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

$\Longrightarrow$ Automatic generation of loop invariants

Sequences of Assignments
Example: Square Root Program

```
{Pre: N ≥ 0 }
a := 0; s := 1; t := 1;
while (s ≤ N) do
      a := a + 1;
      s := s + t + 2;
      t := t + 2;
end while
{Post: a² ≤ N < (a + 1)² }
```

- Need invariant to verify program
- Good invariant: $a^2 \leq N \wedge t = 2a + 1 \wedge s = (a + 1)^2$

Sequences of Assignments
Generating Invariants (1)

- Program states $\equiv$ solution to the recurrence

$$\left\{\begin{array}{lcl} a_{n+1} & = & a_n + 1 \\ s_{n+1} & = & s_n + t_n + 2 \\ t_{n+1} & = & t_n + 2 \end{array}\right. \quad , \quad \left\{\begin{array}{lcl} a_0 & = & 0 \\ s_0 & = & 1 \\ t_0 & = & 1 \end{array}\right.$$

$(a_n, s_n, t_n) \equiv$ program state after n loop iterations

Sequences of Assignments
Generating Invariants (2)

$$\begin{cases} a_n & = & n \\ s_n & = & (n+1)^2 \\ t_n & = & 2n+1 \end{cases}$$

- The **infinite** formula

$$(a = 0 \wedge s = 1 \wedge t = 1) \vee (a = 1 \wedge s = 4 \wedge t = 3) \vee \cdots \equiv$$

$$\equiv \bigvee_{n=0}^{\infty} (a = n \wedge s = (n+1)^2 \wedge t = 2n+1)$$

  is invariant

- Want a **finite** invariant formula !

Sequences of Assignments
Eliminating Loop Counters

- The infinite formula can be replaced by

$$\exists n(a = n \wedge s = (n+1)^2 \wedge t = 2n+1)$$

- Need for quantifier elimination
- In the example it is obvious:
  $a = n \Longrightarrow s = (a+1)^2 \wedge t = 2a+1$ 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

$\{\texttt{Pre:}\ N \geq 1 \wedge N \bmod 2 = 1 \wedge R^2 \geq N > (R-1)^2\,\}$

```
x := R; y := 0; r := R^2 - N;
while (r ≠ 0) do
    if (r < 0) then
        r := r + 2x + 1; x := x + 1;
    else
        r := r - 2y - 1; y := y + 1;
    end if
end while
```

$\{\texttt{Post:}\ x \neq y \wedge N \bmod (x-y) = 0\,\}$

- Good invariant: $N \geq 1 \wedge\ N + r = x^2 - y^2$

Handling Conditional Statements
Generating Invariants (1)

- 1st idea:

  1. Compute invariants for two distinct loops:

     while *true* **do**
     $r := r + 2x + 1;$
     $x := x + 1;$
     end while

     while *true* **do**
     $r := r - 2y - 1;$
     $y := y + 1;$
     end while

  2. Compute *common* invariants for both loops

- Finding *common* invariants $\equiv$
  Finding *intersection* of polynomial invariant ideals

