Automated Generation of Polynomial Invariants for Imperative Program Verification in *Theorema*

Laura Kovács, Tudor Jebelean

Research Institute for Symbolic Computation, Linz

e-Austria Institute, Timişoara

{kovacs,jebelean}@risc.uni-linz.ac.at

Joint work with:
D. Kapur (Univ. of New Mexico)

Outline

Program Verification

The *Theorema* System

Imperative Program Verification in *Theorema*

- P-solvable Imperative Loops
- Automatized Invariant Generation

Conclusions

Outline

Program Verification

The *Theorema* System

Imperative Program Verification in *Theorema*

- P-solvable Imperative Loops
- Automatized Invariant Generation

Conclusions

Program Verification

Rule–based Programming **Theorema**

Specifications, programs and verification can be viewed in a uniform framework (higher–order predicate logic)

- (consequence) verification: checking that each clause is true.

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning
  1. *Predicate Transformer (weakest precondition)* [Dijkstra76, Gries81]

Program Verification

Rule–based Programming **Theorema**

Specifications, programs and verification can be viewed in a uniform framework (higher–order predicate logic)

- (consequence) verification: checking that each clause is true.

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning

  1. *Predicate Transformer (weakest precondition)* [Dijkstra76, Gries81]

Program Verification

Rule–based Programming **Theorema**

Specifications, programs and verification can be viewed in a uniform framework (higher–order predicate logic)

- (consequence) verification: checking that each clause is true.

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning
  1. *Predicate Transformer (weakest precondition)* [Dijkstra76, Gries81]

Program Verification

Rule–based Programming **Theorema** → **B.Buchberger, A.Crăciun, N.Popov, T.Jebelean**

Specifications, programs and verification can be viewed in a uniform framework (higher–order predicate logic)

- (consequence) verification: checking that each clause is true.

Imperative Programming **Theorema**→ **L.Kovács, T.Jebelean**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning
  1. *Predicate Transformer (weakest precondition)* [Dijkstra76, Gries81]

Outline

Program Verification

The *Theorema* System

Imperative Program Verification in *Theorema*
- P-solvable Imperative Loops
- Automatized Invariant Generation

Conclusions

The *Theorema* System

Theorema : A computer aided mathematical assistant

- Proving / Computing / Solving

  using: specified “knowledge bases”

  applying: provers, simplifiers and solvers from the *Theorema* library

- Composing / Structuring / Manipulating mathematical texts

- Advantages of Program Verification in *Theorema* :

  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- { Proving, Computing, Solving }
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- { Composing, Structuring, Manipulating } mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- Proving / Computing / Solving
  using: specified “knowledge bases”
  applying: provers, simplifiers and solvers from the *Theorema* library
- Composing / Structuring / Manipulating mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- { Proving / Computing / Solving }
  using: specified “knowledge bases”
  applying: provers, simplifiers and solvers from the *Theorema* library
- { Composing / Structuring / Manipulating } mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- { Proving / Computing / Solving }
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- { Composing / Structuring / Manipulating } mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- { Proving / Computing / Solving }
  using: specified “knowledge bases”
  applying: provers, simplifiers and solvers from the *Theorema* library
- { Composing / Structuring / Manipulating } mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- { Proving, Computing, Solving }
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- { Composing, Structuring, Manipulating } mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

