An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants

Enric Rodríguez-Carbonell

Universitat Politècnica de Catalunya
Barcelona

Deepak Kapur

University of New Mexico
Albuquerque

Introduction
Why are invariants important ?

- It is necessary to verify safety properties of systems:
  - Imperative programs
  - Reactive systems
  - Concurrent systems
  - Etc.
- Most often systems have an infinite number of states
  $\longrightarrow$ techniques for finite-state systems cannot be applied
- Exact reachable set of a system is not computable generally
- **Solution:** overapproximate reachable states $\rightarrow$

  INVARIANTS
- Abstract interpretation allows to compute invariants

Introduction
Related Work

Different classes of invariants:

- intervals (Cousot & Cousot 1976, Harrison 1977)
- linear inequalities (Cousot & Halbwachs 1978, Colón & Sankaranarayanan & Sipma 2003)
- congruences (Granger 1991)
- octagonal inequalities (Mine 2001)
- trapezoidal congruences (Masdupuy 1993)
- ...
- **polynomial equalities** (Müller-Olm & Seidl 2004,Sankaranarayanan & Sipma & Manna 2004, Rodríguez-Carbonell & Kapur 2004)

Introduction

Overview Polynomial Invariants

Work	Restrictions	Equality Conditions	Disequality Conditions	Complete
MOS, POPL'04	bounded degree	no	no	yes
SSM, POPL'04	prefixed form	yes	no	no
MOS, IPL'04	prefixed form	no	yes	yes
RCK, ISSAC'04	no restriction	no	no	yes
RCK, SAS'04	bounded degree	yes	yes	yes*

Overview of the Talk

1. **Overview of the Method**

2. **Ideals of Polynomials**

3. **Abstract Semantics**

4. **Widening Operator**

Overview of the Method (1)

- Generates polynomial invariants by abstract interpretation
- Program states $\equiv$ values variables take
- States abstracted to ideal of polynomials vanishing on states
- Programming language admits
  - Polynomial assignments: $variable := polynomial$
  - Polynomial equalities and disequalities in conditions:

$$polynomial = 0 \quad , \quad polynomial \neq 0$$

- Parametric widening ∇_d
- If conditions are ignored and assignments are linear, finds **all** polynomial invariants of degree $\leq d$

Overview of the Method (2)

- Our implementation has been successfully applied to a number of programs
- Ideals of polynomials represented by finite bases of generators: Gröbner bases
- There are several packages manipulating ideals and Gröbner bases
- Our implementation uses the algebraic geometry tool *Macaulay 2*

Ideals of Polynomials
Preliminaries

- An **ideal** is a set of polynomials I such that
  1. $0 \in I$
  2. If $p, q \in I$, then $p + q \in I$
  3. If $p \in I$ and q any polynomial, $pq \in I$

- Example 1: polynomials vanishing on a set of points A
  1. 0 vanishes everywhere
  2. If p, q vanish on A, then $p + q$ vanishes on A
  3. If p vanishes on A, then pq vanishes on A

- Ideal generated by $p_1, \ldots, p_k$:

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

Ideals of Polynomials
Ideals as Abstract Values

- Program states $\equiv$ values variables take
- States abstracted to ideal of polynomials vanishing on states
- Abstraction function I

$$\begin{aligned} I : \{\text{sets of states}\} &\longrightarrow \{\text{ideals}\} \\ A &\longmapsto \{\textbf{polynomials vanishing on } A\} \end{aligned}$$

- Concretization function V

$$\begin{aligned} V : \{\text{ideals}\} &\longrightarrow \{\text{sets of states}\} \\ I &\longmapsto \{\textbf{zeroes of } I\} \end{aligned}$$