- Gröbner bases used to compute 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
```

$\langle\, y \;,\; -r - N + x^2 \,\rangle$

$\langle\, x - R \;,\; r - R^2 + N + y^2 \,\rangle$

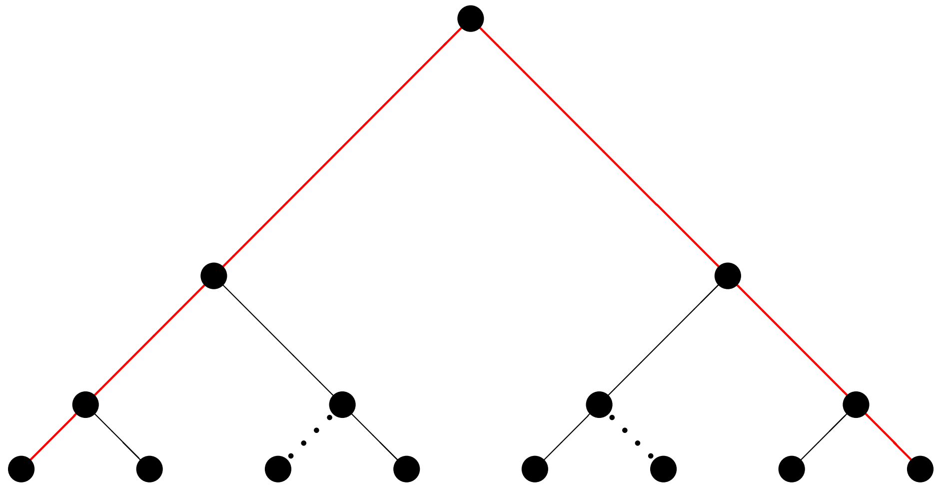
$$\langle\, x^2 - r - N - y^2 \;,\; yx - Ry \;,\; y^3 - R^2 y + ry + Ny \,\rangle$$

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)

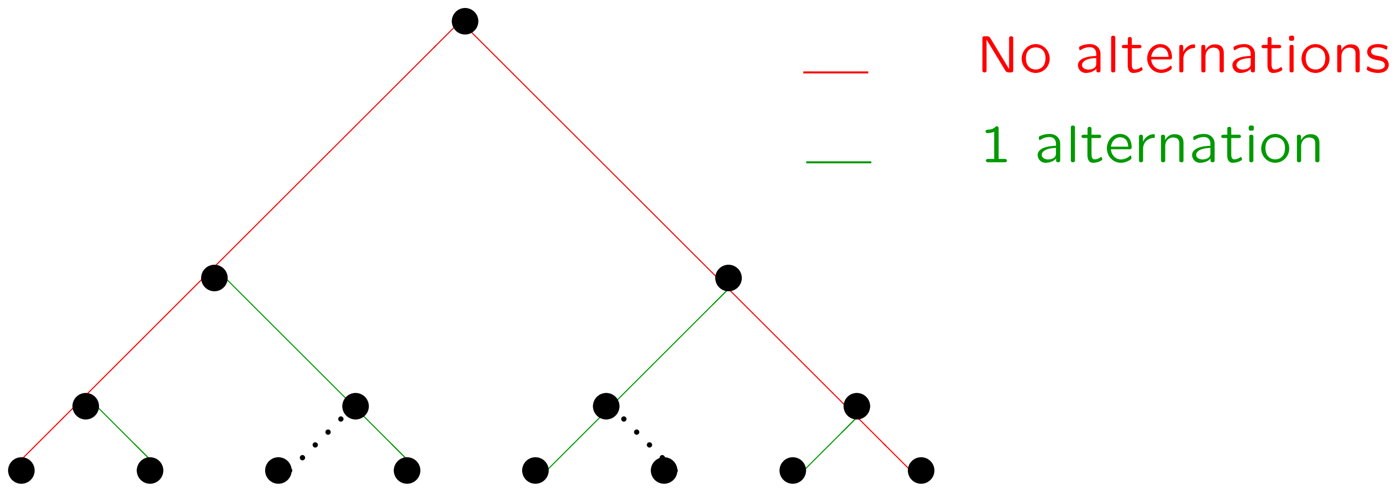
- Tree of all possible execution paths:



- 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

$x := \bar{\alpha};$

while *true* **do**

$\quad \bar{x} := f(\bar{x});$

$\quad$ **or**

$\quad \bar{x} := g(\bar{x});$

end while

Algorithm

$I' := \langle 1 \rangle; I := \langle x_1 - \alpha_1, \cdots, x_m - \alpha_m \rangle;$

while $I' \neq I$ **do**

$\quad I' := I;$

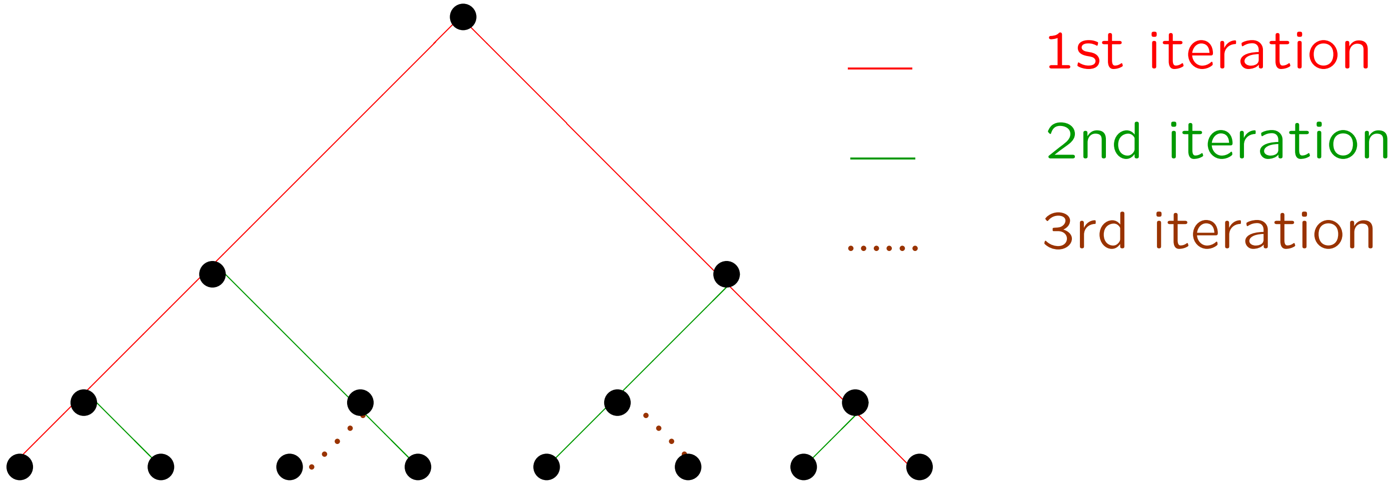
$\quad I := \bigcap_{n=0}^{\infty} [\, I(\bar{x} \leftarrow f^{-n}(\bar{x}))$

$\quad\quad \bigcap I(\bar{x} \leftarrow g^{-n}(\bar{x}))\,];$

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
  $$\left.\begin{array}{l}\text{2. Eliminating variables}\\ \text{3. Intersecting ideals}\end{array}\right\}\text{Gröbner bases}$$

Algorithm for Computing Invariants (4)
Table of Examples

PROGRAM	COMPUTING	VARIABLES	BRANCHES	TIMING
freire1	$\sqrt[2]{}$	2	1	$<$ 3 s.
freire2	$\sqrt[3]{}$	3	1	$<$ 5 s.
cohencu	cube	4	1	$<$ 5 s.
cousot	toy	2	2	$<$ 4 s.
divbin	division	3	2	$<$ 5 s.
dijkstra	$\sqrt[2]{}$	3	2	$<$ 6 s.
fermat2	factor	3	2	$<$ 4 s.
wensley2	division	4	2	$<$ 5 s.
euclidex2	gcd	6	2	$<$ 6 s.
lcm2	lcm	4	2	$<$ 5 s.