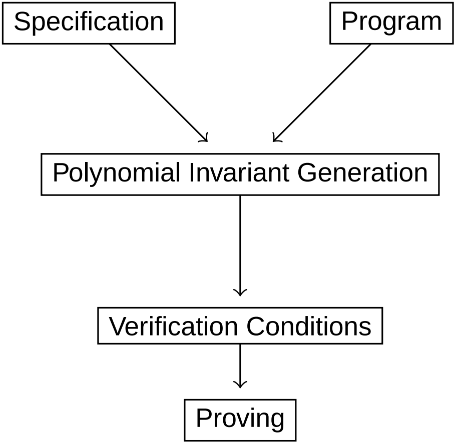
The *Theorema* System

Theorema : A computer aided mathematical assistant

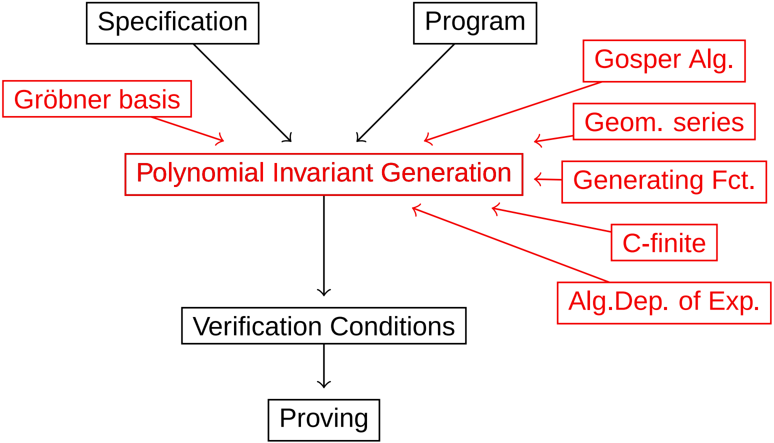
- Proving / Computing / Solving
  using: specified “knowledge bases”
  applying: provers, simplifiers and solvers from the *Theorema* library
- Composing / Structuring / Manipulating mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and using natural style inference
  2. access to powerful computing and solving algorithms (Mathematica)

Outline

Imperative Program Verification in *Theorema*



Imperative Program Verification in *Theorema*



Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
  1. Recurrence Solving: find closed forms of the loop variables (→ compute algebraic dependencies of exponential sequences);
  2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables → all poly invariants;
- Loops with assignments and with/without conditionals. Assignments are:
  - Non–mutual recurrences:
    ◇ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
    ◇ geometric series: $x(k+1) = c * x(k)$;
    ◇ C-finite:
    $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
  - Mutual recurrences: generating functions;
- Implementation successfully applied to many programs working on numbers.

Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
  1. Recurrence Solving: find closed forms of the loop variables ($\rightarrow$ compute algebraic dependencies of exponential sequences);
  2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables $\rightarrow$ all poly invariants;

- Loops with assignments and with/without conditionals. Assignments are:
  - Non–mutual recurrences:
    ◇ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
    ◇ geometric series: $x(k+1) = c * x(k)$;
    ◇ C-finite:
    $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
  - Mutual recurrences: generating functions;

- Implementation successfully applied to many programs working on numbers.

Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
  1. Recurrence Solving: find closed forms of the loop variables ($\rightarrow$ compute algebraic dependencies of exponential sequences);
  2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables $\rightarrow$ all poly invariants;
- Loops with assignments and with/without conditionals. Assignments are:
  - Non–mutual recurrences:
    ◇ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
    ◇ geometric series: $x(k+1) = c * x(k)$;
    ◇ C-finite:
    $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
  - Mutual recurrences: generating functions;
- Implementation successfully applied to many programs working on numbers.

Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
    1. Recurrence Solving: find closed forms of the loop variables (→ compute algebraic dependencies of exponential sequences);
    2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables → all poly invariants;
- Loops with assignments and with/without conditionals. Assignments are:
    - Non–mutual recurrences:
      ◇ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
      ◇ geometric series: $x(k+1) = c * x(k)$;
      ◇ C-finite:
      $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
    - Mutual recurrences: generating functions;
- Implementation successfully applied to many programs working on numbers.

Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
  1. Recurrence Solving: find closed forms of the loop variables ($\rightarrow$ compute algebraic dependencies of exponential sequences);
  2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables $\rightarrow$ all poly invariants;
- Loops with assignments and with/without conditionals. Assignments are:
  - Non–mutual recurrences:
    ◇ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
    ◇ geometric series: $x(k+1) = c * x(k)$;
    ◇ C-finite:
    $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
  - Mutual recurrences: generating functions;
- Implementation successfully applied to many programs working on numbers.

