An Algorithm for Automated Generation of Invariants for Loops with Conditionals

Laura Ildikó Kovács, Tudor Jebelean

Research Institute for Symbolic Computation
Johannes Kepler University, Linz
{kovacs, jebelean@risc.uni-linz.ac.at}

Institute e-Austria Timişoara, Romania

Outline

Outline

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 inferences
  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 inferences
  2. access to powerful computing and solving algorithms (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- $\left\{ \begin{array}{l} \text{Proving} \\ \text{Computing} \\ \text{Solving} \end{array} \right.$

  using: specified “knowledge bases”

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

- $\left\{ \begin{array}{l} \text{Composing} \\ \text{Structuring} \\ \text{Manipulating} \end{array} \right.$ 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

- $\left\{ \begin{array}{l} \text{Proving} \\ \text{Computing} \\ \text{Solving} \end{array} \right.$
  using: specified “knowledge bases”
  applying: provers, simplifiers and solvers from the *Theorema* library
- $\left\{ \begin{array}{l} \text{Composing} \\ \text{Structuring} \\ \text{Manipulating} \end{array} \right.$ mathematical texts
- Advantages of Program Verification in *Theorema* :
  1. proofs in natural language and [illegible]
  2. access to powerful computing and solving capabilities (Mathematica)

The *Theorema* System

Theorema : A computer aided mathematical assistant

- $\left\{ \begin{array}{l} \text{Proving} \\ \text{Computing} \\ \text{Solving} \end{array} \right.$
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- $\left\{ \begin{array}{l} \text{Composing} \\ \text{Structuring} \\ \text{Manipulating} \end{array} \right.$ 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

- $\left\{\begin{array}{l}\text{Proving}\\ \text{Computing}\\ \text{Solving}\end{array}\right.$
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- $\left\{\begin{array}{l}\text{Composing}\\ \text{Structuring}\\ \text{Manipulating}\end{array}\right.$ 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

- $\left\{\begin{array}{l}\text{Proving}\\\text{Computing}\\\text{Solving}\end{array}\right.$
  using: specified "knowledge bases"
  applying: provers, simplifiers and solvers from the *Theorema* library
- $\left\{\begin{array}{l}\text{Composing}\\\text{Structuring}\\\text{Manipulating}\end{array}\right.$ 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

- $\left\{ \begin{array}{l} \text{Proving} \\ \text{Computing} \\ \text{Solving} \end{array} \right.$

  using: specified “knowledge bases”

  applying: provers, simplifiers and solvers from the *Theorema* library
- $\left\{ \begin{array}{l} \text{Composing} \\ \text{Structuring} \\ \text{Manipulating} \end{array} \right.$ 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

Program Verification

Rule–based Programming **Theorema**

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

- (consequence) verification: proving specifications based on definitions (both are logical formulae).

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning: *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: proving specifications based on definitions (both are logical formulae).

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning: *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: proving specifications based on definitions (both are logical formulae).

Imperative Programming **Theorema**

Additional assertions are needed (invariants, termination terms)

- Backward Reasoning: *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)

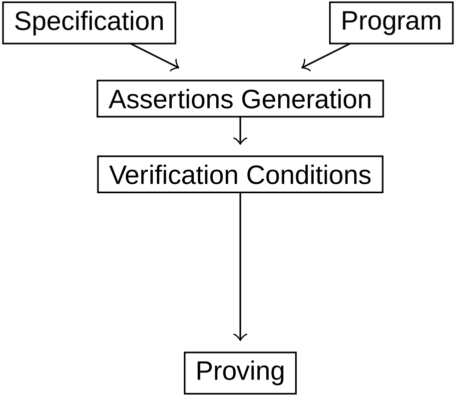
- (consequence) verification: proving specifications based on definitions (both are logical formulae).

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

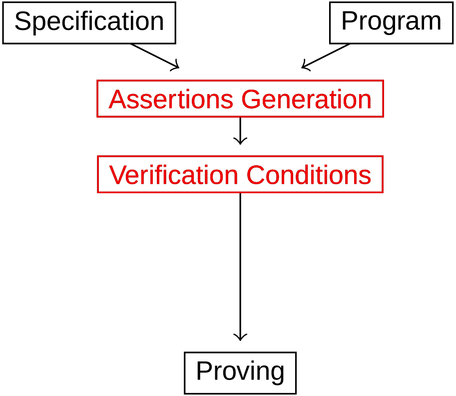
Additional assertions are needed (invariants, termination terms)

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

Imperative Program Verification in *Theorema*



Imperative Program Verification in *Theorema*



Outline

Verification Environment for Imperative Programs

Example: Wensley's Algorithm for Real Division

Specification Specification["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],

