Can Algebraic Geometry Rescue Program Verification?

Deepak Kapur

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

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, Journal of Systems Sciences and Complexity-2006).

   b) Ideal-Theoretic Methods based on Fixed Point Computations (ISSAC-2004, SAS-2004, ICTAC-2004, Science of Programming-2007, Journal of Symbolic Computation-2007) – with Enric Rodríguez-Carbonell

3. Experimental Results
4. Role of Computational Logic and Algebra
5. How could these results be generalized?

Program Verification/Formal Methods/Static Analysis

- 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/Formal Methods/Static Analysis

- 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. $\{P\} \ \ Prog\_Frag \ \ \{Q\}$

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. $\{P\} \ \ Prog\_Frag \ \ \{Q\}$

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. $\{P\} \ \ Prog\_Frag \ \ \{Q\}$

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: $x * y + z = M * N$.

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.

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

Program states (i.e., the set of values which make the above formula true) can be characterized by the algebraic variety associated with the radical ideal of $\{p_1, \cdots, p_k\}$.

Radical Ideal of Invariants

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

Radical Ideal of Invariants

- If $p = 0, q = 0$ are invariants, $p + q = 0$ is also an invariant.
- If $p = 0$ is an invariant, for any polynomial s, $s\ p = 0$ is also an invariant.
- 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?

Expressiveness

- $p = 0 \vee q = 0$ is equivalent to $p * q = 0$.

Expressiveness

- $p = 0 \vee q = 0$ is equivalent to $p * q = 0$.
- $p \neq 0$ is equivalent to $p * z - 1 = 0$.

Expressiveness

- $p = 0 \vee q = 0$ is equivalent to $p * q = 0$.
- $p \neq 0$ is equivalent to $p * z - 1 = 0$.
- $0 \leq p \leq 2$ is equivalent to $p * (p - 1) * (p - 2) = 0$.

Ideal-Theoretic Methods

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

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

while b **do**

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

end while

Algorithm

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

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

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

end while

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

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

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:

$$t = 2a + 1, \quad s = (a + 1)^2.$$

Invariant generation without a priori bound on degree

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 rational eigenvalues, the algorithm computes the strongest invariant in at most $m + 1$ steps, where m is the number of program variables in the loop.

,

Invariant generation without a priori bound on degree

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 rational eigenvalues, the algorithm computes the strongest invariant in at most $m + 1$ 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.

Invariant generation without a priori bound on degree

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 rational eigenvalues, the algorithm computes the strongest invariant in at most $m + 1$ 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.

The above is true when a loop body does not have any conditionals; the method terminates in a single iteration.

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, 2004

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 Algebraic Computation (ISSAC-2004)*, July 2004, Santander Spain

An expanded version is in the *Journal of Symbolic Computation*

Key Results from Polynomial Ideal Theory

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

Quantifier Elimination Techniques

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.

Soundness and Completeness

- 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 not exist. Invariants can be guaranteed not to exist if no approximations are made, while generating verification conditions.
  - If all assignments making the formula true can be finitely described, invariants generated may be the strongest of the hypothesized form. Invariants generated are guaranteed to be the strongest if no approximations are made, while generating verification conditions.

Example: Termination of Multiplication

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

Similar to generating loop invariants, hypothesize a termination function $T(x, y, z)$

Termination Conditions:

$$\exists DC1 \geq 0, DC2 \geq 0, \forall x, y, z: \quad [(\ y\ mod\ 2 = 0 \Rightarrow$$
$$((T(x,y,z) - T(2x, y\ div\ 2, z)) \geq DC1 \wedge T(2x, y\ div\ 2, z) \geq 0] \wedge$$
$$[(\ y\ mod\ 2 \neq 0 \Rightarrow ((T(x,y,z) - T(2x, (y-1)\ div\ 2, x+z)) \geq DC2 \wedge$$
$$T(2x, (y-1)\ div\ 2, x+z) \geq 0)] \wedge$$
$$T(M, N, 0) \geq 0$$

Are there values of parameters in $T(x, y, z)$ such that those values make the above formula valid?