Overview of our Method - using Algebraic Techniques

- Based on the *difference equations method* [ElspasGreen72]:
    1. Recurrence Solving: find closed forms of the loop variables ($\rightarrow$ compute algebraic dependencies of exponential sequences);
    2. Polynomial Eq. Generation: variable elimination by Gröbner basis generators of the ideal of valid polynomials relations among loop variables $\rightarrow$ all poly invariants;
- Loops with assignments and with/without conditionals. Assignments are:
    - Non–mutual recurrences:
      $\diamond$ Gosper–summable: $x(k+1) = x(k) + h(k+1)$, where $h(k+1)$ is a hypergeometric term;
      $\diamond$ geometric series: $x(k+1) = c * x(k)$;
      $\diamond$ C-finite:
      $x(k+d) = c_{d-1} * x(k+d-1) + \ldots + c_1 * x(k+1) + c_0 * x(k) + f(k)$;
    - Mutual recurrences: generating functions;
- Implementation successfully applied to many programs working on numbers.

Algebraic Dependencies among Exponential Sequences

Let $\theta_1, \ldots, \theta_s \in \bar{\mathbb{K}}$, and their exponential sequences $\theta_1^n, \ldots, \theta_s^n \in \bar{\mathbb{K}}$.

An algebraic dependency of these sequences is a polynomial p :

$$p(\theta_1^n, \ldots, \theta_s^n) = 0, \quad (\forall n \geq 1).$$

Example

- The algebraic dependency among the exponential sequences of $\theta_1 = 2$ and $\theta_2 = 4$ is:

$$\theta_1^{2n} - \theta_2^n = 0$$

- There is no algebraic dependency among the exponential sequences of $\theta_1 = 2$ and $\theta_2 = 3$.

Algebraic Dependencies among Exponential Sequences

Let $\theta_1, \ldots, \theta_s \in \bar{\mathbb{K}}$, and their exponential sequences $\theta_1^n, \ldots, \theta_s^n \in \bar{\mathbb{K}}$.

An algebraic dependency of these sequences is a polynomial p :

$$p(\theta_1^n, \ldots, \theta_s^n) = 0, \quad (\forall n \geq 1).$$

Example

- The algebraic dependency among the exponential sequences of $\theta_1 = 2$ and $\theta_2 = 4$ is:
$$\theta_1^{2n} - \theta_2^n = 0$$
- There is no algebraic dependency among the exponential sequences of $\theta_1 = 2$ and $\theta_2 = 3$.

P-solvable Imperative Loops

The recursively changed variables $x_1, \ldots, x_m$ have their closed forms of the following nature:

$$\begin{cases} x_1(n) & = & p_{1,1}(n)\theta_1^n + \cdots + p_{1,s}(n)\theta_s^n \\ x_2(n) & = & p_{2,1}(n)\theta_1^n + \cdots + p_{2,s}(n)\theta_s^n \\ & \vdots & \\ x_m(n) & = & p_{m,1}(n)\theta_1^n + \cdots + p_{m,s}(n)\theta_s^n \end{cases},$$

where:

1. n is the loop counter;
2. $x_i(n)$ $(1 \leq i \leq m)$ represent the value of x_i at iteration n;
3. $p_{1,1}, \ldots, p_{1,s}, \ldots\ldots, p_{m,1}, \ldots, p_{m,s} \in \mathbb{K}[n]$;
4. $\theta_1, \ldots, \theta_s \in \bar{\mathbb{K}}$;
5. there exist algebraic dependencies among $\theta_1^n, \ldots, \theta_s^n$.

P-solvable Imperative Loops

The recursively changed variables $x_1, \ldots, x_m$ have their closed forms of the following nature:

$$\begin{cases} x_1(n) & = & q_1(n, \theta_1^n, \ldots, \theta_s^n) \\ x_2(n) & = & q_2(n, \theta_1^n, \ldots, \theta_s^n) \\ & \vdots & \\ x_m(n) & = & q_m(n, \theta_1^n, \ldots, \theta_s^n) \end{cases},$$

where:

- there exist algebraic dependencies among $\theta_1^n, \ldots, \theta_s^n$.

