Automatic Generation of Polynomial Invariants for System Verification

Enric Rodríguez-Carbonell
Plan of the Talk

- Introduction
  - Need for program verification
  - Invariants and abstract interpretation
  - Polynomial invariants
Plan of the Talk

- Introduction
- Generation of Invariant Polynomial Equalities
  (with D. Kapur: ISSAC’04, SAS’04)
  - Related work
  - Abstract domain of ideals
  - Particular case: loops without nesting
Plan of the Talk

- Introduction
- Generation of Invariant Polynomial Equalities
- Applications of Polynomial Equality Invariants
  - Imperative programs
    (with D. Kapur: ICTAC’04)
  - Petri nets
    (with R. Clarisó, J. Cortadella: ATPN’05)
  - Hybrid systems
    (with A. Tiwari: HSCC’05)

Plan of the Talk

- Introduction
- Generation of Invariant Polynomial Equalities
- Applications of Polynomial Equality Invariants
- Generation of Invariant Polynomial Inequalities
  - (with R. Bagnara, E. Zaffanella: SAS’05)
    - Abstract domain of polynomial cones
Plan of the Talk

- Introduction
- Generation of Invariant Polynomial Equalities
- Applications of Polynomial Equality Invariants
- Generation of Invariant Polynomial Inequalities
- Conclusions and Future Work
Introduction

- Need for program verification
- Invariants and abstract interpretation
- Polynomial invariants

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Need for Software Verification

- Critical systems
  - safety
  - security
  - economy
Need for Software Verification

- Critical systems
  - safety
  - security
  - economy

Failure of the Ariane 5 launcher in 1996
Need for Software Verification

- Critical systems
  - safety
  - security
  - economy
- Fundamental finding errors asap. See Microsoft’s Software Productivity Tools group: verification pays off
Need for Software Verification

- Critical systems
  - safety
  - security
  - economy

- Fundamental finding errors asap. See Microsoft’s Software Productivity Tools group: verification pays off

- Invariants are crucial for program verification!
Introduction

- Need for program verification
- Invariants and abstract interpretation
- Polynomial invariants

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
CORRECTNESS OF THE SYSTEM:

\[ \text{SYSTEM STATES} \cap \text{BAD STATES} = \emptyset \]
CORRECTNESS OF THE SYSTEM:

\[ \text{SYSTEM STATES} \cap \text{BAD STATES} = \emptyset \]

SUFFICIENT CONDITION:

\[ \text{INVARIANT} \cap \text{BAD STATES} = \emptyset \]
Abstract interpretation allows computing invariants:

- Intervals (Cousot & Cousot 1976, Harrison 1977)
- Congruences (Granger 1991)
- Linear inequalities (Cousot & Halbwachs 1978, Colón & Sankaranarayanan & Sipma 2003)
Abstract interpretation allows computing invariants:

- intervals (Cousot & Cousot 1976, Harrison 1977)

\[ x \in [0, 1] \land y \in [0, \infty) \]
Abstract interpretation allows computing invariants:

- intervals (Cousot & Cousot 1976, Harrison 1977)
  \[ x \in [0, 1] \land y \in [0, \infty) \]

- congruences (Granger 1991)
  \[ x \equiv y \mod(2) \]
Abstract interpretation allows computing invariants:

- intervals (Cousot & Cousot 1976, Harrison 1977)
  \[ x \in [0, 1] \land y \in [0, \infty) \]

- congruences (Granger 1991)
  \[ x \equiv y \mod(2) \]

  \[ x + 2y - 3z \leq 3 \]
octagonal inequalities (Mine 2001)

\[ x - y \leq 3 \]
Overview of Abstract Interpretation

- **octagonal inequalities (Mine 2001)**
  \[ x - y \leq 3 \]

- **octahedral inequalities (Clariso & Cortadella 2004)**
  \[ x - y + z \leq 2 \]
Overview of Abstract Interpretation

- octagonal inequalities (Mine 2001)
  \[ x - y \leq 3 \]

- octahedral inequalities (Clariso & Cortadella 2004)
  \[ x - y + z \leq 2 \]

- ...
Overview of Abstract Interpretation

- octagonal inequalities (Mine 2001)
  
  \[ x - y \leq 3 \]

- octahedral inequalities (Clariso & Cortadella 2004)
  
  \[ x - y + z \leq 2 \]

- ...

- polynomial equalities and inequalities
  
  \[ x = y^2 \]

  \[ (a + 1)^2 > b^2 \geq a^2 \]
Sets of variable values overapproximated by abstract values

Abstract Interpretation: Overapproximation

Intervals

System States

Linear Inequalities

Polynomial Equalities
Abstract Interpretation: Operations

- Invariants generated by symbolic execution of system using abstract values
- Symbolic execution requires abstracting concrete operations on states:

  - Image
  - Projection
  - Union: merging in loops and conditionals
  - Intersection: guards in loops and conditionals

assignments
assignments
Termination is not guaranteed in general
Termination is not guaranteed in general
Termination is not guaranteed in general
Termination is not guaranteed in general
Termination is not guaranteed in general

Union in loops must be extrapolated: widening operator introduced to ensure termination
Termination is not guaranteed in general

Union in loops must be extrapolated: widening operator introduced to ensure termination
Introduction

- Need for program verification
- Invariants and abstract interpretation
- Polynomial invariants

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Why Care about Polynomial Invariants?

- **Linear invariants** used to verify many classes of systems:
  - Imperative programs
  - Logic programs
  - Hybrid systems
  - ...

But some applications require polynomial invariants: The abstract interpreter ASTRÉE employs polynomial invariants to verify absence of run-time errors in flight control software.
Why Care about Polynomial Invariants?

- **Linear invariants** used to verify many classes of systems:
  - Imperative programs
  - Logic programs
  - Hybrid systems
  - ...

- But some applications require **polynomial invariants**:

  The abstract interpreter **ASTRÉE** employs polynomial invariants to verify absence of run-time errors in flight control software.
Introduction

Generation of Invariant Polynomial Equalities
  - Related work
    - Abstract domain of ideals
    - Particular case: loops without nesting

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Related Work (1)

- Iterative fixpoint approaches
  - Forward propagation
    - Rodríguez-Carbonell & Kapur 2004
    - Colón 2004
  - Backward propagation
    - Müller-Olm & Seidl 2004
Related Work (1)

- Iterative fixpoint approaches
  - Forward propagation
    - Rodríguez-Carbonell & Kapur 2004
    - Colón 2004
  - Backward propagation
    - Müller-Olm & Seidl 2004

- Constraint-based approaches
  - Sankaranarayanan & Sipma & Manna 2004
## Related Work (2)

<table>
<thead>
<tr>
<th>Work</th>
<th>Restrictions</th>
<th>Conds =</th>
<th>Conds ≠</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS, POPL’04</td>
<td>bounded deg</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>SSM, POPL’04</td>
<td>fixed form</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MOS, IPL’04</td>
<td>fixed form</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>COL, SAS’04</td>
<td>bounded deg</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>RCK, SAS’04</td>
<td>bounded deg</td>
<td>yes</td>
<td>yes</td>
<td>yes*</td>
</tr>
<tr>
<td>RCK, ISSAC’04</td>
<td>no restriction</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Introduction