Example: Square Root Program

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

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

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

end while

An example of Multiple Invariants

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

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

$\{I(a,s,t) =$

$(u_1\, a^2 + u_2\, s^2 + u_3\, t^2 + u_4\, as + u_5\, at + u_6\, st + u_7\, a + u_8\, s + u_9\, t + u_{10} = 0)\}$

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

end while

An example of Multiple Invariants

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

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

$\{I(a,s,t) =$

$(u_1\, a^2 + u_2\, s^2 + u_3\, t^2 + u_4\, as + u_5\, at + u_6\, st + u_7\, a + u_8\, s + u_9\, t + u_{10} = 0)\}$

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

end while

Quantifier elimination on the verification condition gives:

$u_1 = -u_5, \;\; u_7 = -2u_3 - u_5 + 2u_{10}, \;\; u_8 = -4u_3 - u_5, \;\; u_9 = 3u_3 + u_5 - u_{10}$

An example of Multiple Invariants

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

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

$\{I(a,s,t) =$

$(u_1\ a^2 + u_2\ s^2 + u_3\ t^2 + u_4\ as + u_5\ at + u_6\ st + u_7\ a + u_8\ s + u_9\ t + u_{10} = 0)\}$

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

end while

Quantifier elimination on the verification condition gives:

$$u_1 = -u_5, \quad u_7 = -2u_3 - u_5 + 2u_{10}, \quad u_8 = -4u_3 - u_5, \quad u_9 = 3u_3 + u_5 - u_{10}$$

Making exactly one of u_5, u_3, u_{10} to be 1, and other parameters to be 0, the following independent invariants are generated:

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

Example: Invariant expressed using Inequalities
(Cousot&Halbwachs, POPL (1978))

$i := 2; \quad j := 0;$
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
(Cousot&Halbwachs, POPL (1978))

$i := 2; \quad j := 0;$
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
(Cousot&Halbwachs, POPL (1978))

$i := 2; \quad j := 0;$
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 (Cousot&Halbwachs, POPL (1978))

$i := 2; \quad j := 0;$
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
(Cousot&Halbwachs, POPL (1978))

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

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

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

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

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

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: Exponentiation

$x := A; \quad y := B; \quad z := 1;$

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

if $odd(y)$ **then** $\quad y := y - 1; \qquad z := x * z;$

else $\quad x := x * x; \qquad y := y/2;$

end while

Example: Exponentiation

$x := A; \quad y := B; \quad z := 1;$

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

if $odd(y)$ **then** $\quad y := y - 1; \qquad z := x * z;$

else $\quad x := x * x; \qquad y := y/2;$

end while

$$I(x, y, z) : (x^y) * z = A^B$$

Example: Exponentiation

$x := A; \quad y := B; \quad z := 1;$

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

if $odd(y)$ **then** $\quad y := y - 1; \qquad z := x * z;$

else $\quad x := x * x; \qquad y := y/2;$

end while

$$I(x, y, z) : (x^y) * z = A^B$$

$$I'(x, y, z) : (y * log(x) + log(z)) = B * log(A)$$

Example: Combination of Theories (Gulwani and Tiwari, 2007)

$a1 := 0; \quad a2 := 0; \quad b1 := 1; \quad b2 := f(1);$

$c1 := 2; \quad c2 := 2; \quad d1 := 3; \quad d2 := f(4);$

while $(b1 \neq b2)$ **do**

$a1 := a1 + 1; \quad a2 := a2 + 2; \quad b1 := f(b1); \quad b2 := f(b2);$

$c1 := f(2c1 - c2); \quad c2 := f(c2); \quad d1 := f(1 + d1); \quad d2 := f(d2 + 1)$

end while

Example: Combination of Theories (Gulwani and Tiwari, 2007)

$a1 := 0; \quad a2 := 0; \quad b1 := 1; \quad b2 := f(1);$
$c1 := 2; \quad c2 := 2; \quad d1 := 3; \quad d2 := f(4);$
while $(b1 \neq b2)$ **do**
$\quad a1 := a1 + 1; \quad a2 := a2 + 2; \quad b1 := f(b1); \quad b2 := f(b2);$
$\quad c1 := f(2c1 - c2); \quad c2 := f(c2); \quad d1 := f(1 + d1); \quad d2 := f(d2 + 1)$
end while