Invariant Generation for Loops with Conditionals

Example: Program for Computing Square Roots, by K. Zuse

Specification Specification["SqrtZuse", SqrtZuse[$\downarrow a, \downarrow err, \uparrow q$],

Pre $\rightarrow (a \geq 1) \wedge (err > 0)$,

Post $\rightarrow (q^2 \leq a) \wedge (a < q^2 + err)$)]

Program

Invariant Generation for Loops with Conditionals

Example: Program for Computing Square Roots, by K. Zuse

Specification Specification[“SqrtZuse”, SqrtZuse[$\downarrow a, \downarrow err, \uparrow q$],

Pre $\rightarrow (a \geq 1) \wedge (err > 0)$,

Post $\rightarrow (q^2 \leq a) \wedge (a < q^2 + err)$)]

Program

Invariant Generation for Loops with Conditionals

Example: Program for Computing Square Roots, by K. Zuse

Specification Specification[“SqrtZuse”, SqrtZuse[$\downarrow a, \downarrow err, \uparrow q$],

Pre $\rightarrow (a \geq 1) \wedge (err > 0)$,

Post $\rightarrow (q^2 \leq a) \wedge (a < q^2 + err)$)]

Program

Invariant Generation for Loops with Conditionals

Example: Program for Computing Square Roots, by K. Zuse

Specification

Specification["SqrtZuse", SqrtZuse[$\downarrow a, \downarrow err, \uparrow q$],
Pre $\rightarrow (a \geq 1) \wedge (err > 0)$,
Post $\rightarrow (q^2 \leq a) \wedge (a < q^2 + err)$)]

Program

Program["SqrtZuse", SqrtZuse[$\downarrow a, \downarrow err, \uparrow q$],
Module[$\{r, p\}$,
$r := a - 1;\ q := 1;\ p := 1/2;$
While[$(2 * p * r \geq err)$,
If[$2 * r - 2 * q * p \geq 0$
Then $r := 2 * r - 2 * q - p;\ q := q + p;\ p := p/2$,
Else $r := 2 * r;\ p := p/2$]]]]

Invariant Generation for Loops with Conditionals

Example: Program for Computing Square Roots, by K. Zuse

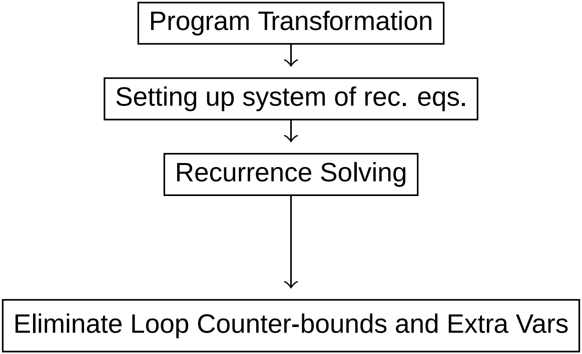
Specification

$\text{Specification}[\text{“SqrtZuse”}, \text{SqrtZuse}[\downarrow a, \downarrow err, \uparrow q],$
$\quad \text{Pre} \rightarrow (a \geq 1) \wedge (err > 0),$
$\quad \text{Post} \rightarrow (q^2 \leq a) \wedge (a < q^2 + err))]$

