Will Algebraic Geometry

Rescue

Program Verification?

Deepak Kapur

Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA

Joint Work with Enric Rodríguez-Carbonell

Overview of the Talk

1. Program Verification and Role of Loop Invariants
2. Recent Successes in Generating Loop Invariants Automatically

   a) Quantifier-Elimination: Eliminating Program Variables from Parameterized Formulas Hypothesized as Assertions(ACA-2004).

   b) Ideal-Theoretic Methods based on Fixed Point Computations (ISSAC-2004, SAS-2004, ICTAC-2004).

3. Experimental Results
4. Role of Algebraic Geometry
5. How could these results be extended/generalized?

Program Verification

- Invariants or inductive assertions play a key role in verifying properties of programs:
  - Pre/postconditions: useful documentation
  - program annotation by hand put considerable burden on programmers
  - Loop invariants: difficult to discover/generate

Program Verification

- Invariants or inductive assertions play a key role in verifying properties of programs:
  - Pre/postconditions: useful documentation
  - program annotation by hand put considerable burden on programmers
  - Loop invariants: difficult to discover/generate

- **How to generate loop invariants automatically?**

Example: Multiplication Program

{Pre: $N \geq 0$}
$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

if $y \ mod \ 2 = 0$ **then** $x := 2x; \quad y := y \ div \ 2$
else $x := 2x; \quad y := (y-1) \ div \ 2; \quad z := x + z;$

end while
{Post: $z = M * N$}

Example: Multiplication Program

{Pre: $N \geq 0$}
$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

if $y \ mod \ 2 = 0$ **then** $x := 2x; \quad y := y \ div \ 2$
else $x := 2x; \quad y := (y-1) \ div \ 2; \quad z := x + z;$

end while
{Post: $z = M * N$}

Inductive assertion associated with a program control point is a formula over program variables (and input) that is preserved by an execution path.

Example: Multiplication Program

{Pre: $N \geq 0$}
$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

if $y \ mod \ 2 = 0$ **then** $x := 2x; \quad y := y \ div \ 2$
else $x := 2x; \quad y := (y-1) \ div \ 2; \quad z := x + z;$

end while
{Post: $z = M * N$}

Inductive assertion associated with a program control point is a formula over program variables (and input) that is preserved by an execution path.

Loop invariant is a formula typically associated with the entry point of the loop which is true whenever control passes through the entry point.

Example: Multiplication Program

{Pre: $N \geq 0$}
$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

if $y \ mod \ 2 = 0$ **then** $x := 2x; \quad y := y \ div \ 2$
else $x := 2x; \quad y := (y-1) \ div \ 2; \quad z := x + z;$

end while
{Post: $z = M * N$}

Inductive assertion associated with a program control point is a formula over program variables (and input) that is preserved by an execution path.

Loop invariant is a formula typically associated with the entry point of the loop which is true whenever control passes through the entry point.

Invariant: $z = M * N - x * y$.

Example: Square Root Program

{Pre: $N \geq 0$}
$a := 0; \quad s := 1; \quad t := 1;$
while $(s \leq N)$ **do**
$\quad a := a + 1; \qquad s := s + t + 2; \qquad t := t + 2;$
end while
{Post: $a^2 \leq N < (a+1)^2$}

- Invariant: $a^2 \leq N \wedge t = 2a + 1 \wedge s = (a+1)^2$

Example: Multiplication Program

$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

if $y \ mod \ 2 = 0$ **then** $x := 2x; \quad y := y \ div \ 2$
else $x := 2x; \quad y := (y - 1) \ div \ 2; \quad z := x + z;$
end while

Example: Multiplication Program

$x := M; y := N; z := 0;$

while $(y \neq 0)$ **do**

$\{I(x,y,z) : (A\ xyz + B\ xy + C\ xz + D\ yz + E\ x + F\ y + G\ z + H = 0)\}$

if $y\ mod\ 2 = 0$ **then** $x := 2x;\quad y := y\ div\ 2$

else $x := 2x;\quad y := (y-1)\ div\ 2;\quad z := x + z;$

end while

Example: Multiplication Program

$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

$\{I(x,y,z) : (A\ xyz + B\ xy + C\ xz + D\ yz + E\ x + F\ y + G\ z + H = 0)\}$

if $y\ mod\ 2 = 0$ **then** $x := 2x;\quad y := y\ div\ 2$
else $x := 2x;\quad y := (y-1)\ div\ 2;\quad z := x + z;$
end while