$I : (A\ a1 + B\ a2 + C\ b1 + D\ b2 + E\ c1 + F\ c2 + H\ d1 + J\ d2 + K +$
$\quad L\ f(A'\ a1 + B'\ a2 + C'\ b1 + D'\ b2 + E'\ c1 + F'\ c2 + H'\ d1 + J'\ d2 + K')) = 0.$

Example: Combination of Theories (Gulwani and Tiwari, 2007)

$a1 := 0; \quad a2 := 0; \quad b1 := 1; \quad b2 := f(1);$
$c1 := 2; \quad c2 := 2; \quad d1 := 3; \quad d2 := f(4);$
while $(b1 \neq b2)$ **do**
$\quad a1 := a1 + 1; \quad a2 := a2 + 2; \quad b1 := f(b1); \quad b2 := f(b2);$
$\quad c1 := f(2c1 - c2); \quad c2 := f(c2); \quad d1 := f(1 + d1); \quad d2 := f(d2 + 1)$
end while

$I : (A\ a1 + B\ a2 + C\ b1 + D\ b2 + E\ c1 + F\ c2 + H\ d1 + J\ d2 + K +$
$\quad L\ f(A'\ a1 + B'\ a2 + C'\ b1 + D'\ b2 + E'\ c1 + F'\ c2 + H'\ d1 + J'\ d2 + K')) = 0.$

Generate verification conditions:

$I_0 : C + Df(b1) + E2 + F2 + H3 + Jf(4) + K +$
$\quad L\ f(C' + D'f(b1) + E'2 + F'2 + H'3 + J'f(4) + K' = 0$

$VC : (I(a1, a2, b1, b2, c1, c2, d1, d2) \Rightarrow$
$I(a1 + 1, a2 + 2, f(b1), f(b2), f(2c1 - c2), f(c2), f(1 + d1), f(d2 + 1)).$

Example: Combination of Theories

$$I : (A\ a1 + B\ a2 + C\ b1 + D\ b2 + E\ c1 + F\ c2 + H\ d1 + J\ d2 + K + L\ f(A'\ a1 + B'\ a2 + C'\ b1 + D'\ b2 + E'\ c1 + F'\ c2 + H'\ d1 + J'\ d2 + K')) = 0.$$

$$VC : (I(a1, a2, b1, b2, c1, c2, d1, d2) \Rightarrow$$
$$I(a1 + 1, a2 + 2, f(b1), f(b2), f(2c1 - c2), f(c2), f(1 + d1), f(d2 + 1)).$$

Solve these constraints over the combined theories of Presburger Arithmetic and Theory of Equality with Uninterpreted Symbols.

Example: Combination of Theories

$$I : (A\ a1 + B\ a2 + C\ b1 + D\ b2 + E\ c1 + F\ c2 + H\ d1 + J\ d2 + K + L\ f(A'\ a1 + B'\ a2 + C'\ b1 + D'\ b2 + E'\ c1 + F'\ c2 + H'\ d1 + J'\ d2 + K')) = 0.$$

$$VC : (I(a1, a2, b1, b2, c1, c2, d1, d2) \Rightarrow$$
$$I(a1 + 1, a2 + 2, f(b1), f(b2), f(2c1 - c2), f(c2), f(1 + d1), f(d2 + 1)).$$

Solve these constraints over the combined theories of Presburger Arithmetic and Theory of Equality with Uninterpreted Symbols.

Quantifier-Elimination

Example: Combination of Theories

$$I : (A\ a1 + B\ a2 + C\ b1 + D\ b2 + E\ c1 + F\ c2 + H\ d1 + J\ d2 + K + L\ f(A'\ a1 + B'\ a2 + C'\ b1 + D'\ b2 + E'\ c1 + F'\ c2 + H'\ d1 + J'\ d2 + K')) = 0.$$

$$VC : (I(a1, a2, b1, b2, c1, c2, d1, d2) \Rightarrow$$
$$I(a1 + 1, a2 + 2, f(b1), f(b2), f(2c1 - c2), f(c2), f(1 + d1), f(d2 + 1)).$$

Solve these constraints over the combined theories of Presburger Arithmetic and Theory of Equality with Uninterpreted Symbols.