factor	factor	4	4	$<$ 20 s.

PC Linux Pentium 4 2.5 Ghz

Termination (1)

Toy program

$x := 0; y := 0;$
while *true* **do**
$\quad x := x + 1;$
or
$\quad y := y + 1;$
end while

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

$$f(x, y) = (x + 1, y) \qquad\qquad g(x, y) = (x, y + 1)$$

- Initial state $(x, y) = (0, 0) \longrightarrow$ initial ideal $\langle x, y \rangle$

Termination (2)

- 1st iteration of the algorithm

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

$$\left\{\begin{array}{lcl} x_{n+1} & = & x_n + 1 \\ y_{n+1} & = & y_n \end{array}\right., \left\{\begin{array}{lcl} x_0 & = & 0 \\ y_0 & = & 0 \end{array}\right. \longrightarrow \left\{\begin{array}{lcl} x_n & = & n \\ y_n & = & 0 \end{array}\right.$$

Invariant ideal 1st branch: $\langle y \rangle$

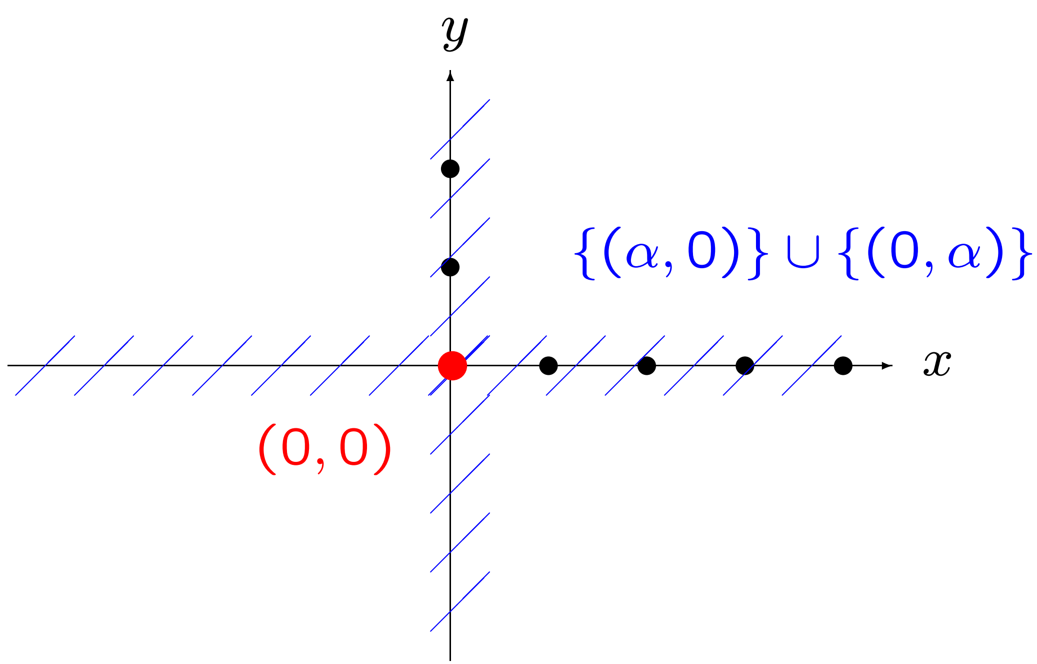
2nd branch: $g(x,y) = (x, y+1)$

$$\left\{\begin{array}{lcl} x_{n+1} & = & x_n \\ y_{n+1} & = & y_n + 1 \end{array}\right., \left\{\begin{array}{lcl} x_0 & = & 0 \\ y_0 & = & 0 \end{array}\right. \longrightarrow \left\{\begin{array}{lcl} x_n & = & 0 \\ y_n & = & n \end{array}\right.$$

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

Work	Restrictions	Nesting	Conditions	Complete	Application
[MOS03]	bounded degree	yes	no	yes	*intra*procedural
[SSM03]	prefixed form	yes	yes	no	*inter*procedural
[MOS04]	prefixed form	yes	yes	yes	*inter*procedural
[RCK04]	bounded degree	yes	yes	yes	*inter*procedural
[RCK04]	no restriction	no	no	yes	*inter*procedural

Conclusions

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