Verification Conditions:

$\forall x, y, z:\quad [(\ y\ mod\ 2 = 0 \wedge I(x,y,z)) \Rightarrow I(2x, y\ div\ 2, z)] \wedge$
$[(\ y\ mod\ 2 \neq 0 \wedge I(x,y,z)) \Rightarrow I(2x, (y-1)\ div\ 2, x+z)] \wedge$
$I(M, N, 0)$

Example: Multiplication Program

$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

$\{I(x,y,z) : (A\ xyz + B\ xy + C\ xz + D\ yz + E\ x + F\ y + G\ z + H = 0)\}$

if $y\ mod\ 2 = 0$ **then** $x := 2x;\quad y := y\ div\ 2$
else $x := 2x;\quad y := (y-1)\ div\ 2;\quad z := x + z;$
end while

Verification Conditions:

$\forall x, y, z:\quad [(\ y\ mod\ 2 = 0 \wedge I(x,y,z)) \Rightarrow I(2x, y\ div\ 2, z)] \wedge$
$[(\ y\ mod\ 2 \neq 0 \wedge I(x,y,z)) \Rightarrow I(2x, (y-1)\ div\ 2, x+z)] \wedge$
$I(M,N,0)$

Are there values of A, B, C, D, E, F, G, H such that $I(x,y,z)$ for those values make the above formula valid?

Example: Multiplication Program

$x := M; y := N; z := 0;$
while $(y \neq 0)$ **do**

$\{I(x,y,z) : (A\ xyz + B\ xy + C\ xz + D\ yz + E\ x + F\ y + G\ z + H = 0)\}$

if $y\ mod\ 2 = 0$ **then** $x := 2x;\quad y := y\ div\ 2$
else $x := 2x;\quad y := (y-1)\ div\ 2;\quad z := x + z;$
end while

Verification Conditions:

$\forall x, y, z:\quad [(\ y\ mod\ 2 = 0 \wedge I(x,y,z)) \Rightarrow I(2x, y\ div\ 2, z)] \wedge$
$[(\ y\ mod\ 2 \neq 0 \wedge I(x,y,z)) \Rightarrow I(2x, (y-1)\ div\ 2, x+z)] \wedge$
$I(M,N,0)$

Are there values of A, B, C, D, E, F, G, H such that $I(x,y,z)$ for those values make the above formula valid?

Eliminate x, y, z to get a formula in A, B, C, D, E, F, G, H.

Quantifier Elimination

First Path:

$(2A\ xuz + 2B\ xu + C\ xz + 2D\ uz + E\ x + 2F\ u + G\ z + H = 0)$

$\Rightarrow (2A\ xuz + 2B\ xu + 2C\ xz + D\ uz + 2E\ x + F\ u + G\ z + H = 0)$

The conclusion becomes $C\ xz - D\ uz + E\ x - F\ u = 0$.

If $C = D = E = F = 0$, then the above formula is valid.

Quantifier Elimination

First Path:

$(2A\ xuz + 2B\ xu + C\ xz + 2D\ uz + E\ x + 2F\ u + G\ z + H = 0)$

$\Rightarrow (2A\ xuz + 2B\ xu + 2C\ xz + D\ uz + 2E\ x + F\ u + G\ z + H = 0)$

The conclusion becomes $C\ xz - D\ uz + E\ x - F\ u = 0$.

If $C = D = E = F = 0$, then the above formula is valid.

$$I(x, y, z) = A\ xyz + B\ xy + G\ z + H = 0$$

Quantifier Elimination

First Path:

$(2A\ xuz + 2B\ xu + C\ xz + 2D\ uz + E\ x + 2F\ u + G\ z + H = 0)$
$\Rightarrow (2A\ xuz + 2B\ xu + 2C\ xz + D\ uz + 2E\ x + F\ u + G\ z + H = 0)$

The conclusion becomes $C\ xz - D\ uz + E\ x - F\ u = 0$.

If $C = D = E = F = 0$, then the above formula is valid.

$$I(x, y, z) = A\ xyz + B\ xy + G\ z + H = 0$$

Second Path:

$(A\ x(2\ u + 1)z + B\ x(2\ u + 1) + G\ z + H = 0) \Rightarrow$
$(2A\ xu(x + z) + 2B\ xu + G\ (x + z) + H = 0)$

The conclusion becomes $2A\ x^2u - A\ xz + (G - B)\ x = 0$.