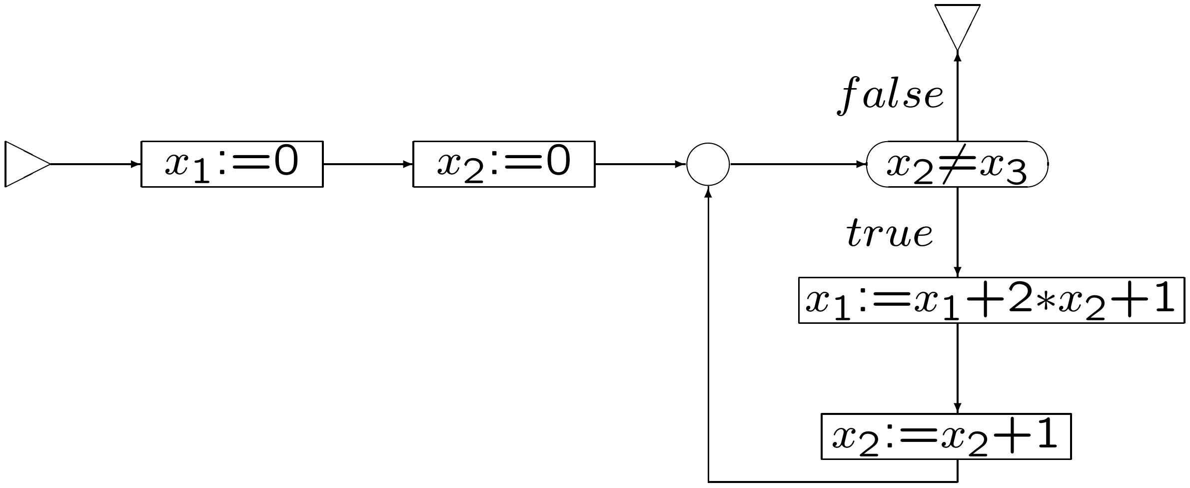
$$\langle p_1, ..., p_k \rangle \longleftrightarrow p_1 = 0 \wedge \cdots \wedge p_k = 0$$

Abstract Semantics
Programming Model (1)

Programs $\equiv$ finite connected flowcharts

- Entry node
- Assignment nodes: polynomial assignments
- Test nodes: polynomial dis/equalities
- Simple/loop junction nodes
- Exit nodes

Abstract Semantics
Programming Model (2)



$x_1 := 0; x_2 := 0;$

while $x_2 \neq x_3$ **do**

$\quad x_1 := x_1 + 2 * x_2 + 1; x_2 := x_2 + 1;$

end while

Abstract Semantics
Assignments

- Assignment node labelled with $x_i := f(\bar{x})$
- Input ideal: $\langle p_1, ..., p_k \rangle$
- Output ideal:
  - Want to express in terms of ideals

    $\exists x_i'(x_i = f(x_i \leftarrow x_i') \wedge p_1(x_i \leftarrow x_i') = 0 \wedge \cdots \wedge p_k(x_i \leftarrow x_i') = 0)$

    where $x_i' \equiv$ previous value of x_i before the assignment
  - **Solution:**
    - eliminate x_i' from the ideal

$$\langle x_i - f(x_i \leftarrow x_i'), p_1(x_i \leftarrow x_i'), ..., p_k(x_i \leftarrow x_i') \rangle$$

Abstract Semantics
Tests: Polynomial Equalities

- Test node labelled with $q = 0$
- Input ideal: $\langle p_1, ..., p_k \rangle$
- Output ideal: (*true* path)
  - Want to express in terms of ideals

$$p_1 = 0 \wedge \cdots \wedge p_k = 0 \wedge q = 0$$

  - **Solution:**
    - Add q to list of generators of input ideal
    - Take maximal set of polynomials with same zeroes

$$\mathbf{IV}(p_1, ..., p_k, q)$$

Abstract Semantics
Tests: Polynomial Disequalities

- Test node labelled with $q \neq 0$
- Input ideal: $\langle p_1, ..., p_k \rangle$
- Output ideal: (*true* path)
  - Want to express in terms of ideals

$$p_1 = 0 \wedge \cdots \wedge p_k = 0 \wedge q \neq 0$$

  - **Solution:**
    - **quotient ideal** $\langle p_1, ..., p_k \rangle : \langle q \rangle \equiv$
      maximal ideal of polynomials vanishing on
      zeroes of $\langle p_1, ..., p_k \rangle$ $\setminus$ zeroes of $\langle q \rangle$

Abstract Semantics
Simple Junction Nodes (1)

- Input ideals (one for each path):

  Path 1: $\langle p_{11}, ..., p_{1k_1} \rangle$

  ...