Generation of Invariant Polynomial Equalities
  - Related work
  - Abstract domain of ideals
  - Particular case: loops without nesting

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Overview of our Method

- States abstracted to ideal of polynomials evaluating to 0
Overview of our Method

- States abstracted to ideal of polynomials evaluating to 0
- Programming language admits
  - Polynomial assignments: `variable := polynomial`
  - Polynomial equalities and disequalities in conditions:
    \[ \text{polynomial} = 0 \quad \text{and} \quad \text{polynomial} \neq 0 \]
Overview of our Method

- States abstracted to ideal of polynomials evaluating to 0
- Programming language admits
  - Polynomial assignments: \( \text{variable} := \text{polynomial} \)
  - Polynomial equalities and disequalities in conditions:
    \[
    \text{polynomial} = 0, \quad \text{polynomial} \neq 0
    \]
- Implementation successfully applied to many programs
Overview of our Method

- States abstracted to ideal of polynomials evaluating to 0
- Programming language admits
  - Polynomial assignments: \( \text{variable} := \text{polynomial} \)
  - Polynomial equalities and disequalities in conditions:
    \[
    \text{polynomial} = 0 \quad , \quad \text{polynomial} \neq 0
    \]
- Implementation successfully applied to many programs
- Ideals of polynomials represented by special finite bases of generators: Gröbner bases
Overview of our Method

- States abstracted to ideal of polynomials evaluating to 0
- Programming language admits
  - Polynomial assignments: \textit{variable} := \textit{polynomial}
  - Polynomial equalities and disequalities in conditions:
    \[ \textit{polynomial} = 0 \quad , \quad \textit{polynomial} \neq 0 \]
- Implementation successfully applied to many programs
- Ideals of polynomials represented by special finite bases of generators: Gröbner bases
- Many tools available manipulating ideals, Gröbner bases, e.g. Macaulay 2, Maple
Intuitively, an ideal is a set of polynomials and all their consequences.
Intuitively, an **ideal** is a set of polynomials and all their consequences.

An **ideal** is a set of polynomials $I$ such that:

- $0 \in I$
- If $p, q \in I$, then $p + q \in I$
- If $p \in I$ and $q$ any polynomial, $pq \in I$
E.g. polynomials evaluating to 0 on a set of points $S$
Ideals of Polynomials (2)

- E.g. polynomials evaluating to 0 on a set of points $S$
- 0 evaluates to 0 everywhere

$$\forall \omega \in S, \quad 0(\omega) = 0$$
Ideals of Polynomials (2)

- E.g. polynomials evaluating to 0 on a set of points $S$
- $0$ evaluates to 0 everywhere

\[ \forall \omega \in S, \quad 0(\omega) = 0 \]

- If $p, q$ evaluate to 0 on $S$, then $p + q$ evaluates to 0 on $S$

\[ \forall \omega \in S, \quad p(\omega) = q(\omega) = 0 \implies p(\omega) + q(\omega) = 0 \]
Ideals of Polynomials (2)

- E.g. polynomials evaluating to 0 on a set of points \( S \)
- 0 evaluates to 0 everywhere

\[ \forall \omega \in S, \quad 0(\omega) = 0 \]

- If \( p, q \) evaluate to 0 on \( S \), then \( p + q \) evaluates to 0 on \( S \)

\[ \forall \omega \in S, \quad p(\omega) = q(\omega) = 0 \implies p(\omega) + q(\omega) = 0 \]

- If \( p \) evaluates to 0 on \( S \), then \( pq \) evaluates to 0 on \( S \)

\[ \forall \omega \in S, \quad p(\omega) = 0 \implies p(\omega) \cdot q(\omega) = 0 \]
E.g. multiples of a polynomial $p$, $\langle p \rangle$

- $0 = 0 \cdot p \in \langle p \rangle$
- $q_1 \cdot p + q_2 \cdot p = (q_1 + q_2)p \in \langle p \rangle$
- If $q_2$ is any polynomial, then $q_2 \cdot q_1 \cdot p \in \langle p \rangle$
E.g. multiples of a polynomial $p$, $\langle p \rangle$

- $0 = 0 \cdot p \in \langle p \rangle$
- $q_1 \cdot p + q_2 \cdot p = (q_1 + q_2)p \in \langle p \rangle$
- If $q_2$ is any polynomial, then $q_2 \cdot q_1 \cdot p \in \langle p \rangle$

In general, ideal generated by $p_1, \ldots, p_k$:

$$\langle p_1, \ldots, p_k \rangle = \left\{ \sum_{j=1}^{k} q_j \cdot p_j \text{ for arbitrary } q_j \right\}$$
Ideals of Polynomials (3)

- E.g. multiples of a polynomial $p$, $\langle p \rangle$
  
  $0 = 0 \cdot p \in \langle p \rangle$
  
  $q_1 \cdot p + q_2 \cdot p = (q_1 + q_2)p \in \langle p \rangle$
  
  If $q_2$ is any polynomial, then $q_2 \cdot q_1 \cdot p \in \langle p \rangle$

- In general, ideal generated by $p_1, \ldots, p_k$:

  $$\langle p_1, \ldots, p_k \rangle = \{ \sum_{j=1}^{k} q_j \cdot p_j \text{ for arbitrary } q_j \}$$

- Hilbert’s basis theorem: all ideals are finitely generated

  → there is finite representation for ideals
Several operations available. Given ideals $I, J$ in the variables $x_1, \ldots, x_n$: 
Operations with Ideals

- Several operations available. Given ideals $I, J$ in the variables $x_1, \ldots, x_n$:
  - projection: $I \cap \mathbb{C}[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$
Operations with Ideals

Several operations available. Given ideals $I$, $J$ in the variables $x_1, ..., x_n$:

- **projection**: $I \cap \mathbb{C}[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]$ 
- **addition**: $I + J = \{p + q \mid p \in I, q \in J\}$
Operations with Ideals

Several operations available. Given ideals $I$, $J$ in the variables $x_1, ..., x_n$:

- **projection**: $I \cap \mathbb{C}[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]$ 
- **addition**: $I + J = \{ p + q \mid p \in I, q \in J \}$ 
- **quotient**: $I : J = \{ p \mid \forall q \in J, p \cdot q \in I \}$
Operations with Ideals

Several operations available. Given ideals $I$, $J$ in the variables $x_1, ..., x_n$:

- **projection**: $I \cap \mathbb{C}[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]$
- **addition**: $I + J = \{p + q \mid p \in I, q \in J\}$
- **quotient**: $I : J = \{p \mid \forall q \in J, p \cdot q \in I\}$
- **intersection**: $I \cap J$
Operations with Ideals