Quantifier-Elimination

- Purify by introducing new symbols and obtaining formulas over individual theories.
- Identify symbols to be made equivalent (a la Nelson and Oppen combination framework).
- Find values of parameters making the verification conditions valid.
- Each such combination (equivalence class of symbols and parameter values) gives an invariant.

Example: Combination of Theories

For the above example, we get,

All parameter values 0 except $A = 2, B = -1$: $2a1 - a2 = 0$.

Example: Combination of Theories

For the above example, we get,
All parameter values 0 except $A = 2, B = -1$: $2a1 - a2 = 0$.
All parameter values 0 except $D = 1, L = -1, C' = 1$: $b2 - f(b1) = 0$.

Example: Combination of Theories

For the above example, we get,

All parameter values 0 except $A = 2, B = -1$: $2a1 - a2 = 0$.

All parameter values 0 except $D = 1, L = -1, C' = 1$: $b2 - f(b1) = 0$.

All parameter values 0 except $E = 1, F = -1$: $c1 - c2 = 0$.

Example: Combination of Theories

For the above example, we get,
All parameter values 0 except $A = 2, B = -1$: $2a1 - a2 = 0$.
All parameter values 0 except $D = 1, L = -1, C' = 1$: $b2 - f(b1) = 0$.
All parameter values 0 except $E = 1, F = -1$: $c1 - c2 = 0$.
All parameter values 0 except $J = 1, L = -1, H' = 1, K' = 1$:
$d2 - f(d1 + 1) = 0$.

Example: Combination of Theories

For the above example, we get,
All parameter values 0 except $A = 2, B = -1$: $2a1 - a2 = 0$.
All parameter values 0 except $D = 1, L = -1, C' = 1$: $b2 - f(b1) = 0$.
All parameter values 0 except $E = 1, F = -1$: $c1 - c2 = 0$.
All parameter values 0 except $J = 1, L = -1, H' = 1, K' = 1$:
$d2 - f(d1 + 1) = 0$.
Invariants are: $2a1 = a2, \quad b2 = f(b1), \quad c1 = c2, \quad d2 = f(d1 + 1), \cdots$.

Quantifier-Elimination/Constraint Solving

- Generalized Presburger Arithmetic (for invariants expressed using linear inequalities)
- Polynomials over an algebraic closed field: Parametric Gröbner Basis Algorithm (Kapur, 1994)
  - (similar to Weispfenning's Comprehensive Gröbner Basis Algorithms)
- Quantifier Elimination Techniques for Real Closed Fields (REDLOG, QEPCAD)
- Combination of Theories–Presburger Arithmetic with Theory of Equality over Uninterpreted Symbols (Shostak, 1979; Nelson, 1981), and with Boolean Algebra (Kuncak, 2007), etc.
- Reduction Approach to Decision Procedures for Theories over Abstract Data Structures, including Finite Lists, Finite Sets, Finite Arrays, Finite Multisets (Kapur and Zarba, 2005).

Making the Approach Practical

- Exploiting the structure of verification conditions in quantifier elimination.
  - Seidl et al's work on modular arithmetic, Herbrand equalities, polynomial relations, where restrictions are imposed on the kind of assignment statements are handled so that invariants can be derived using algorithms in linear algebra, vector spaces, R-module.
- 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.

An expanded version has appeared in *Journal of Systems Science and Complexity, 2006*

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 and Future Work

- **Algebraic Geometry and Real Geometry** are powerful tools to be used for automatically generating invariants expressed using **polynomials**.
- We have proposed a **general framework** for generating invariants using fixed-point computations on polynomial ideals, solving recurrence relations, and quantifier elimination.
- Results extend to other data structures using
  - **combination of theories** for combining the above theories over numbers with theory of equality with uninterpreted symbols– (Shostak, 1979; Nelson, 1981), and (Kapur, 1997); for Boolean Algebra (Kuncak, 2007), and
  - the **reduction approach to decision procedures** for theories of finite sets, finite multisets, finite arrays, and finite lists –(Kapur and Zarba, 2005).
- Need to develop even **richer theories** for data structures (Bradley et al, 2006).