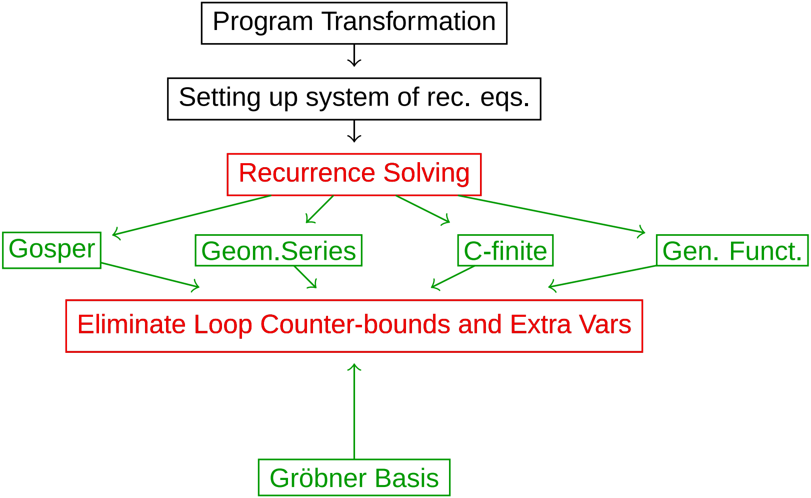
Program

$\text{Program}[\text{“SqrtZuse”}, \text{SqrtZuse}[\downarrow a, \downarrow err, \uparrow q],$
$\quad \text{Module}[\{r, p\},$
$\qquad r := a - 1;\ q := 1;\ p := 1/2;$
$\qquad \text{While}[(2 * p * r \geq err),$
$\qquad\quad \text{If}[2 * r - 2 * q * p \geq 0$
$\qquad\qquad \text{Then } r := 2 * r - 2 * q - p;\ q := q + p;\ p := p/2,$
$\qquad\qquad \text{Else } r := 2 * r;\ p := p/2],$
$\qquad\quad \textit{Invariant} \rightarrow I]]]$

Invariant Generation - The Algorithm



Invariant Generation - The Algorithm



Program Transformation

$\{I\}$
While$[b,$
$\quad c1;\ \text{If}[b1\ \text{Then}\ c2\ \text{Else}\ c3];\ c4]$
$\{I \wedge \neg b\}$

$\longrightarrow$

$\{I\}$
While$[b,$
$\quad \text{While}[b \wedge b1', c1; c2; c4];$
$\quad \text{While}[b \wedge \neg b1', c1; c3; c4]]$
$\{I \wedge \neg b\}$

Program Transformation

$$
\begin{array}{l}
\text{Module}[\{r, p\}, \\
r := a - 1;\ q := 1;\ p := 1/2; \\
\text{While}[(2pr \geq err), \\
\text{If}[2r - 2qp \geq 0 \\
\text{Then } r := 2r - 2q - p; \\
\quad q := q + p;\ p := p/2 \\
\text{Else } r := 2r;\ p := p/2]]]
\end{array}
\longrightarrow
$$

Program Transformation

$$\begin{array}{l} \text{Module}[\{r, p\}, \\ r := a - 1;\ q := 1;\ p := 1/2; \\ \text{While}[(2pr \geq err), \\ \text{If}[2r - 2qp \geq 0 \\ \text{Then } r := 2r - 2q - p; \\ \quad q := q + p;\ p := p/2 \\ \text{Else } r := 2r;\ p := p/2]]] \end{array} \longrightarrow \begin{array}{l} \text{Module}[\{r, p\}, \\ r := a - 1;\ q := 1;\ p := 1/2; \\ \text{While}[(2pr \geq err), \\ \text{While}[(2pr \geq err) \wedge (2r - 2qp \geq 0), \\ r := 2r - 2q - p; \\ q := q + p;\ p := p/2]; \\ \text{While}[(2pr \geq err) \wedge \neg(2r - 2qp \geq 0), \\ r := 2r;\ p := p/2]]] \end{array}$$

Extracting system of recurrences

$r := a - 1;\ q := 1;\ p := 1/2;$

$\text{While}[(2pr \geq err),$

$\text{While}[(2pr \geq err) \wedge (2r - 2qp \geq 0),$

$\quad r := 2r - 2q - p;$

$\quad q := q + p;\ p := p/2];$

$\text{While}[(2pr \geq err) \wedge \neg(2r - 2qp \geq 0),$

$\quad r := 2r;\ p := p/2]]$

$i = \overline{0, \mathbf{I}}$

$$\begin{cases} p(i+1) & = & p(i)/2 \\ q(i+1) & = & q(i) + p(i) \\ r(i+1) & = & 2r(i) - 2q(i) - p(i) \end{cases}$$