- Several operations available. Given ideals $I$, $J$ in the variables $x_1, ..., x_n$:
  - projection: $I \cap \mathbb{C}[x_1, ..., x_{i-1}, x_{i+1}, ..., x_n]$
  - addition: $I + J = \{p + q \mid p \in I, q \in J\}$
  - quotient: $I : J = \{p \mid \forall q \in J, p \cdot q \in I\}$
  - intersection: $I \cap J$

- All operations implemented using Gröbner bases
- These operations are used in abstract semantics
Our Widening Operator

- Parametric widening $I \nabla_d J$
- Based on taking polynomials of $I \cap J$ of degree $\leq d$
Our Widening Operator

- Parametric widening $I \nabla_d J$
- Based on taking polynomials of $I \cap J$ of degree $\leq d$
- Definition uses Gröbner bases:

$$I \nabla_d J := V(\{p \in GB(I \cap J) \mid \deg(p) \leq d\})$$
**Our Widening Operator**

- Parametric widening $I \nabla_d J$
- Based on taking polynomials of $I \cap J$ of degree $\leq d$
- Definition uses **Gröbner bases**:

  $$I \nabla_d J := \text{V}(I(\{p \in GB(I \cap J) \mid \deg(p) \leq d\}))$$

- Termination guaranteed
Example

\[
a := 0; b := 0;
\]

\[
\text{while } b \neq c \text{ do }
\]

\[
a := a + 2b + 1; b := b + 1;
\]

\[
\text{end while}
\]
Example

\[ a := 0; b := 0; \]

while \( b \neq c \) do

\[ a := a + 2b + 1; b := b + 1; \]

end while

\[
\begin{align*}
F_0(I) &= \langle 0 \rangle \\
F_1(I) &= (\langle a \rangle + \langle I_0(a \leftarrow a') \rangle) \cap \mathbb{C}[a, b, c] \\
F_2(I) &= (\langle b \rangle + \langle I_1(b \leftarrow b') \rangle) \cap \mathbb{C}[a, b, c] \\
F_3(I) &= I_3 \bigtriangledown_2 (I_2 \cap I_6) \\
F_4(I) &= \langle I_3 \rangle : \langle b - c \rangle \\
F_5(I) &= I_4(a \leftarrow a - 2b - 1) \\
F_6(I) &= I_5(b \leftarrow b - 1) \\
F_7(I) &= I(V(I_3 + \langle b - c \rangle))
\end{align*}
\]
Example

\[ a := 0; b := 0; \]

while \( b \neq c \) do

\[ a := a + 2b + 1; b := b + 1; \]

end while

\[
F_0(I) = \langle 0 \rangle \\
F_1(I) = (\langle a \rangle + \langle I_0(a \leftarrow a') \rangle) \cap \mathbb{C}[a, b, c] \\
F_2(I) = (\langle b \rangle + \langle I_1(b \leftarrow b') \rangle) \cap \mathbb{C}[a, b, c] \\
F_3(I) = I_3 \nabla_2 (I_2 \cap I_6) \\
F_4(I) = \langle I_3 \rangle : \langle b - c \rangle \\
F_5(I) = I_4(a \leftarrow a - 2b - 1) \\
F_6(I) = I_5(b \leftarrow b - 1) \\
F_7(I) = I(V(I_3 + \langle b - c \rangle))
\]

In 6 steps found loop invariant:

\[ a = b^2 \]
Introduction

Generation of Invariant Polynomial Equalities

- Related work
- Abstract domain of ideals
- Particular case: loops without nesting

Applications of Polynomial Equality Invariants

- Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Particular case: loops without nesting

Are there programs for which no widening is required?
Particular case: loops without nesting

- Are there programs for which no widening is required?
- Yes: unnested loops with solvable assignments with eigenvalues in $\mathbb{Q}^+$
Particular case: loops without nesting

- Are there programs for which no widening is required?
- Yes: unnested loops with solvable assignments with eigenvalues in \( \mathbb{Q}^+ \)
- Solvable assignments generalize linear assignments
Are there programs for which no widening is required?

Yes: unnested loops with solvable assignments with eigenvalues in $\mathbb{Q}^+$

Solvable assignments generalize linear assignments

```plaintext
a := 0 ;
b := 0 ;
while b ≠ c do
    a := a + 2b + 1 ;
b := b + 1;
end while
```
Overview of the Method

\((a_n, b_n, c) \equiv \text{program state after } n \text{ loop iterations}\)

\[
\begin{align*}
    a_{n+1} &= a_n + 2b_n + 1 \\
    b_{n+1} &= b_n + 1
\end{align*}
\]

\[
\begin{align*}
    a_0 &= 0 \\
    s_0 &= 0
\end{align*}
\]
Overview of the Method

- \((a_n, b_n, c) \equiv \text{program state after } n \text{ loop iterations}\)

\[
\begin{align*}
   a_{n+1} &= a_n + 2b_n + 1 \\
   b_{n+1} &= b_n + 1
\end{align*}
\]

\[
\begin{align*}
   a_0 &= 0 \\
   s_0 &= 0
\end{align*}
\]

- Solution to recurrence:

\[
\begin{align*}
   a_n &= n^2 \\
   b_n &= n
\end{align*}
\]

- Program states characterized by \(\exists n (a = n^2 \land b = n)\)
Overview of the Method

\[ (a_n, b_n, c) \equiv \text{program state after } n \text{ loop iterations} \]

\[
\begin{align*}
    a_{n+1} &= a_n + 2b_n + 1 \\
    b_{n+1} &= b_n + 1
\end{align*}
\]

\[
\begin{align*}
    a_0 &= 0 \\
    s_0 &= 0
\end{align*}
\]

Solution to recurrence:
\[
\begin{align*}
    a_n &= n^2 \\
    b_n &= n
\end{align*}
\]

Program states characterized by \( \exists n (a = n^2 \land b = n) \)

Quantifier elimination: \( b = n \implies a = b^2 \) is loop invariant
Overview of the Method

- \((a_n, b_n, c) \equiv \text{program state after } n \text{ loop iterations}\)

\[
\begin{align*}
a_{n+1} &= a_n + 2b_n + 1 \\
b_{n+1} &= b_n + 1
\end{align*}
\]

- **Solution to recurrence:**
  \[
  \begin{align*}
a_n &= n^2 \\
b_n &= n
\end{align*}
  \]

- **Program states characterized by** \(\exists n (a = n^2 \land b = n)\)

- **Quantifier elimination:** \(b = n \implies a = b^2\) is loop invariant

- **Gröbner bases** can be used to eliminate loop counters
\[ x := R; \]
\[ y := 0; \]
\[ r := R^2 - N; \]

\textbf{while } ? \textbf{ do}

\hspace{1em} \textbf{if } ? \textbf{ then}

\hspace{2em} r := r + 2x + 1;

\hspace{2em} x := x + 1;

\hspace{1em} \textbf{else}

\hspace{2em} r := r - 2y - 1;

\hspace{2em} y := y + 1;

\hspace{1em} \textbf{end if}

\textbf{end while}
1st idea:

\begin{verbatim}
1. Compute invariants for two distinct loops:
   while r := r + x + 1; y := y + 1;
   do while r := r^2 y + 1;
   x := x + 1;
end while end while