Quantifier Elimination

First Path:

$(2A\ xuz + 2B\ xu + C\ xz + 2D\ uz + E\ x + 2F\ u + G\ z + H = 0)$
$\Rightarrow (2A\ xuz + 2B\ xu + 2C\ xz + D\ uz + 2E\ x + F\ u + G\ z + H = 0)$

The conclusion becomes $C\ xz - D\ uz + E\ x - F\ u = 0$.

If $C = D = E = F = 0$, then the above formula is valid.

$$I(x, y, z) = A\ xyz + B\ xy + G\ z + H = 0$$

Second Path:

$(A\ x(2\ u + 1)z + B\ x(2\ u + 1) + G\ z + H = 0) \Rightarrow$
$(2A\ xu(x + z) + 2B\ xu + G\ (x + z) + H = 0)$

The conclusion becomes $2A\ x^2u - A\ xz + (G - B)\ x = 0$.

If $A = 0, G = B$, then the condition due to the second path is valid.

From initial values, $I(M, N, 0) : (BMN + H = 0)$, which implies $H = -BMN$.

Quantifier Elimination

First Path:

$(2A\ xuz + 2B\ xu + C\ xz + 2D\ uz + E\ x + 2F\ u + G\ z + H = 0)$
$\Rightarrow (2A\ xuz + 2B\ xu + 2C\ xz + D\ uz + 2E\ x + F\ u + G\ z + H = 0)$

The conclusion becomes $C\ xz - D\ uz + E\ x - F\ u = 0$.

If $C = D = E = F = 0$, then the above formula is valid.

$$I(x, y, z) = A\ xyz + B\ xy + G\ z + H = 0$$

Second Path:

$(A\ x(2\ u + 1)z + B\ x(2\ u + 1) + G\ z + H = 0) \Rightarrow$
$(2A\ xu(x + z) + 2B\ xu + G\ (x + z) + H = 0)$

The conclusion becomes $2A\ x^2u - A\ xz + (G - B)\ x = 0$.

If $A = 0, G = B$, then the condition due to the second path is valid.

From initial values, $I(M, N, 0) : (BMN + H = 0)$, which implies $H = -BMN$.

This gives

$I(x, y, z) = (B\ xy + B\ z - BMN = 0)$, which simplifies to:

$$I(x, y, z) = (xy + z - MN = 0).$$

Simple Abstract Program and Invariant Computation

Initialization

while b **do**

$\{I(\bar{x}, \bar{u})\}$

Loop body

end while

Verification Condition: (ignoring tests)

$$\forall \ \bar{x} : \quad I(\bar{x}, \bar{u}) \Rightarrow [I(\bar{x}, \bar{u})|^{\bar{e}}_{\bar{x}} \wedge \cdots \wedge I(\bar{x}, \bar{u})|^{\bar{f}}_{\bar{x}}]$$

- Eliminate $\bar{x}$ to get a formula in $\bar{u}$.
- Generate values of $\bar{u}$ which make the formula true.
- Plug these values of $\bar{u}$ in $I(\bar{x}, \bar{u})$ to get an invariant.

Method for Automatically Generating Invariants by Quantifier Elimination

1. Hypothesize assertions, which are parametrized formulas, at various points in a program.
   - Typically entry of every loop and entry and exit of every procedure suffice.
2. Generate verification conditions for every path in the program (a path from an assertion to another assertion including itself).
   - Depending upon the logical language chosen to write invariants, approximations of assignments and test conditions may be necessary.
3. Find a formula expressed only using parameters using quantifier elimination on program variables.
4. Every assignment of parameter values which make the formula true, gives an invariant.
   - If no parameter values can be found, then invariants of hypothesized forms may (do) not exist.
   - If all assignments making the formula true can be finitely described, invariants generated may be (is) the strongest of the hypothesized form.

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$
while $(x > 0)$ **do**

    $x := x - 1; \quad n := m + 2; \quad m := m + 1$
    while $(m > 0)$ **do**

        $s := s + 1; \quad m := m - 1$
    end while
    $m := n$
end while

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$

while $(x > 0)$ **do**

$\{I(m, n, s, x)\}$

$\quad x := x - 1; \quad n := m + 2; \quad m := m + 1$

$\quad$ **while** $(m > 0)$ **do**

$\{J(m, n, s, x)\}$

$\quad\quad s := s + 1; \quad m := m - 1$

$\quad$ **end while**

$\quad m := n$

end while

Associate Invariants at loop entry: $I(m, n, s, x)$ and $J(m, n, s, x)$

Verification Conditions:

$I(m, n, s, x) \implies ((J(m, n, s, x)|_m^{m+1})|_n^{m+2})|_x^{x-1}$

$J(m, n, s, x) \implies (J(m, n, s, x)|_m^{m-1})|_s^{s+1}$

$(J(m, n, s, x) \wedge m = 0) \implies I(m, n, s, x)|_m^n$

Initial Condition: $I(0, 0, 0, T)$

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$

while $(x > 0)$ **do**

$\{I(m, n, s, x)\}$

$\quad x := x - 1; \quad n := m + 2; \quad m := m + 1$

$\quad$ **while** $(m > 0)$ **do**

$\{J(m, n, s, x)\}$

$\quad\quad s := s + 1; \quad m := m - 1$

$\quad$ **end while**

$\quad m := n$

end while

Hypothesize: $I(m, n, s, x) : A\ m + B\ n + C\ s + D\ x + E = 0$

$J(m, n, s, x) : F\ m + G\ n + H\ s + K\ x + L = 0$

$I(m, n, s, x) : \quad 2x - 2T + m = 0$

$J(m, n, s, x) : \quad 2x - 2T + n = 0$

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$

while $(x > 0)$ **do**

$\{I(m, n, s, x)\}$

$\quad x := x - 1; \quad n := m + 2; \quad m := m + 1$

$\quad$ **while** $(m > 0)$ **do**

$\{J(m, n, s, x)\}$

$\quad\quad s := s + 1; \quad m := m - 1$

$\quad$ **end while**

$\quad m := n$

end while

Hypothesize: $I(m, n, s, x)$ and $J(m, n, s, x)$ as polynomials of deg. 2.

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$

while $(x > 0)$ **do**

$\{I(m, n, s, x)\}$

$\quad x := x - 1; \quad n := m + 2; \quad m := m + 1$

$\quad$ **while** $(m > 0)$ **do**

$\{J(m, n, s, x)\}$

$\quad\quad s := s + 1; \quad m := m - 1$

$\quad$ **end while**

$\quad m := n$

end while

Hypothesize: $I(m, n, s, x)$ and $J(m, n, s, x)$ as polynomials of deg. 2.

$I(m, n, s, x) : \{2x - 2T + m = 0, \quad s + x^2 - T^2 + m\ x = 0, \quad m^2 - n^2 = 0\}$

$J(m, n, s, x) : \{2x - 2T + n = 0, \quad m + s + x^2 + s\ n - T^2 = 0\}$

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := T$

while $(x > 0)$ **do**

$\{I(m, n, s, x)\}$

$\quad x := x - 1; \quad n := m + 2; \quad m := m + 1$

$\quad$ **while** $(m > 0)$ **do**

$\{J(m, n, s, x)\}$

$\quad\quad s := s + 1; \quad m := m - 1$

$\quad$ **end while**

$\quad m := n$

end while

Hypothesize: $I(m, n, s, x)$ and $J(m, n, s, x)$ as polynomials of deg. 2.

$I(m, n, s, x) : \{2x - 2T + m = 0, \quad s + x^2 - T^2 + m\ x = 0, \quad m^2 - n^2 = 0\}$

$J(m, n, s, x) : \{2x - 2T + n = 0, \quad m + s + x^2 + s\ n - T^2 = 0\}$

$Post : s = T^2$

Example: Invariant expressed using Inequalities

$i := 2; \quad j := 0;$

while b **do**

$\{I(i, j)\}$

if c **then** $\quad i := i + 4$

else $\quad i := i + 2; \quad j := j + 1$

end while

.

,

Example: Invariant expressed using Inequalities

$i := 2; \quad j := 0;$
while b **do**
$\{I(i,j)\}$

if c **then** $\quad i := i + 4$
else $\quad i := i + 2; \quad j := j + 1$
end while

If $I(i,j) = (A\ i + B\ j + C = 0)$ is hypothesized, quantifier elimination gives $A = 0, B = 0, C = 0$ as the values of the parameters, generating the trivial invariant *true*.

Example: Invariant expressed using Inequalities

$i := 2; \quad j := 0;$
while b **do**
$\{I(i,j)\}$

if c **then** $\quad i := i + 4$
else $\quad i := i + 2; \quad j := j + 1$
end while

If $I(i,j) = (A\ i + B\ j + C = 0)$ is hypothesized, quantifier elimination gives $A = 0, B = 0, C = 0$ as the values of the parameters, generating the trivial invariant *true*.