  Path l: $\langle p_{l1}, ..., p_{lk_l} \rangle$

- Output ideal:
  - Want to express in terms of ideals

    $$\bigvee_{i=1}^{l} \bigwedge_{j=1}^{k_i} p_{ij} = 0$$

  - **Solution:**
    - Take *common* polynomials for all paths $\equiv$ Compute *intersection* of all input ideals

      $$\bigcap_{i=1}^{l} \langle p_{i1}, ..., p_{ik_i} \rangle$$

Abstract Semantics
Simple Junction Nodes (2)

Example:

- Input ideal 1st branch: $\langle x \rangle$
- Input ideal 2nd branch: $\langle x - 1 \rangle$
- Input ideal 3rd branch: $\langle x - 2 \rangle$
- Output ideal:

$$\langle x \rangle \cap \langle x - 1 \rangle \cap \langle x - 2 \rangle = \langle x(x-1)(x-2) \rangle$$

Degree increases !!

Abstract Semantics
Loop Junction Nodes (1)

- Input ideals: $J_1, \cdots, J_l$
- Previous output ideal: I
- Output ideal:
  - As with simple junction nodes:

$$I \cap \left(\bigcap_{i=1}^{l} J_i\right)$$

  - **Problem:** Non-termination of forward propagation !
  - **Solution: WIDENING** $\longrightarrow$ bounding degree

Abstract Semantics
Loop Junction Nodes (2)

Example:

$x := 0$;
while *true* **do**
    $x := x + 1$;
end while

Generating loop invariant by forward propagation:

- 1st iteration: $\langle x \rangle$
- 2nd iteration: $\langle x(x-1) \rangle$
- 3rd iteration: $\langle x(x-1)(x-2) \rangle$
- ...

Unless we bound the degree, the procedure does not terminate

Widening Operator
Definition

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

$$I \nabla_d J := \mathbf{IV}(\{p \in GB(I \cap J) \mid \mathsf{deg}(p) \leq d\})$$

Widening Operator
A Completeness Result

- **THEOREM**. If conditions are ignored and assignments are linear, forward propagation computes **all** invariants of degree $\leq d$
- Key ideas of the proof:
  - $I \nabla_d J$ retains all polynomials of degree d of $I \cap J$
  - Graded term orderings used in Gröbner bases: glex, grevlex

Table of Examples

PROGRAM	COMPUTING	d	VARS	IF'S	LOOPS	LOOP DEPTH	TIME
cohencu	cube	3	5	0	1	1	2.45
dershowitz	real division	2	7	1	1	1	1.71
divbin	integer division	2	5	1	2	1	1.91
euclidex1	Bezout's coefs	2	10	0	2	2	7.15
euclidex2	Bezout's coefs	2	8	1	1	1	3.69
fermat	divisor	2	5	0	3	2	1.55
prod4br	product	3	6	3	1	1	8.49
freire1	integer sqrt	2	3	0	1	1	0.75
hard	integer division	2	6	1	2	1	2.19
lcm2	lcm	2	6	1	1	1	2.03
readers	simulation	2	6	3	1	1	4.15

Future Work

- Design widening operators not bounding degree
- Integrate with linear inequalities
- Study abstract domains for polynomial inequalities
- Apply to other classes of programs

Conclusions

- Method for generating polynomial invariants
- Based on abstract interpretation
- Programming language admits
  - Polynomial assignments
  - Polynomial dis/equalities in conditions
- If conditions are ignored and assignments are linear, finds **all** polynomial invariants of degree $\leq d$
- Implemented using Macaulay 2
- Successfully applied to many programs