2. Compute common invariants for both loops

Finding common invariants

Finding intersection of invariant ideals

But this is not sound!
\end{verbatim}
1st idea:

1. Compute invariants for two distinct loops:

```
while ? do
    r := r + 2x + 1;
    x := x + 1;
end while

while ? do
    r := r - 2y - 1;
    y := y + 1;
end while
```
1st idea:

1. Compute invariants for two distinct loops:

\[
\begin{align*}
\text{while } ? \text{ do} & \quad \text{while } ? \text{ do} \\
 r & := r + 2x + 1; \quad r & := r - 2y - 1; \\
 x & := x + 1; \quad y & := y + 1; \\
\text{end while} & \quad \text{end while}
\end{align*}
\]

2. Compute *common* invariants for both loops
Our Handling of Conditional Statements (2)

1st idea:

1. Compute invariants for two distinct loops:

   while ? do
   r := r + 2x + 1;
   x := x + 1;
   end while

   while ? do
   r := r − 2y − 1;
   y := y + 1;
   end while

2. Compute common invariants for both loops

Finding common invariants ≡ Finding intersection of invariant ideals
1st idea:

1. Compute invariants for two distinct loops:

   while ? do
   \[ r := r + 2x + 1; \]
   \[ x := x + 1; \]
   end while

   while ? do
   \[ r := r - 2y - 1; \]
   \[ y := y + 1; \]
   end while

2. Compute common invariants for both loops

   Finding common invariants \( \equiv \)
   Finding intersection of invariant ideals

   But this is not sound!
2nd idea: take intersection as initial condition and repeat
2nd idea: take intersection as initial condition and repeat

<table>
<thead>
<tr>
<th><strong>Program</strong></th>
<th><strong>Algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{x} := \bar{\alpha}; )</td>
<td>( I' := \langle 1 \rangle; I := \langle x_1 - \alpha_1, \ldots, x_m - \alpha_m \rangle; )</td>
</tr>
<tr>
<td>while ( ? ) do</td>
<td>while ( I' \neq I ) do</td>
</tr>
<tr>
<td>( \bar{x} := f(\bar{x}); )</td>
<td>( I' := I; )</td>
</tr>
<tr>
<td>or ( \bar{x} := g(\bar{x}); )</td>
<td>( I := \bigcap_{n=0}^{\infty} \left[ I(\bar{x} \leftarrow f^{-n}(\bar{x})) \right] \cap I(\bar{x} \leftarrow g^{-n}(\bar{x})) );</td>
</tr>
<tr>
<td>end while</td>
<td>end while</td>
</tr>
</tbody>
</table>
Properties of our Algorithm

- No widening employed!
- Termination in $n + 1$ steps, where $n =$ number of variables
No widening employed!

Termination in $n + 1$ steps, where $n =$ number of variables

Correct and complete:
finds all polynomial equality invariants
Properties of our Algorithm

- No widening employed!
- **Termination** in $n + 1$ steps, where $n = \text{number of variables}$
- **Correct and complete:** finds all polynomial equality invariants
- **Implemented in Maple:**

  1. Solving recurrences
  2. Eliminating variables
  3. Intersecting ideals

\[ \text{Gröbner bases} \]
Proof of Termination (1)

\[ x := 0; y := 0; \]
\[ \text{while } ? \text{ do} \]
\[ x := x + 1; \]
\[ \text{or} \]
\[ y := y + 1; \]
\[ \text{end while} \]
Proof of Termination (1)

\[ x := 0; y := 0; \]

\textbf{while} \ ? \ \textbf{do}

\[ x := x + 1; \]

\textbf{or}

\[ y := y + 1; \]

\textbf{end while}

Program states \( \equiv \mathbb{N} \times \mathbb{N} \)
Proof of Termination (1)

\[ x := 0; y := 0; \]

\textbf{while} ? \textbf{do}

\[ x := x + 1; \]

\textbf{or}

\[ y := y + 1; \]

\textbf{end while}

- Program states \( \equiv \mathbb{N} \times \mathbb{N} \)

- Initial state \( (x, y) = (0, 0) \longrightarrow \text{initial ideal} \langle x, y \rangle \)
Proof of Termination (2)

Step 0: $\langle x, y \rangle \rightarrow \{(0, 0)\}$ dimension 0
Step 0:  \( \langle x, y \rangle \rightarrow \{ (0, 0) \} \)  

dimension 0
Proof of Termination (2)

Step 0: \( \langle x, y \rangle \rightarrow \{(0, 0)\} \) dimension 0

Step 1: \( \langle xy \rangle \rightarrow \{(\alpha, 0)\} \cup \{(0, \alpha)\} \) dimension 1
Proof of Termination (2)

Step 0: \[ \langle x, y \rangle \rightarrow \{ (0, 0) \} \quad \text{dimension 0} \]

Step 1: \[ \langle xy \rangle \rightarrow \{ (\alpha, 0) \} \cup \{ (0, \alpha) \} \quad \text{dimension 1} \]

Step 2: \[ \langle 0 \rangle \rightarrow \mathbb{R}^2 \quad \text{dimension 2} \]
The dimension has increased at every step, and there is a finite number of variables!
Example

\[ x := R; \]
\[ y := 0; \]
\[ r := R^2 - N; \]

while ? do
  if ? then
    \[ r := r + 2x + 1; \]
    \[ x := x + 1; \]
  else
    \[ r := r - 2y - 1; \]
    \[ y := y + 1; \]
  end if
end while
Example

\begin{align*}
x & := R; \\
y & := 0; \\
r & := R^2 - N; \\
\textbf{while} \quad ? \quad \textbf{do} & \\
\quad \textbf{if} \quad ? \quad \textbf{then} & \\
\qquad r & := r + 2x + 1; \\
\qquad x & := x + 1; \\
\quad \textbf{else} & \\
\qquad r & := r - 2y - 1; \\
\qquad y & := y + 1; \\
\quad \textbf{end if} & \\
\textbf{end while} & \\
\end{align*}

Invariant polynomial equality:

\[ x^2 - y^2 = r + N \]
Introduction

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants
  - Imperative programs
  - Petri nets
  - Hybrid systems

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Imperative Programs

**Pre:** \( \{ \, N \geq 1 \, \} \)

\[ x := R; \quad y := 0; \quad r := R^2 - N; \]

**Inv:** \( \{ \, N \geq 1 \, \land \, x^2 - y^2 = r + N \, \} \)

**while** \( r \neq 0 \) **do**

\[ \text{if } r < 0 \text{ then} \]
\[ r := r + 2x + 1; \]
\[ x := x + 1; \]

\[ \text{else} \]
\[ r := r - 2y - 1; \]
\[ y := y + 1; \]

