Automatic Generation of
Polynomial Loop Invariants:
Algebraic Foundations

Enric Rodríguez-Carbonell
Universitat Politècnica
de Catalunya

Deepak Kapur
University of New Mexico
Overview of the Talk

1. **Motivation** for automatically generating invariants
2. Simple loops with **sequences of assignments**
3. Loops including **conditional statements**
4. **Algorithm** for generating polynomial invariants
5. **Termination** of the algorithm
Motivation

Program Verification

- Program verification failed due to:
  - program annotation by hand
  - weak theorem provers

- Current theorem provers are quite powerful

- About program annotation:
  - Pre/postconditions: useful documentation
  - Loop invariants: tedious to write

$\Rightarrow$ Automatic generation of loop invariants
Sequences of Assignments

Example: Square Root Program

{Pre: $N \geq 0$}
$a := 0; s := 1; t := 1;$
while $(s \leq N)$ do
    $a := a + 1;$
    $s := s + t + 2;$
    $t := t + 2;$
end while

{Post: $a^2 \leq N < (a + 1)^2$}

- Need invariant to verify program
- Good invariant: $a^2 \leq N \land t = 2a + 1 \land s = (a + 1)^2$
Sequences of Assignments
Generating Invariants (1)

- Program states \( \equiv \) solution to the recurrence

\[
\begin{align*}
a_{n+1} &= a_n + 1 \\
s_{n+1} &= s_n + t_n + 2 \\
t_{n+1} &= t_n + 2
\end{align*}
\]

\[
\begin{align*}
a_0 &= 0 \\
s_0 &= 1 \\
t_0 &= 1
\end{align*}
\]

\((a_n, s_n, t_n) \equiv \text{program state after } n \text{ loop iterations}\)
Sequences of Assignments
Generating Invariants (2)

\[
\begin{align*}
    a_n &= n \\
    s_n &= (n + 1)^2 \\
    t_n &= 2n + 1
\end{align*}
\]

- The **infinite** formula

\[
(a = 0 \land s = 1 \land t = 1) \lor (a = 1 \land s = 4 \land t = 3) \lor \cdots \equiv \\
\equiv \bigvee_{n=0}^{\infty} (a = n \land s = (n + 1)^2 \land t = 2n + 1)
\]

is invariant

- Want a **finite** invariant formula!
Sequences of Assignments
Eliminating Loop Counters

- The infinite formula can be replaced by
  \[ \exists n (a = n \land s = (n + 1)^2 \land t = 2n + 1) \]
- Need for quantifier elimination
- In the example it is obvious:
  \[ a = n \implies s = (a + 1)^2 \land t = 2a + 1 \] is loop invariant
- **Gröbner bases** can be used to eliminate auxiliary variables such as loop counters
Polynomial Invariants Form an Ideal

- For any program state \((a, s, t)\),
  
  \[ s - (a + 1)^2 = 0 \]
  
  \[ t - (2a + 1) = 0 \]

- For any polynomials \(p, q\),
  
  \[ p(a, s, t)(s - (a + 1)^2) + q(a, s, t)(t - (2a + 1)) = 0 \]

- In general polynomial invariants form an ideal
Handling Conditional Statements

Example: Factor Program

\{\text{Pre: } N \geq 1 \land N \mod 2 = 1 \land R^2 \geq N > (R - 1)^2 \}\}

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

- Good invariant: $N \geq 1 \land N + r = x^2 - y^2$
Handling Conditional Statements
Generating Invariants (1)

1st idea:

1. Compute invariants for two distinct loops:

\[
\text{while } \text{true do} \quad \text{while } \text{true do}
\]
\[
\quad r := r + 2x + 1; \quad r := r - 2y - 1;
\]
\[
\quad x := x + 1; \quad y := y + 1;
\]
\[
\text{end while} \quad \text{end while}
\]

2. Compute \textit{common} invariants for both loops

- Finding \textit{common} invariants $\equiv$
  - Finding \textit{intersection} of polynomial invariant ideals