Pre $\rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0)$,

Post $\rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)$]

Program

Verification Environment for Imperative Programs

Example: Wensley's Algorithm for Real Division

Specification Specification["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],

Pre $\rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0)$,

Post $\rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)$]

Program

Verification Environment for Imperative Programs

Example: Wensley's Algorithm for Real Division

Specification Specification["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],

Pre $\rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0)$,

Post $\rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)$]

Program

Verification Environment for Imperative Programs

Example: Wensley's Algorithm for Real Division

Specification

Specification["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],
Pre $\rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0)$,
Post $\rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)$]

Program

Program["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],
Module[$\{a, b, d, y\}$,
$a := 0;\ b := Q/2;\ d := 1;\ y := 0;$
While[$d \geq Tol$,
If[$P < a + b$,
$b := b/2; d := d/2$,
$a := a + b; y := y + d/2; b := b/2; d := d/2$]];
$r := y$]]

Verification Environment for Imperative Programs

Example: Wensley's Algorithm for Real Division

Specification

$$\begin{array}{l}\text{Specification}[\text{“ReDiv”}, \text{ReDiv}[\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r], \\ \quad \text{Pre} \rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0), \\ \quad \text{Post} \rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)]\end{array}$$

Program

$$\begin{array}{l}\text{Program}[\text{“ReDiv”}, \text{ReDiv}[\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r], \\ \quad \text{Module}[\{a, b, d, y\}, \\ \qquad a := 0;\ b := Q/2;\ d := 1;\ y := 0; \\ \qquad \text{While}[d \geq Tol, \\ \qquad\quad \text{If}[P < a + b, \\ \qquad\qquad b := b/2; d := d/2, \\ \qquad\qquad a := a + b; y := y + d/2; b := b/2; d := d/2], ; \\ \qquad\qquad \mathit{Invariant} \rightarrow I,\ \mathit{TerminationTerm} \rightarrow T]; \\ \qquad r := y]]\end{array}$$

Verification Environment for Imperative Programs in *Theorema*

VCG VCG[Program[“ReDiv”], Specification[“ReDiv”]]

- Based on Hoare Logic;
- Using the Weakest Precondition Strategy;
- Output: verification conditions in a *Theorema* lemma
  $\rightarrow$ proving lemma

Execute Execute[ReDiv[7.1, 19.4, 0.01, r]]

$r = \frac{23}{64}$

Verification Environment for Imperative Programs in *Theorema*

VCG VCG[Program[“ReDiv”], Specification[“ReDiv”]]

- Based on Hoare Logic;
- Using the Weakest Precondition Strategy;
- Output: verification conditions in a *Theorema* lemma
  → proving lemma

Execute Execute[ReDiv[7.1, 19.4, 0.01, r]]

$r = \frac{23}{64}$

Verification Environment for Imperative Programs in *Theorema*

VCG VCG[Program[“ReDiv”], Specification[“ReDiv”]]

- Based on Hoare Logic;
- Using the Weakest Precondition Strategy;
- Output: verification conditions in a *Theorema* lemma
  $\rightarrow$ proving lemma

Execute Execute[ReDiv[7.1, 19.4, 0.01, r]]

$r = \frac{23}{64}$

Verification Environment for Imperative Programs in *Theorema*

VCG VCG[Program[“ReDiv”], Specification[“ReDiv”]]

- Based on Hoare Logic;
- Using the Weakest Precondition Strategy;
- Output: verification conditions in a *Theorema* lemma
  → proving lemma

Execute Execute[ReDiv[7.1, 19.4, 0.01, r]]

$r = \frac{23}{64}$

Our Approach: Algebraic and Combinatorial Methods

- Based on the *difference equations method* [ElspasGreen72]
  1. First step: find explicit forms of the loop variables, as functions of the loop counter
  2. Second step: eliminate loop counter

- In *Theorema* : invariant generation using combinatorial and algebraic techniques:
  1. Loops with assignments only (Synasc03, Synasc04): Gosper-summable, geometric series, generating functions;
  2. Loops with conditionals: combinatorics alg., Gröbner basis $\longrightarrow$ algebraic invariants;
  3. Nested Loops.

Our Approach: Algebraic and Combinatorial Methods

- Based on the *difference equations method* [ElspasGreen72]
  1. First step: find explicit forms of the loop variables, as functions of the loop counter
  2. Second step: eliminate loop counter

- In *Theorema* : invariant generation using combinatorial and algebraic techniques:
  1. Loops with assignments only (Synasc03, Synasc04): Gosper-summable, geometric series, generating functions;
  2. Loops with conditionals: combinatorics alg., Gröbner basis $\longrightarrow$ algebraic invariants;
  3. Nested Loops.