**end if**

**end while**

**Post:** \( \{ x \neq y \land N \mod (x - y) = 0 \} \)
Imperative Programs

Pre: \{ N \geq 1 \}
\begin{align*}
x & := R; \quad y := 0; \quad r := R^2 - N; \\
\text{Inv:} \quad & N \geq 1 \land x^2 - y^2 = r + N \}
\end{align*}

while \( r \neq 0 \) do
  if \( r < 0 \) then
    \begin{align*}
    r & := r + 2x + 1; \\
    x & := x + 1;
    \end{align*}
  else
    \begin{align*}
    r & := r - 2y - 1; \\
    y & := y + 1;
    \end{align*}
  end if
end while

Post: \{ x \neq y \land N \mod (x - y) = 0 \}

\[ N \geq 1 \implies R^2 - 0^2 = (R^2 - N) + N \]
Imperative Programs

Pre: \{ \ N \geq 1 \} 
\begin{align*}
x & := R; \ y := 0; \ r := R^2 - N; 
\end{align*}

Inv: \{ \ N \geq 1 \ \land \ x^2 - y^2 = r + N \}

while \ r \neq 0 \ do 
\begin{align*}
\text{if } r < 0 \text{ then} & \\
\quad r & := r + 2x + 1; \\
\quad x & := x + 1; 
\end{align*}
\begin{align*}
\text{else} & \\
\quad r & := r - 2y - 1; \\
\quad y & := y + 1; 
\end{align*}
end if
end while

Post: \{ x \neq y \ \land \ N \mod (x - y) = 0 \}

- N \geq 1 \implies R^2 - 0^2 = (R^2 - N) + N
- x^2 - y^2 = r + N \ \land \ r < 0 \implies (x + 1)^2 - y^2 = (r + 2x + 1) + N
Imperative Programs

**Pre:** \{ N \geq 1 \}

\begin{align*}
x & := R; \ y := 0; \ r := R^2 - N;
\end{align*}

**Inv:** \{ N \geq 1 \land x^2 - y^2 = r + N \}

while \( r \neq 0 \) do

- if \( r < 0 \) then
  \begin{align*}
  r & := r + 2x + 1; \\
x & := x + 1;
  \end{align*}

- else
  \begin{align*}
  r & := r - 2y - 1; \\
y & := y + 1;
  \end{align*}

end if

end while

**Post:** \{ x \neq y \land N \mod (x - y) = 0 \}

\begin{align*}
N \geq 1 & \implies R^2 - 0^2 = (R^2 - N) + N \\
x^2 - y^2 = r + N \land r < 0 & \implies (x + 1)^2 - y^2 = (r + 2x + 1) + N \\
x^2 - y^2 = r + N \land r > 0 & \implies x^2 - (y + 1)^2 = (r - 2y - 1) + N
\end{align*}
Imperative Programs

Pre: \{ N \geq 1 \}
\begin{align*}
x & := R; \quad y := 0; \quad r := R^2 - N;
\end{align*}

Inv: \{ N \geq 1 \land x^2 - y^2 = r + N \}

while \( r \neq 0 \) do
\begin{align*}
\text{if } r < 0 \text{ then } & \quad r := r + 2x + 1; \\
& \quad x := x + 1;
\end{align*}
\begin{align*}
\text{else } & \quad r := r - 2y - 1; \\
& \quad y := y + 1;
\end{align*}
end if
end while

Post: \{ x \neq y \land N \mod (x - y) = 0 \}

- \quad N \geq 1 \implies R^2 - 0^2 = (R^2 - N) + N
- \quad x^2 - y^2 = r + N \land r < 0 \implies (x + 1)^2 - y^2 = (r + 2x + 1) + N
- \quad x^2 - y^2 = r + N \land r > 0 \implies x^2 - (y + 1)^2 = (r - 2y - 1) + N
- \quad N \geq 1 \land x^2 - y^2 = r + N \implies x \neq y \land N \mod (x - y) = 0
Introduction

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants
  • Imperative programs
  • Petri nets
  • Hybrid systems

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Petri Nets: Introduction

- Petri nets: mathematical model for studying systems
  - concurrency
  - parallelism
  - non-determinism
Petri Nets: Introduction

- **Petri nets**: mathematical model for studying systems
  - concurrency
  - parallelism
  - non-determinism

- **Applications**:  
  - Manufacturing and Task Planning
  - Communication Networks
  - Hardware Design
Definitions

- A Petri net is a bipartite directed graph where:
  - Nodes partitioned into places (○) and transitions (|)
  - Arcs are labelled with a natural number
- A marking maps a number of tokens to each place
Dynamics (1)

- Dynamics of a Petri net described by
  - initial marking
  - firing of transitions
Dynamics (1)

- Dynamics of a Petri net described by
  - initial marking
  - firing of transitions

- A transition is enabled if there are $\geq$ tokens in each input place than indicated in the arcs
Dynamics (1)

- Dynamics of a Petri net described by
  - initial marking
  - firing of transitions
- A transition is enabled if there are $\geq$ tokens in each input place than indicated in the arcs
- When a transition is enabled, it can fire:
  1. the number of tokens indicated in the arcs is removed from input places
  2. tokens are produced in output places according to arcs
Dynamics (2)

\[ t_1 \rightarrow t_2 \]

\[ p_1 \]

\[ p_2 \]

\[ p_3 \]
Enabling of transitions may also depend on inhibitor arcs.

An inhibitor arc is an arc connecting place $p$ to transition $t$ so that there cannot be tokens in $p$ for $t$ to be enabled.
Enabling of transitions may also depend on inhibitor arcs.

An inhibitor arc is an arc connecting place $p$ to transition $t$ so that there cannot be tokens in $p$ for $t$ to be enabled.
Deadlocks are markings for which all transitions are disabled

Given a Petri net with an initial marking:

- Invariant properties of reachable states?
- Any deadlocks?

![Petri net diagram]

- $t_1$ disabled
- $t_2$ disabled

DEADLOCK !!
Deadlocks are markings for which all transitions are disabled.

Given a Petri net with an initial marking:

- Invariant properties of reachable states?
- Any deadlocks?
Define variable $x_i$ meaning number of tokens at place $p_i$.
Translation into Loop Programs

- Define variable $x_i$ meaning number of tokens at place $p_i$
- Initial marking transformed into sequence of initializing assignments
Translation into Loop Programs

- Define variable $x_i$ meaning number of tokens at place $p_i$
- Initial marking transformed into sequence of initializing assignments
- Transitions transformed into conditional statements
Translation into Loop Programs

- Define variable $x_i$ meaning number of tokens at place $p_i$
- Initial marking transformed into sequence of initializing assignments
- Transitions transformed into conditional statements
- Enabling of a transition with input place $p_i$ and label $c_i$:
  $$\cdots (x_i \neq 0) \land (x_i \neq 1) \land \cdots \land (x_i \neq c_i - 1) \cdots$$
Translation into Loop Programs

- Define variable $x_i$ meaning number of tokens at place $p_i$
- Initial marking transformed into sequence of initializing assignments
- Transitions transformed into conditional statements
- Enabling of a transition with input place $p_i$ and label $c_i$:
  \[ \cdots (x_i \neq 0) \land (x_i \neq 1) \land \cdots \land (x_i \neq c_i - 1) \cdots \]
- Enabling of a transition with inhibitor place $p_i$:
  \[ x_i = 0 \]
Translation into Loop Programs