- Gröbner bases used to compute \textit{intersection of ideals}
Handling Conditional Statements
Generating Invariants (2)

while true do
  \( r := r + 2x + 1; \)
  \( x := x + 1; \)
end while

while true do
  \( r := r - 2y - 1; \)
  \( y := y + 1; \)
end while

\(< y , -r - N + x^2 > \)  \(< x - R , r - R^2 + N + y^2 > \)
\(< x^2 - r - N - y^2 , yx - Ry , y^3 - R^2y + ry + Ny > \)

**Problem:** not all polynomials in the intersection are invariants
- The only invariant polynomial is \( x^2 - r - N - y^2 \)
- Others are not invariants of the original loop
Handling Conditional Statements
Generating Invariants (3)

- Tree of all possible execution paths:

- Found common invariants to the *two* extreme paths
- True invariants are common to *all* paths!
Handling Conditional Statements
Generating Invariants (4)

- 2nd idea: intersecting with more paths
- For example: paths with at most one alternation

\[
\langle x^2 - r - N - y^2, yx - Ry, y^3 - R^2y + ry + Ny \rangle
\]
\[
\langle x^2 - r - N - y^2 \rangle
\]
Algorithm for Computing Invariants (1)

Program

\[ x := \bar{x}; \]

while true do

\[ \bar{x} := f(\bar{x}); \]

or

\[ \bar{x} := g(\bar{x}); \]

end while

Algorithm

\[ I' := \langle 1 \rangle; I := \langle x_1 - \alpha_1, \ldots, x_m - \alpha_m \rangle; \]

while \( I' \neq I \) do

\[ I' := I; \]

\[ I := \bigcap_{n=0}^{\infty} \left[ I(\bar{x} \leftarrow f^{-n}(\bar{x})) \right. \]

\[ \left. \cap I(\bar{x} \leftarrow g^{-n}(\bar{x})) \right] \]

end while
Algorithm for Computing Invariants (2)

- After $N$ iterations:
  
  $I \equiv$ intersection for all paths with $\leq N - 1$ alternations
Algorithm for Computing Invariants (3)

- The value of $I$ stabilizes
- Termination in $O(m)$ iterations, where $m =$ number of variables
- Correctness and completeness proofs in the report
- Implemented in Maple:
  1. Solving recurrences
  2. Eliminating variables
  3. Intersecting ideals

\{ Gröbner bases \}
Algorithm for Computing Invariants (4)

Table of Examples

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>COMPUTING</th>
<th>VARIABLES</th>
<th>BRANCHES</th>
<th>TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>freire1</td>
<td>√</td>
<td>2</td>
<td>1</td>
<td>&lt; 3 s.</td>
</tr>
<tr>
<td>freire2</td>
<td>√</td>
<td>3</td>
<td>1</td>
<td>&lt; 5 s.</td>
</tr>
<tr>
<td>cohencu</td>
<td>cube</td>
<td>4</td>
<td>1</td>
<td>&lt; 5 s.</td>
</tr>
<tr>
<td>cousot</td>
<td>toy</td>
<td>2</td>
<td>2</td>
<td>&lt; 4 s.</td>
</tr>
<tr>
<td>divbin</td>
<td>division</td>
<td>3</td>
<td>2</td>
<td>&lt; 5 s.</td>
</tr>
<tr>
<td>dijkstra</td>
<td>√</td>
<td>3</td>
<td>2</td>
<td>&lt; 6 s.</td>
</tr>
<tr>
<td>fermat2</td>
<td>factor</td>
<td>3</td>
<td>2</td>
<td>&lt; 4 s.</td>
</tr>
<tr>
<td>wensley2</td>
<td>division</td>
<td>4</td>
<td>2</td>
<td>&lt; 5 s.</td>
</tr>
<tr>
<td>euclidex2</td>
<td>gcd</td>
<td>6</td>
<td>2</td>
<td>&lt; 6 s.</td>
</tr>
<tr>
<td>lcm2</td>
<td>lcm</td>
<td>4</td>
<td>2</td>
<td>&lt; 5 s.</td>
</tr>
<tr>
<td>factor</td>
<td>factor</td>
<td>4</td>
<td>4</td>
<td>&lt; 20 s.</td>
</tr>
</tbody>
</table>