$j = \overline{0, \mathbf{J}},\ j' = j + \mathbf{I}$

$$\begin{cases} p(j'+1) & = & p(j')/2 \\ q(j'+1) & = & q(j') \\ r(j'+1) & = & 2r(j') \end{cases}$$

Extracting system of recurrences

$r := a - 1;\; q := 1;\; p := 1/2;$

$\text{While}[(2pr \geq err),$

$\text{While}[(2pr \geq err) \wedge (2r - 2qp \geq 0),$

$\quad r := 2r - 2q - p;$

$\quad q := q + p;\; p := p/2];$

$\text{While}[(2pr \geq err) \wedge \neg(2r - 2qp \geq 0),$

$\quad r := 2r;\; p := p/2]]$

$i = \overline{0, \mathbf{I}}$

$$\begin{cases} p(i+1) = p(i)/2 \\ q(i+1) = q(i) + p(i) \\ r(i+1) = 2r(i) - 2q(i) - p(i) \end{cases}$$

$j = \overline{0, \mathbf{J}},\; j' = j + \mathbf{I}$

$$\begin{cases} p(j'+1) = p(j')/2 \\ q(j'+1) = q(j') \\ r(j'+1) = 2r(j') \end{cases}$$

Extracting system of recurrences

$r := a - 1;\ q := 1;\ p := 1/2;$

$\text{While}[(2pr \geq err),$

$\text{While}[(2pr \geq err) \wedge (2r - 2qp \geq 0),$

$\quad r := 2r - 2q - p;$

$\quad q := q + p;\ p := p/2];$

$\text{While}[(2pr \geq err) \wedge \neg(2r - 2qp \geq 0),$

$\quad r := 2r;\ p := p/2]]$

$i = \overline{0, \mathbf{I}}$

$$\begin{cases} p(i+1) &= p(i)/2 \\ q(i+1) &= q(i) + p(i) \\ r(i+1) &= 2r(i) - 2q(i) - p(i) \end{cases}$$

$j = \overline{0, \mathbf{J}},\ j' = j + \mathbf{I}$

$$\begin{cases} p(j'+1) &= p(j')/2 \\ q(j'+1) &= q(j') \\ r(j'+1) &= 2r(j') \end{cases}$$

Solving system of recurrences

$r := a - 1;\ q := 1;\ p := 1/2;$

While[. . . ,

While[. . . ,

$r := 2r - 2q - p;$

$q := q + p;\ p := p/2];$

While[. . . ,

$r := 2r;\ p := p/2]]$

$i = \overline{0, \mathbf{I}}$

$$\begin{cases} p(i) \overset{geom\_series}{=} \frac{1}{2^i} p(0) \\ q(i) \overset{Gosper}{\underset{zb}{=}} q(0) + 2p(0) - \frac{1}{2^{i-1}} p(0) \\ r(i) \overset{C-finite}{\underset{SumCracker}{=}} 2^i (r(0) - 2q(0) - 2p(0)) - \frac{1}{2^{i-1}} p(0) + 2q(0) + 4p(0) \end{cases}$$

$j = \overline{0, \mathbf{J}},\ j' = j + \mathbf{I}$

$$\begin{cases} p(j') \overset{geom\_series}{=} \frac{1}{2^j} p(\mathbf{I}) \\ q(j') = q(\mathbf{I}) \\ r(j') \overset{geom\_series}{=} 2^j r(\mathbf{I}) \end{cases}$$

Solving system of recurrences

$r := a - 1;\ q := 1;\ p := 1/2;$
While[. . . ,
While[. . . ,
$r := 2r - 2q - p;$
$q := q + p;\ p := p/2];$

While[. . . ,
$r := 2r;\ p := p/2]]$

$i = \overline{0, \mathbf{I}},\quad x(i) = 2^i, y(i) = 2^{-i}$

$$\begin{cases} p(i) & = & p(0)y(i) \\ q(i) & = & q(0) + 2p(0) - 2p(0)y(i) \\ r(i) & = & x(i)(r(0) - 2q(0) - 2p(0)) - \\ & & 2p(0)y(i) + 2q(0) + 4p(0) \\ 0 & \underset{\text{Dependencies}}{=} & x(i)y(i) - 1 \end{cases}$$