- Define variable $x_i$ meaning number of tokens at place $p_i$
- Initial marking transformed into sequence of initializing assignments
- Transitions transformed into conditional statements
- Enabling of a transition with input place $p_i$ and label $c_i$:
  \[ \cdots (x_i \neq 0) \land (x_i \neq 1) \land \cdots \land (x_i \neq c_i - 1) \cdots \]
- Enabling of a transition with inhibitor place $p_i$: $x_i = 0$
- Firing of a transition
  - with input place $p_i$ and label $c_i$: $x_i := x_i - c_i$
  - with output place $p_i$ and label $c_i$: $x_i := x_i + c_i$
$x_1 := 1; x_2 := 1; x_3 := 2;$

while ? do

$t_1 : \text{if } x_1 \neq 0 \land x_2 \neq 0 \land x_3 \neq 0 \rightarrow$

$x_1 := x_1 - 1;$
$x_2 := x_2 + 2;$
$x_3 := x_3 - 1;$

$t_2 : \{ x_2 \neq 0 \land x_3 \neq 0 \land x_3 \neq 1 \rightarrow$

$x_1 := x_1 + 1;$
$x_2 := x_2 - 1;$
$x_3 := x_3 - 2;$

end if
end while
Translation into Programs (3)

\[
x_1 := 1; x_2 := 1; x_3 := 2;
\]

while ? do

\[t_1 : \text{if } x_1 = 0 \land x_2 \neq 0 \land x_3 \neq 0 \rightarrow\]
\[
x_1 := x_1 - 1;
x_2 := x_2 + 2;
x_3 := x_3 - 1;
\]

\[t_2 : \Box x_2 \neq 0 \land x_3 \neq 0 \land x_3 \neq 1 \rightarrow\]
\[
x_1 := x_1 + 1;
x_2 := x_2 - 1;
x_3 := x_3 - 2;
\]

end if

end while
Abstract interpretation is applied to the loop program to obtain polynomial invariants of the Petri net.
Abstract interpretation is applied to the loop program to obtain polynomial invariants of the Petri net.

Example:

**Deadlock**

**Initial Marking**

**Deadlock**
Polynomial invariants obtained:

\[ \text{Inv} = \begin{cases} 
5x_1 + 3x_2 + x_3 - 10 &= 0 \\
5x_3^2 + 2x_2 - 11x_3 &= 0 \\
x_2x_3 + 2x_3^2 - 5x_3 &= 0 \\
5x_2^2 - 17x_2 + 6x_3 &= 0 
\end{cases} \]
Polynomial invariants obtained:

\[ \text{Inv} = \begin{cases} 
5x_1 + 3x_2 + x_3 - 10 &= 0 \\
5x_2^2 + 2x_2 - 11x_3 &= 0 \\
x_2x_3 + 2x_3^2 - 5x_3 &= 0 \\
5x_2^2 - 17x_2 + 6x_3 &= 0 
\end{cases} \]

In this example invariants characterize reachability set

\[ \text{Inv} \iff (x_1, x_2, x_3) \in \{(0, 3, 1), (1, 1, 2), (2, 0, 0)\} \]
Polynomial invariants obtained:

\[
Inv = \begin{cases} 
5x_1 + 3x_2 + x_3 - 10 & = 0 \\
5x_3^2 + 2x_2 - 11x_3 & = 0 \\
x_2x_3 + 2x_3^2 - 5x_3 & = 0 \\
5x_2^2 - 17x_2 + 6x_3 & = 0
\end{cases}
\]

In this example invariants characterize reachability set

\[
Inv \Leftrightarrow (x_1, x_2, x_3) \in \{(0, 3, 1), (1, 1, 2), (2, 0, 0)\}
\]

In general overapproximation of reach set is obtained
Introduction

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants
- Imperative programs
- Petri nets
- Hybrid systems

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Hybrid System: discrete system in analog environment
Hybrid Systems: Introduction

- Hybrid System: discrete system in analog environment
- Examples:
  - A thermostat that heats/cools depending on the temperature in the room

![Diagram of a thermostat showing heating and cooling with maximum and minimum temperature limits.](image-url)
Hybrid Systems: Introduction

- Hybrid System: discrete system in analog environment

Examples:

- A thermostat that heats/cools depending on the temperature in the room

- A robot controller that changes the direction of movement if the robot is too close to a wall.
Hybrid Systems: Introduction

- Hybrid System: discrete system in analog environment
- Examples:
  - A thermostat that heats/cools depending on the temperature in the room
  - A robot controller that changes the direction of movement if the robot is too close to a wall.
  - A biochemical reaction whose behaviour depends on concentration of substances in environment
A hybrid system is a finite automaton with real-valued variables that change continuously according to a system of differential equations at each location.
A hybrid system is a finite automaton with real-valued variables that change continuously according to a system of differential equations at each location.
A hybrid system is a finite automaton with real-valued variables that change continuously according to a system of differential equations at each location.

Restricted to linear differential equations at locations.
A computation is a sequence of states (discrete location, valuation of variables)

\[(l_0, x_0), (l_1, x_1), (l_2, x_2), \ldots\]

such that
A computation is a sequence of states (discrete location, valuation of variables)

\[(l_0, x_0), (l_1, x_1), (l_2, x_2), \ldots\]

such that

1. Initial state \((l_0, x_0)\) satisfies the initial condition
A computation is a sequence of states (discrete location, valuation of variables)

\[(l_0, x_0), (l_1, x_1), (l_2, x_2), \ldots\]

such that

1. Initial state \((l_0, x_0)\) satisfies the initial condition
2. For each consecutive pair of states \((l_i, x_i), (l_{i+1}, x_{i+1})\):
   - Discrete transition: there is a transition of the automaton \((l_i, l_{i+1}, \rho)\) such that \((x_i, x_{i+1}) \models \rho\)
A computation is a sequence of states (discrete location, valuation of variables)

\[(l_0, x_0), (l_1, x_1), (l_2, x_2), \ldots\]

such that

1. Initial state \((l_0, x_0)\) satisfies the initial condition
2. For each consecutive pair of states \((l_i, x_i), (l_{i+1}, x_{i+1})\):
   - Discrete transition: there is a transition of the automaton \((l_i, l_{i+1}, \rho)\) such that \((x_i, x_{i+1}) \models \rho\)
   - Continuous evolution: there is a trajectory going from \(x_i\) to \(x_{i+1}\) along the flow determined by the differential equation \(\dot{x} = Ax + B\) at location \(l_i = l_{i+1}\)
Goal: generate invariant polynomial equalities
Goal: generate invariant polynomial equalities

- We know how to deal with discrete systems
- How to handle continuous evolution?
Goal: generate invariant polynomial equalities
- We know how to deal with discrete systems
- How to handle continuous evolution?