PC Linux Pentium 4 2.5 Ghz
Termination (1)

Toy program

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

while true do

\[ x := x + 1; \\
\text{or}
\]

\[ y := y + 1; \\]

end while

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

\[ f(x, y) = (x + 1, y) \]

\[ g(x, y) = (x, y + 1) \]

- Initial state \((x, y) = (0, 0) \rightarrow \) initial ideal \(\langle x, y \rangle\)
Termination (2)

- 1st iteration of the algorithm

1st branch: $f(x, y) = (x + 1, y)$

\[
\begin{align*}
x_{n+1} &= x_n + 1, \\
y_{n+1} &= y_n
\end{align*}
\]

\[
\begin{array}{llll}
\{ & x_0 = 0 & \rightarrow & x_n = n \\
\{ & y_0 = 0 & \rightarrow & y_n = 0 \\
\end{array}
\]

Invariant ideal 1st branch: \langle y \rangle

2nd branch: $g(x, y) = (x, y + 1)$

\[
\begin{align*}
x_{n+1} &= x_n \\
y_{n+1} &= y_n + 1
\end{align*}
\]

\[
\begin{array}{llll}
\{ & x_0 = 0 & \rightarrow & x_n = 0 \\
\{ & y_0 = 0 & \rightarrow & y_n = n \\
\end{array}
\]

Invariant ideal 2nd branch: \langle x \rangle

Intersection ideal: \langle xy \rangle
Termination (3)

- **Step 0**: \( \langle x, y \rangle \rightarrow \{(0, 0)\} \), dimension 0
- **Step 1**: \( \langle xy \rangle \rightarrow \{(\alpha, 0)\} \cup \{(0, \alpha)\} \), dimension 1

The dimension has increased!
Termination (4)

- 2nd iteration of the algorithm
  - Ideal computed: \( \{0\} \)
  - Solution space: \( \mathbb{R}^2 \), dimension 2

The dimension has increased again!

- At each step of the algorithm, the dimension increases
- If there are \( m \) variables, it terminates in \( O(m) \) steps
Related Work (1)

- Karr (1976): *linear equalities*
- Cousot, Halbwachs (1978): *linear inequalities*
- Colón, Sankaranarayanan, Sipma (2003): *linear inequalities*
- Müller-Olm, Seidl (2003): *polynomial equalities*
- Sankaranarayanan et al (2004): *polynomial equalities*
- Müller-Olm, Seidl (2004): *polynomial equalities*
- Rodríguez-Carbonell, Kapur (2004): *polynomial equalities*
Related Work (2)
Overview Polynomial Invariants

<table>
<thead>
<tr>
<th>Work</th>
<th>Restrictions</th>
<th>Nesting</th>
<th>Conditions</th>
<th>Complete</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>[MOS03]</td>
<td>bounded degree</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>intraprocedural</td>
</tr>
<tr>
<td>[SSM03]</td>
<td>prefixed form</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>interprocedural</td>
</tr>
<tr>
<td>[MOS04]</td>
<td>prefixed form</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>interprocedural</td>
</tr>
<tr>
<td>[RCK04]</td>
<td>bounded degree</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>interprocedural</td>
</tr>
<tr>
<td>[RCK04]</td>
<td>no restriction</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>interprocedural</td>
</tr>
</tbody>
</table>
Conclusions

- Correct and complete algorithm for polynomial invariants
- First method not bounding \textit{a priori} degree of invariants
- Applicable to loops \textbf{without nesting}
- Terminates in $O(m)$ iterations, where $m =$ number of variables
- Implemented and being integrated into a verifier
- Part of a \textbf{general framework} for generating invariants
  - Rich theory in algebraic geometry and polynomial ideals
  - Beyond numbers and polynomials we need:
    - solving recurrences
    - eliminating variables
    - ...