If $I(i,j) = (A\ i + B\ j + C \leq 0)$, the verification condition is:
$\forall i,j : [(A\ i + B\ j + C \leq 0) \Rightarrow [(A\ i + 4A + B\ j + C \leq 0)$

$\wedge (A\ i + 2A + B\ j + B + C \leq 0)]] \wedge (2A + C \leq 0).$

Example: Invariant expressed using Inequalities

$i := 2; \quad j := 0;$
while b **do**
$\{I(i,j)\}$

if c **then** $\quad i := i + 4$
else $\quad i := i + 2; \quad j := j + 1$
end while

If $I(i,j) = (A\ i + B\ j + C = 0)$ is hypothesized, quantifier elimination gives $A = 0, B = 0, C = 0$ as the values of the parameters, generating the trivial invariant *true*.

If $I(i,j) = (A\ i + B\ j + C \leq 0)$, the verification condition is:
$\forall i,j : [(A\ i + B\ j + C \leq 0) \Rightarrow [(A\ i + 4A + B\ j + C \leq 0)$

$\wedge (A\ i + 2A + B\ j + B + C \leq 0)]] \wedge (2A + C \leq 0).$

An equivalent quantifier-free formula is:

$$(C \leq 0 \wedge A = 0 \wedge B \leq 0) \vee (2A + C \leq 0 \wedge A < 0 \wedge 2A + B \leq 0).$$

Example: Invariant expressed using Inequalities

$i := 2; \quad j := 0;$
while b **do**
$\{I(i,j)\}$

if c **then** $\quad i := i + 4$
else $\quad i := i + 2; \quad j := j + 1$
end while

If $I(i,j) = (A\ i + B\ j + C = 0)$ is hypothesized, quantifier elimination gives $A = 0, B = 0, C = 0$ as the values of the parameters, generating the trivial invariant *true*.

If $I(i,j) = (A\ i + B\ j + C \leq 0)$, the verification condition is:
$\forall i,j : [(A\ i + B\ j + C \leq 0) \Rightarrow [(A\ i + 4A + B\ j + C \leq 0)$

$\wedge (A\ i + 2A + B\ j + B + C \leq 0)]] \wedge (2A + C \leq 0).$

An equivalent quantifier-free formula is:

$$(C \leq 0 \wedge A = 0 \wedge B \leq 0) \vee (2A + C \leq 0 \wedge A < 0 \wedge 2A + B \leq 0).$$

Solving for parameters gives

$$(-j \leq 0 \wedge -i + 2j + 2 \leq 0)$$

as an invariant.

Quantifier-Elimination Methods

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)
- Parametric Gröbner Basis Algorithm (Kapur, 1994)
  - (similar to Weispfenning's Comprehensive Gröbner Basis Algorithms)
- Quantifier Elimination Techniques for Real Closed Fields (REDLOG, QEPCAD)
- Other methods

Making the Approach Practical

- Exploiting the structure of verification conditions in quantifier elimination.
- Guessing/hypothesizing parameterized form based on assignment statements.
- Extending the approach for expressing parameterized properties over other data structures.
- Exploiting incomplete specifications.
- Reusing derivations as programs evolve.
- Finding bugs in programs.

More details can be found in

D. Kapur, "Automatically generating loop invariants using quantifier elimination," Proc. *10th International IMACS Conference on Applications of Computer Algebra (ACA 2004),* Lamar, TX, July 2004.

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := A$
while $(x > 0)$ **do**

    $x := x - 1; \quad n := m + 2; \quad m := m + 1$
    while $(m > 0)$ **do**

        $s := s + 1; \quad m := m - 1$
    end while
    $m := n$
end while

Polynomial Invariants Form an Ideal

- States at a program point $\equiv$ set of values variables take

Polynomial Invariants Form an Ideal

- States at a program point $\equiv$ set of values variables take
- Characterize states by a conjunction of polynomial equations.

$$(p_1 = 0 \wedge \cdots \wedge p_k = 0).$$

Polynomial Invariants Form an Ideal

- States at a program point $\equiv$ set of values variables take
- Characterize states by a conjunction of polynomial equations.

$$(p_1 = 0 \wedge \cdots \wedge p_k = 0).$$

The set of values which make the above formula true is the variety associated with the radical ideal of $\{p_1, \cdots, p_k\}$.