$j = \overline{0, \mathbf{J}},\ j' = j + \mathbf{I},\quad u(j') = 2^j, v(j') = 2^{-j}$

$$\begin{cases} p(j') & = & p(\mathbf{I})v(j') \\ q(j') & = & q(\mathbf{I}) \\ r(j') & = & r(\mathbf{I})u(j') \\ 0 & \underset{\text{Dependencies}}{=} & u(j')v(j') - 1 \end{cases}$$

Variable Elimination

$$\begin{cases} p & = & \frac{1}{2} * y * v \\ q & = & 2 - y \\ r & = & ((a - 4) * x - y + 4) * u \\ x * y - 1 & = & 0 \\ u * v - 1 & = & 0 \end{cases}$$

Variable Elimination

$$\begin{cases} p & = \frac{1}{2} * y * v \\ q & = 2 - y \\ r & = ((a - 4) * x - y + 4) * u \\ x * y - 1 & = 0 \\ u * v - 1 & = 0 \end{cases}$$

Eliminate loop counter-bounds (I,J) and extra vars (u,v,x,y)

$$a - 2 * p * r \quad = \quad q^2$$

Invariant Generation for Loops with Conditionals

$r := a - 1;\ q := 1;\ p := 1/2;$
While$[(2pr \geq err),$
If$[2r - 2qp \geq 0$
Then $r := 2r - 2q - p;$
$\quad q := q + p;\ p := p + 2$
Else $r := 2r;\ p := p/2]]]$

$$a - 2 * p * r \quad = \quad q^2$$
$$\wedge$$
$$(err \geq 0) \wedge (p \geq 0) \wedge (r \geq 0)$$

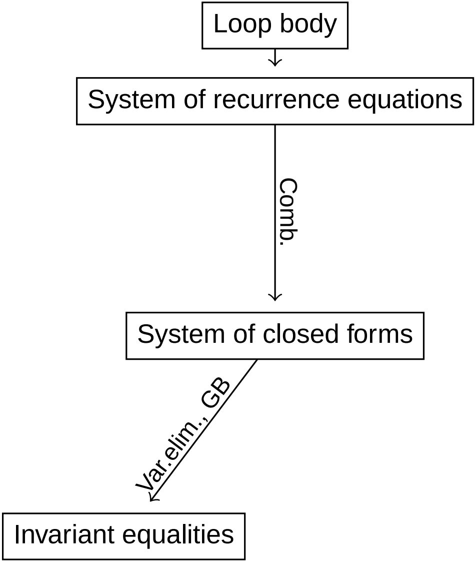
More Examples

Implementation on a Pentium 4, 1.6GHz processor with 512 Mb RAM.

Example	**Comb. Methods**	**Nr.Poly.**	**(sec)**
P-solvable loops with assignments only			
Division	Gosper	1	0.08
Integer square root	Gosper	2	0.09
Integer cubic root	Gosper	2	0.15
Fibonacci	Generating Functions, Alg.Dependencies	1	0.73
P-solvable loops with conditionals and assignments			
Wensley's Algorithm	Gosper, geom.series, Alg.Dependcies	2	0.48
LCM-GCD computation	Gosper	1	0.33
Extended GCD	Gosper	3	0.65
Fermat's factorization	Gosper	1	0.32
Square root	C-finite, Gosper, geom.series, Alg.Dependencies	1	1.28
Binary Division	C-finite, Gosper, geom.series, Alg.Dependencies	1	0.72
Floor of square root	Gosper, C-finite, geom.series, Alg.Dependencies	1	1.06
Factoring Large Numbers	C-finite, Gosper	1	1.9
Hardware Integer Division 1st Loop 2nd Loop	 geom.series, Alg.Dependencies Gosper, geom. series, Alg.Dependencies	 3 2	0.62

Outline

Generation of Invariant (Inequalities)



Generation of Invariant (Inequalities)