Our Approach: Algebraic and Combinatorial Methods

- Based on the *difference equations method* [ElspasGreen72]
  1. First step: find explicit forms of the loop variables, as functions of the loop counter
  2. Second step: eliminate loop counter

- In *Theorema* : invariant generation using combinatorial and algebraic techniques:
  1. Loops with assignments only (Synasc03, Synasc04): Gosper-summable, geometric series, generating functions;
  2. Loops with conditionals: combinatorics alg., Gröbner basis $\longrightarrow$ algebraic invariants;
  3. Nested Loops.

Our Approach: Algebraic and Combinatorial Methods

- Based on the *difference equations method* [ElspasGreen72]
  1. First step: find explicit forms of the loop variables, as functions of the loop counter
  2. Second step: eliminate loop counter

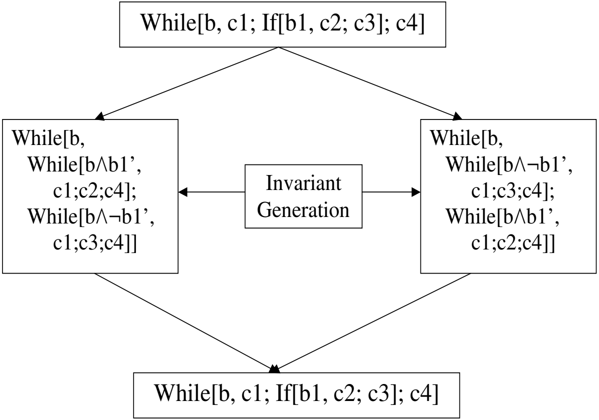
- In *Theorema* : invariant generation using combinatorial and algebraic techniques:
  1. Loops with assignments only (Synasc03, Synasc04): Gosper-summable, geometric series, generating functions;
  2. Loops with conditionals: combinatorics alg., Gröbner basis $\longrightarrow$ algebraic invariants;
  3. Nested Loops.

Our Approach: Algebraic and Combinatorial Methods

- Based on the *difference equations method* [ElspasGreen72]
  1. First step: find explicit forms of the loop variables, as functions of the loop counter
  2. Second step: eliminate loop counter

- In *Theorema* : invariant generation using combinatorial and algebraic techniques:
  1. Loops with assignments only (Synasc03, Synasc04): Gosper-summable, geometric series, generating functions;
  2. Loops with conditionals: combinatorics alg., Gröbner basis $\rightarrow$ algebraic invariants;
  3. Nested Loops.

The Algorithm



The Algorithm

- Program Transformation → Loop with only assignments

The Algorithm

- Program Transformation → Loop with only assignments

$$\frac{\begin{array}{l}\{I \wedge b1'\}\\ \text{While}[b,\\ \quad \text{While}[b \wedge b1', c1; c2; c4];\\ \quad \text{While}[b \wedge \neg b1', c1; c3; c4]]\\ \{I \wedge \neg b\}\end{array} \qquad \begin{array}{l}\{I \wedge \neg b1'\}\\ \text{While}[b,\\ \quad \text{While}[b \wedge \neg b1', c1; c3; c4];\\ \quad \text{While}[b \wedge b1', c1; c2; c4]]\\ \{I \wedge \neg b\}\end{array}}{\{I\} \quad \text{While}[b, c1; IF[b1, c2, c3]; c4] \quad \{I \wedge \neg b\}}$$

The Algorithm

- Program Transformation → Loop with only assignments

$$
\begin{array}{ll}
\text{While}[d \geq \mathit{Tol}, & \text{While}[d \geq \mathit{Tol}, \\
\quad \text{While}[d \geq \mathit{Tol} \wedge P < a+b, & \quad \text{While}[d \geq \mathit{Tol} \wedge P \geq a+b, \\
\qquad b := b/2; & \qquad a := a+b; y := y+d/2; \\
\qquad d := d/2]; & \qquad b := b/2; d := d/2]; \\
\quad \text{While}[d \geq \mathit{Tol} \wedge P \geq a+b, & \quad \text{While}[d \geq \mathit{Tol} \wedge P < a+b, \\
\qquad a := a+b; y := y+d/2; & \qquad b := b/2; \\
\qquad b := b/2; d := d/2]]; & \qquad d := d/2]];
\end{array}
$$