Problem:
computing polynomial invariants of linear systems of differential equations
Form of the Solution

Solution to $\dot{x} = Ax + B$ can be expressed as polynomials in $t$, $e^{\pm at}$, $\cos(bt)$, $\sin(bt)$, where $\lambda = a + bi$ are eigenvalues of matrix $A$.

$$
\begin{pmatrix}
\dot{x} \\
\dot{y} \\
\dot{v}_x \\
\dot{v}_y
\end{pmatrix} =
\begin{pmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & -1/2 \\
0 & 0 & 1/2 & 0
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
v_x \\
v_y
\end{pmatrix}
$$

$$
\begin{cases}
x &= x^* + 2 \sin(t/2) v_x^* + (2 \cos(t/2) - 2) v_y^* \\
y &= y^* + (-2 \cos(t/2) + 2) v_x^* + 2 \sin(t/2) v_y^* \\
v_x &= \cos(t/2) v_x^* - \sin(t/2) v_y^* \\
v_y &= \sin(t/2) v_x^* + \cos(t/2) v_y^*
\end{cases}
$$
Elimination of Time

Idea: eliminate terms depending on $t$ from solution:

- transform solution into polynomials using new variables
- eliminate by means of Gröbner bases using auxiliary equations
Elimination of Time

Idea: eliminate terms depending on $t$ from solution:

- transform solution into polynomials using new variables
- eliminate by means of Gröbner bases using auxiliary equations

\[
\begin{align*}
\text{SOLUTION} & \quad \left\{ 
\begin{align*}
x &= x^* + 2\, zv_x^* + (2\, w - 2)\, v_y^* \\
y &= y^* + (-2\, w + 2)\, v_x^* + 2\, zv_y^* \\
v_x &= wv_x^* - zv_y^* \\
v_y &= zv_x^* + wv_y^*
\end{align*}
\right. \\
\text{INITIAL CONDITIONS} & \quad \left\{ 
\begin{align*}
v_x^* &= 2 \\
v_y^* &= -2
\end{align*}
\right. \\
\text{AUXILIARY EQUATIONS} & \quad \left\{ 
\begin{align*}
w^2 + z^2 &= 1
\end{align*}
\right. \\
\Downarrow & \quad v_x^2 + v_y^2 = 8 \text{ (conservation of energy)}
\end{align*}
\]
**INITIAL CONDITIONS**

\[ v_x = 2 \]
\[ v_y = -2 \]
\[ x = y = b = 0 \]

**RIGHT**

\[
\begin{align*}
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{v}_x &= \dot{\dot{y}} = 0 \\
\dot{b} &= 0
\end{align*}
\]

**MAGNETIC**

\[
\begin{align*}
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{v}_x &= -\frac{v_y}{2} \\
\dot{v}_y &= \frac{v_x}{2} \\
\dot{b} &= 0
\end{align*}
\]

**LEFT**

\[
\begin{align*}
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{v}_x &= \dot{\dot{y}} = 0 \\
\dot{b} &= 0
\end{align*}
\]

\[ x = d \rightarrow \text{skip} \]

\[ x = 0 \rightarrow v_x := -v_x ; b := b + 1 \]

**RIGHT** \[ v_y = -2 \land v_x = 2 \land 2db - 8b + y + x = 0 \]

**MAGNETIC** \[ x - 2v_y - d = 4 \land v_x^2 + v_y^2 = 8 \land 2v_x + y + 2db - 8b + d = 4 \]

**LEFT** \[ v_y = -2 \land v_x = -2 \land 2db - 8b + y - x = 8 \]
Introduction

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Linear equalities
[Karr’76]

Polynomial equalities
[Colon’04]
Linear equalities
[Karr’76]

Polynomial equalities
[Colon’04]

Linear inequalities
[Cousot & Halbwachs’78]

Polynomial inequalities
[Bagnara & Rodríguez-Carbonell & Zaffanella’05]


\begin{verbatim}
\text{while} \ ? \ \text{do}

\hspace{1cm} a := a + 1 ;
\hspace{1cm} b := b + c ;
\hspace{1cm} c := c + 2 ;

\text{end while}
\end{verbatim}

\end{document}
\[ a := 0 ; \]
\[ b := 0 ; \]
\[ c := 1 ; \]

\{ a = 0 \land b = 0 \land c = 1 \} 

\textbf{while} ? \textbf{do} 

\[ a := a + 1 ; \]
\[ b := b + c ; \]
\[ c := c + 2 ; \]

\textbf{end while}
From Linear to Polynomial Equalities

\[
\begin{align*}
a &:= 0 ; \\
b &:= 0 ; \\
c &:= 1 ; \\
\{ \ a = b \land c = 2a + 1 \ \} \\
\text{while } \ ? \ \text{do} \\
\quad a &:= a + 1 ; \\
\quad b &:= b + c ; \\
\quad c &:= c + 2 ; \\
\text{end while}
\end{align*}
\]
\begin{equation}
\begin{align*}
a &:= 0 \\
b &:= 0 \\
c &:= 1 \\
\end{align*}
\end{equation}

\begin{equation}
\{ c = 2a + 1 \}
\end{equation}

\textbf{while } ? \textbf{ do }

\begin{equation}
\begin{align*}
a &:= a + 1 \\
b &:= b + c \\
c &:= c + 2 \\
\end{align*}
\end{equation}

\textbf{end while}

\begin{equation}
\{ c = 2a + 1 \}
\end{equation}

Loop invariant
\[ a := 0 ; \]
\[ b := 0 ; \]
\[ c := 1 ; \]

Introduce new variable \( s \) standing for \( a^2 \)

\[
\text{while } ? \text{ do}
\]
\[
\begin{align*}
  a & := a + 1 ; \\
  b & := b + c ; \\
  c & := c + 2 ;
\end{align*}
\]

end while
From Linear to Polynomial Equalities

\[
\begin{align*}
a &= 0 ; \\
b &= 0 ; \\
c &= 1 ; \\
s &= 0 ;
\end{align*}
\]

while \(?\) do

\[
\begin{align*}
a &= a + 1 ; \\
b &= b + c ; \\
c &= c + 2 ; \\
s &= s + 2a + 1 ;
\end{align*}
\]

end while

Introduce new variable \(s\) standing for \(a^2\)

Extend program with new variable \(s\)

\[
\begin{align*}
 a &= 0 & \rightarrow & s &= 0 \\
a &= a + 1 & \rightarrow & s &= s + 2a + 1
\end{align*}
\]
$a := 0$ ;
$b := 0$ ;
c := 1 ;
$s := 0$ ;
{ $a = 0 \land b = 0 \land c = 1 \land s = 0$ }
while ? do

\[
\begin{align*}
a &:= a + 1 ; \\
b &:= b + c ; \\
c &:= c + 2 ; \\
s &:= s + 2a + 1 ;
\end{align*}
\]
end while
\[ a := 0 ; \]
\[ b := 0 ; \]
\[ c := 1 ; \]
\[ s := 0 ; \]
\[ \{ a = b \land b = s \land c = 2a + 1 \} \]
while \( ? \) do

\[ a := a + 1 ; \]
\[ b := b + c ; \]
\[ c := c + 2 ; \]
\[ s := s + 2a + 1 ; \]
end while
From Linear to Polynomial Equalities

\[ a := 0 ; \]
\[ b := 0 ; \]
\[ c := 1 ; \]
\[ s := 0 ; \]
\[ \{ b = s \land c = 2a + 1 \} \]

