \}.
  \]

- 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
  \[
  19.5267 - 0.3550 \cdot y^2 - 0.3550 \cdot x^2 > 0 \text{ for } x^2 + y^2 < 10 \text{ and } \neg (x^2 + y^2 < 100 \lor x > 0).
  \]

- The learning algorithm can discover in inductive invariant
  \[
  19.5267 - 0.3550 \cdot y^2 - 0.3550 \cdot x^2 > 0 \lor x > 0 \text{ using the new predicate set.}
  \]
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


Thanks & Questions?