Radical Ideal of Invariants

- If $p = 0, q = 0$ are invariants, so are $p + q = 0$ as well as $s\ p = 0$ for any polynomial s.
- if $p^k = 0$ is an invariant, so is $p = 0$.

Radical Ideal of Invariants

- If $p = 0, q = 0$ are invariants, so are $p + q = 0$ as well as $s\ p = 0$ for any polynomial s.
- if $p^k = 0$ is an invariant, so is $p = 0$.

Objective: By Hilbert's Finite Basis Theorem, there is a finite basis of the radical ideal corresponding to program states at a control point. How to construct this ideal and compute its basis?

Computing Polynomial Invariant Ideal

$a := 0; \quad s := 1; \quad t := 1;$

while $(s \leq N)$ **do**

$\quad a := a + 1; \qquad s := s + t + 2; \qquad t := t + 2;$

end while

Computing Polynomial Invariant Ideal

$a := 0; \quad s := 1; \quad t := 1;$

while $(s \leq N)$ **do**

$\quad a := a + 1; \qquad s := s + t + 2; \qquad t := t + 2;$

end while

Program

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

while b **do**

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

end while

Algorithm

$I' := \langle 0 \rangle; I := \langle x_1 - \alpha_1, \cdots, x_m - \alpha_m \rangle;$

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

$\quad I' := I; \qquad I := \bigcap_{n \in \mathbb{N}} I(\bar{x} \leftarrow f^{-n}(\bar{x}))$

end while

f^{-1} above is the inverse of the assignment mapping f.

Eliminate the loop counter n using elimination methods

Ideal-Theoretic Semantics: $x_i := f_i(\bar{x})$

- Input ideal: $\langle p_1, ..., p_k \rangle$
- Output ideal:
  - Want to express in terms of ideals

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

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

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

    - If f_i is invertible, then $\langle p_1(x_i \leftarrow f_i^{-1}(\bar{x})), ..., p_k(x_i \leftarrow f_i^{-1}(\bar{x})) \rangle$.

Example: Program with Nested Loop

$m := 0; \quad n := 0; \quad s := 0; \quad x := A$
while $(x > 0)$ **do**

    $x := x - 1; \quad n := m + 2; \quad m := m + 1$
    while $(m > 0)$ **do**

        $s := s + 1; \quad m := m - 1$
    end while
    $m := n$
end while

Ideal-Theoretic Semantics

Simple Junction Nodes

- Input ideals (one for each path):

  Path 1: $\langle p_{11}, ..., p_{1k_1} \rangle$: $p_{11} = 0 \wedge \cdots \wedge p_{1k_1} = 0$.

  ...

  Path l: $\langle p_{l1}, ..., p_{lk_l} \rangle$: $p_{l1} = 0 \wedge \cdots \wedge p_{lk_l} = 0$.

- Output ideal: Ideal corresponding to $\bigvee_{i=1}^{l} \bigwedge_{j=1}^{k_i} p_{ij} = 0$.

- **Solution:** Take *common* polynomials for all paths $\equiv$

  Compute *intersection* of all input ideals $\bigcap_{i=1}^{l} \langle p_{i1}, ..., p_{ik_i} \rangle$

Computing Polynomial Invariant Ideal

$a := 0; \quad s := 1; \quad t := 1;$

while $(s \leq N)$ **do**

$\quad a := a + 1; \qquad s := s + t + 2; \qquad t := t + 2;$

end while

For the above program, we get:

$$2a - t + 1 = 0, \quad a^2 - at + a + s - t = 0, \quad t^2 - 2a - 4s + 3t = 0$$

Algorithm for Computing Invariants

Program

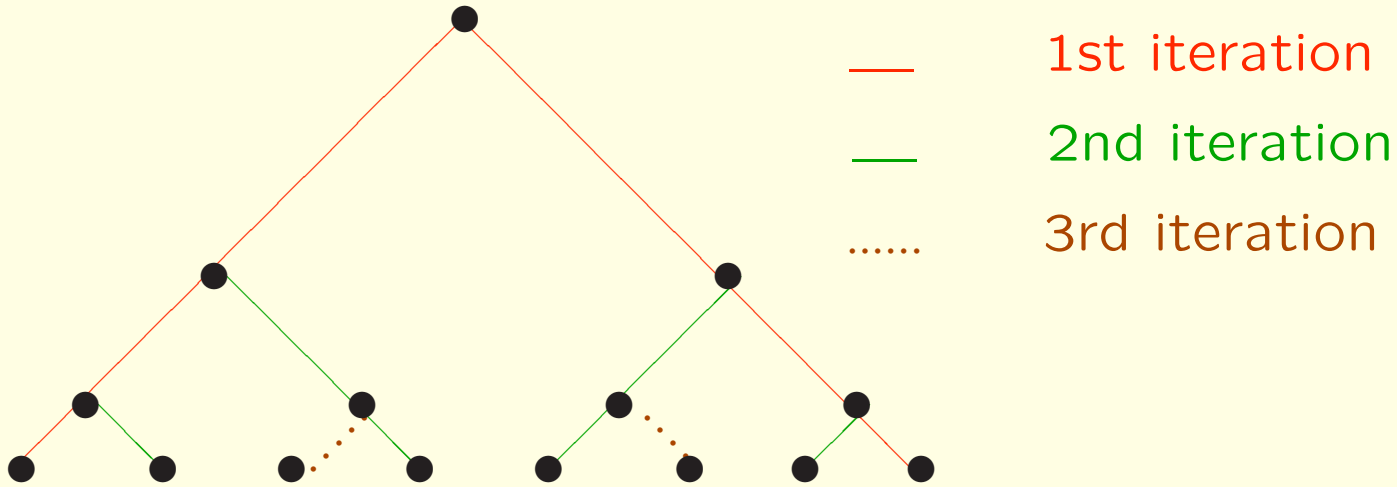
$x := \bar{\alpha}$;
while b **do**
  if c **then** $\bar{x} := f(\bar{x})$
    else
    $\bar{x} := g(\bar{x})$;
end while

Algorithm

$I' := \langle 0 \rangle$; $I := \langle x_1 - \alpha_1, \cdots, x_m - \alpha_m \rangle$;
while $I' \neq I$ **do**
    $I' := I$;
    $I := \bigcap_{n \in \mathbb{N}} [\, I(\bar{x} \leftarrow f^{-n}(\bar{x}))$
        $\bigcap I(\bar{x} \leftarrow g^{-n}(\bar{x}))\,]$;
end while

Algorithm for Computing Invariants

- After N iterations: $I \equiv$ intersection of Ideals for all paths with $\leq N - 1$ alternations



- I stabilizes: Termination in $O(m)$ iterations under certain conditions on assignment statements, where $m =$ number of variables

Main Result

THEOREM. In a loop with assignments $\bar{x} := f_i(\bar{x})$, if tests are ignored and each f_i is a solvable mapping with positive rational eigenvalues, the algorithm computes the strongest invariant in at most $2m + 1$ steps, where m is the number of program variables in the loop.

,

Main Result

THEOREM. In a loop with assignments $\bar{x} := f_i(\bar{x})$, if tests are ignored and each f_i is a solvable mapping with positive rational eigenvalues, the algorithm computes the strongest invariant in at most $2m + 1$ steps, where m is the number of program variables in the loop.

Moreover, if the assignment mappings commute, i.e. $f_i \circ f_j = f_j \circ f_i$ for $1 \leq i, j \leq n$, then the algorithm terminates in at most $n + 1$ steps, where n is the number of branches in the loop.

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

More details can be found in

E. Rodríguez-Carbonell and D. Kapur, "Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations," Proc. *International Symposium on Symbolic and Algebriac Computation (ISSAC-2004)*, July 2004, Santander Spain

Abstract Semantics
Entry Node

- Input ideal: none
- Output ideal:
  - If no precondition available $\longrightarrow \{0\}$
  - If precondition $p_1 = 0 \wedge \cdots \wedge p_k = 0$ available $\longrightarrow \mathbf{IV}(p_1, ..., p_k)$
- $\mathbf{IV}(I) \equiv$ biggest set of polynomials with same zeroes as I

Ideal-Theoretic Semantics

Assignments: $x_i := f(\bar{x})$

- 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

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

Ideal-Theoretic Semantics

Tests: $q = 0$

- Input ideal: $\langle p_1, ..., p_k \rangle$
- Output ideal: (*true* path): $p_1 = 0 \wedge \cdots \wedge p_k = 0 \wedge q = 0$
- **Solution:** Add q to list of generators of input ideal: $\mathbf{IV}(p_1, \cdots, p_k, q)$.
- Output ideal: (*false* path): $p_1 = 0 \wedge \cdots \wedge p_k = 0 \wedge q \neq 0$
- **Solution:**
  - Zeroes of the **quotient ideal** $\langle p_1, ..., p_k \rangle : \langle q \rangle \equiv$ zeroes of $\langle p_1, ..., p_k \rangle$ \ zeroes of $\langle q \rangle$

Ideal-Theoretic Semantics

Simple Junction Nodes

- Input ideals (one for each path):

  Path 1: $\langle p_{11}, ..., p_{1k_1} \rangle$: $p_{11} = 0 \wedge \cdots \wedge p_{1k_1} = 0$.