\textbf{Loop invariant}
\[ \{ b = a^2 \land c = 2a + 1 \} \]

is more precise

\[ a := a + 1 ; \]
\[ b := b + c ; \]
\[ c := c + 2 ; \]
\[ s := s + 2a + 1 ; \]

\textbf{end while}
From Linear to Polynomial Inequalities

\{ \text{Pre} : b \geq 0 \} \\
a := 0 ;

\textbf{while} \ (a + 1)^2 \leq b \ \textbf{do}

\hspace{1cm} a := a + 1 ;

\textbf{end while}

\{ \text{Post} : (a + 1)^2 > b \land b \geq a^2 \}
\{ \text{Pre} : \ b \geq 0 \ \}\}{ \ \text{Post} : \ (a + 1)^2 > b \land b \geq a^2 \ } \\
\begin{align*}
\text{while } (a + 1)^2 \leq b \text{ do } \\
\quad & a := a + 1 ; \\
\text{end while}
\end{align*}

Linear analysis cannot deal with the quadratic condition \((a + 1)^2 \leq b\)
\{ \text{Pre} : b \geq 0 \} \\

a := 0 ; \\

\{ a \geq 0 \land b \geq 0 \} \\
\text{while } (a + 1)^2 \leq b \text{ do} \\
\quad a := a + 1 ; \\
\text{end while} \\

\{ \text{Post} : (a + 1)^2 > b \land b \geq a^2 \}
\begin{align*}
\{ \text{Pre} : & \quad b \geq 0 \} \\
\text{end while} \\
\{ \text{Post} : & \quad (a + 1)^2 > b \land b \geq a^2 \} \end{align*}

Introduce new variable $s$ standing for $a^2$

Extend program with new variable $s$

\begin{align*}
a & := 0 \quad \rightarrow \quad s := 0 \\
a & := a + 1 \quad \rightarrow \quad s := s + 2a + 1
\end{align*}
{ Pre : \( b \geq 0 \) }

\begin{align*}
    a &:= 0 ; \\
    s &:= 0 ; \\
    \{ b \geq s \land \cdots \} \\
\end{align*}

while \( (a + 1)^2 \leq b \) do

\begin{align*}
    a &:= a + 1 ; \\
    s &:= s + 2a + 1 ; \\
\end{align*}

end while

\{ Post : (a + 1)^2 > b \land b \geq a^2 \}
Linearization of Polynomial Constraints

- Abstract values = sets of constraints
- Given a degree bound $d$, all terms $x^\alpha$ with $\deg(x^\alpha) \leq d$ are considered as different and independent variables
**Vector Spaces ↔ Polynomial Cones**

\[ \text{polynomial} = 0 \]

- \( \forall \) polynomial \( p, p \sim p = 0 \)
- Vector space = set of polynomials \( V \) s.t.
  - \( 0 \in V \)
  - \( \forall p, q \in V \) and \( \lambda, \mu \in \mathbb{R} \), \( \lambda p + \mu q \in V \)

\[
\begin{align*}
0 &= 0 \\
p &= 0 & q &= 0 & \lambda, \mu \in \mathbb{R} \\
\lambda p + \mu q &= 0
\end{align*}
\]
Vector Spaces ↔ Polynomial Cones

**polynomial = 0**

- ∀ polynomial $p$, $p \sim p = 0$
- Vector space = set of polynomials $V$ s.t.
  - $0 \in V$
  - $\forall p, q \in V$ and $\lambda, \mu \in \mathbb{R}$, $\lambda p + \mu q \in V$

  \[
  0 = 0
  \]

  \[
  p = 0 \quad q = 0 \quad \lambda, \mu \in \mathbb{R}
  \]

  \[
  \lambda p + \mu q = 0
  \]

**polynomial ≥ 0**

- ∀ polynomial $p$, $p \sim p \geq 0$
- Polynomial cone = set of polynomials $C$ s.t.
  - $1 \in C$
  - $\forall p, q \in C$ and $\lambda, \mu \in \mathbb{R}_+$, $\lambda p + \mu q \in C$

  \[
  1 \geq 0
  \]

  \[
  p \geq 0 \quad q \geq 0 \quad \lambda, \mu \in \mathbb{R}_+
  \]

  \[
  \lambda p + \mu q \geq 0
  \]
Explicitly Adding Other Inference Rules

\[ \text{polynomial} = 0 \]

\[ p = 0 \quad \text{deg}(pq) \leq d \]

\[ pq = 0 \]
polynomial \ = \ 0

p = 0 \quad \text{deg}(pq) \leq d

pq = 0

\text{polynomial} \ \geq \ 0

p \geq 0 \quad p \leq 0 \quad \text{deg}(pq) \leq d

pq = 0

p \geq 0 \quad q \geq 0 \quad \text{deg}(pq) \leq d

pq \geq 0
Introduction

Generation of Invariant Polynomial Equalities

Applications of Polynomial Equality Invariants

Generation of Invariant Polynomial Inequalities

Conclusions and Future Work
Conclusions

- Designed a new abstract domain for generating invariant polynomial equalities based on ideals of polynomials.
Conclusions

- Designed a new abstract domain for generating invariant polynomial equalities based on ideals of polynomials
- Identified a class of programs for which all polynomial equality invariants can be generated
Conclusions

- Designed a new abstract domain for generating invariant polynomial equalities based on ideals of polynomials.
- Identified a class of programs for which all polynomial equality invariants can be generated.
- Applied polynomial equality invariants to verifying imperative programs, Petri nets and hybrid systems.
Conclusions

- Designed a new abstract domain for generating invariant polynomial equalities based on ideals of polynomials.
- Identified a class of programs for which all polynomial equality invariants can be generated.
- Applied polynomial equality invariants to verifying imperative programs, Petri nets and hybrid systems.
- Designed a new abstract domain for generating invariant polynomial inequalities based on polynomial cones.
Future Work

- Extend the techniques to **interprocedural** analyses
Future Work

- Extend the techniques to interprocedural analyses
- Develop methods for tuning the precision/efficiency trade-off
Future Work

- Extend the techniques to **interprocedural** analyses
- Develop methods for **tuning** the precision/efficiency trade-off
- Find **new areas of application** for polynomial invariants
Future Work

- Extend the techniques to **interprocedural** analyses
- Develop methods for **tuning the precision/efficiency trade-off**
- Find **new areas of application** for polynomial invariants
- Design **specific widening operators** for the context of polynomial invariants


E. Rodríguez-Carbonell, D. Kapur. Program verification using automatic generation of polynomial invariants. ICTAC’04.

E. Rodríguez-Carbonell, A. Tiwari. Generating polynomial invariants for hybrid systems. HSCC’05.