  ...

  Path l: $\langle p_{l1}, ..., p_{lk_l} \rangle$: $p_{l1} = 0 \wedge \cdots \wedge p_{lk_l} = 0$.

- Output ideal: Ideal corresponding to $\bigvee_{i=1}^{l} \bigwedge_{j=1}^{k_i} p_{ij} = 0$.

- **Solution:** Take *common* polynomials for all paths $\equiv$

  Compute *intersection* of all input ideals $\bigcap_{i=1}^{l} \langle p_{i1}, ..., p_{ik_i} \rangle$

Ideal-Theoretic Semantics

Loop Junction Nodes

- Input ideals: $J_1, \cdots, J_l$
- Previous output ideal: I
- Output ideal: $I \cap (\bigcap_{i=1}^{l} J_i)$ (as with simple junction nodes)
- **Problem:** Non-termination of forward propagation !

$x := 0$;
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: $\langle x(x-1)(x-2) \rangle$
- ...

Solution: WIDENING $\longrightarrow$ bounding degree

Abstract Interpretation Approach

Widening Operator

- Parametric widening $I \nabla_d J$: Based on taking polynomials of $I \cap J$ of degree $\leq d$
- Uses **Gröbner bases:** $I \nabla_d J = \mathbf{IV}(\{p \in GB(I \cap J) \mid \deg(p) \leq d\})$
  - Polynomials of degree $\leq d$ of $GB(I)$ reduce polynomials of degree $\leq d$ in I to 0

- **THEOREM**. If tests are ignored and assignments are linear, forward propagation based on ideal-theoretic semantics 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
    - Termination based on the fact that the dimension of the associated vector spaces cannot keep decreasing

Table of Examples

PROGRAM	COMPUTING	d	VARS	IF'S	LOOPS	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

PC Linux Pentium 4 2.5 Ghz

More details can be found in

E. Rodríguez-Carbonell and D. Kapur, "An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants," Proc. *11th Static Analysis Symposium (SAS-2004)*, September 2004, Verona, Italy.

Key Results from Polynomial Ideal Theory

- **Hilbert's Finite Basis Theorem:** Every polynomial ideal over a Noetherian ring has a finite basis.
- Every variety can be decomposed into **finitely many irreducible components**.
- Intersection of ideals, quotient of an ideal and elimination ideals can be computed algorithmically (using **Gröbner basis** algorithm).
- Primality and radicality of an ideal are preserved under polynomial substitutions.
- Every finite-dimensional vector space has a finite basis.
- Solvable mappings with positive eigen values increase the dimension of one of the components in an iireducible decomposition of a variety.

Related Work

Work	Restrictions	Nesting	Conditions	Complete
[MOS04]	bounded degree	yes	no	yes
[SSM04]	prefixed form	yes	yes	no
[MOS04]	prefixed form	yes	yes	yes
[RCK04]	no restriction	no	no	yes
[RCK04]	bounded degree	yes	yes	yes
[Kapur04]	prefixed form	yes	yes	??

- Karr (1976): *linear equalities*
- Cousot, Halbwachs (1978): *linear inequalities*
- Colón, Sankaranarayanan, Sipma (2003): *linear inequalities*
- Müller-Olm, Seidl (2004): *polynomial equalities of bounded degree*
- Sankaranarayanan et al (2004): *polynomial equalities of prefixed form*
- Müller-Olm, Seidl (2004): *polynomial equalities of prefixed form*
- Rodríguez-Carbonell, Kapur (ISSAC-2004;SAS-2004): *polynomial equalities*

Conclusions

- **Algebraic Geometry and Real Geometry** are powerful tools for automatically generating invariants expressed using **polynomials**.
- Termination of fixed point computation is based on
  - first algorithm: Ideals of bigger dimension in every iteration are computed, ($O(m)$ iterations where m is the number of changing program variables).
  - second algorithm: Polynomials of bounded degree constitute a finite vector space with a finite basis.
- A **general framework** for generating invariants
  using fixed-point computations and quantifier elimination
  - How to develop rich theories for other data structures?