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops

$$\begin{aligned} a_{j_1} &= a_j & a_{j_2} &= a_{j_1} + 2 * b_{j_1} - \frac{b_{j_1}}{2^{j_2 - 1}} \\ b_{j_1} &= \frac{b_j}{2^{j_1}} & b_{j_2} &= \frac{b_{j_1}}{2^{j_2}} \\ d_{j_1} &= \frac{d_j}{2^{j_1}} & d_{j_2} &= \frac{d_{j_1}}{2^{j_2}} \\ y_{j_1} &= y_j & y_{j_2} &= y_{j_1} + d_{j_1} - \frac{d_{j_1}}{2^{j_2}} \end{aligned}$$

  2. Index and variable manipulation

$$\begin{aligned} a_{j_2} &= a_j + \frac{b_j}{2^{j_1 - 1}}\left(1 - \frac{1}{2^{j_2}}\right) \\ b_{j_2} &= \frac{b_j}{2^{j_1 + j_2}} \\ d_{j_2} &= \frac{d_j}{2^{j_1 + j_2}} \\ y_{j_2} &= y_j + \frac{d_j}{2^{j_1}}\left(1 - \frac{1}{2^{j_2}}\right) \end{aligned}$$

The Algorithm

- Program Transformation $\longrightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops

$$\begin{aligned} a_{j_1} &= a_j & a_{j_2} &= a_{j_1} + 2 * b_{j_1} - \frac{b_{j_1}}{2^{j_2-1}} \\ b_{j_1} &= \frac{b_j}{2^{j_1}} & b_{j_2} &= \frac{b_{j_1}}{2^{j_2}} \\ d_{j_1} &= \frac{d_j}{2^{j_1}} & d_{j_2} &= \frac{d_{j_1}}{2^{j_2}} \\ y_{j_1} &= y_j & y_{j_2} &= y_{j_1} + d_{j_1} - \frac{d_{j_1}}{2^{j_2}} \end{aligned}$$

  2. Index and variable manipulation

$$\begin{aligned} a_{j_2} &= a_j + \frac{b_j}{2^{j_1-1}}\left(1 - \frac{1}{2^{j_2}}\right) \\ b_{j_2} &= \frac{b_j}{2^{j_1+j_2}} \\ d_{j_2} &= \frac{d_j}{2^{j_1+j_2}} \\ y_{j_2} &= y_j + \frac{d_j}{2^{j_1}}\left(1 - \frac{1}{2^{j_2}}\right) \end{aligned}$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops

$$
\begin{aligned}
a_{j_1} &= a_j \\
b_{j_1} &= \frac{b_j}{2^{j_1}} \\
d_{j_1} &= \frac{d_j}{2^{j_1}} \\
y_{j_1} &= y_j
\end{aligned}
\qquad
\begin{aligned}
a_{j_2} &= a_{j_1} + 2 * b_{j_1} - \frac{b_{j_1}}{2^{j_2-1}} \\
b_{j_2} &= \frac{b_{j_1}}{2^{j_2}} \\
d_{j_2} &= \frac{d_{j_1}}{2^{j_2}} \\
y_{j_2} &= y_{j_1} + d_{j_1} - \frac{d_{j_1}}{2^{j_2}}
\end{aligned}
$$

  2. Index and variable manipulation

$$
\begin{aligned}
a_{j_2} &= a_j + \frac{b_j}{2^{j_1-1}}\left(1 - \frac{1}{2^{j_2}}\right) \\
b_{j_2} &= \frac{b_j}{2^{j_1+j_2}} \\
d_{j_2} &= \frac{d_j}{2^{j_1+j_2}} \\
y_{j_2} &= y_j + \frac{d_j}{2^{j_1}}\left(1 - \frac{1}{2^{j_2}}\right)
\end{aligned}
$$

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops

$$
\begin{aligned}
a_{j_1} &= a_j \\
b_{j_1} &= \frac{b_j}{2^{j_1}} \\
d_{j_1} &= \frac{d_j}{2^{j_1}} \\
y_{j_1} &= y_j
\end{aligned}
\qquad
\begin{aligned}
a_{j_2} &= a_{j_1} + 2 * b_{j_1} - \frac{b_{j_1}}{2^{j_2 - 1}} \\
b_{j_2} &= \frac{b_{j_1}}{2^{j_2}} \\
d_{j_2} &= \frac{d_{j_1}}{2^{j_2}} \\
y_{j_2} &= y_{j_1} + d_{j_1} - \frac{d_{j_1}}{2^{j_2}}
\end{aligned}
$$

  2. Index and variable manipulation

$$
\begin{aligned}
a_{j_2} &= a_j + \frac{b_j}{2^{j_1 - 1}}\left(1 - \frac{1}{2^{j_2}}\right) \\
b_{j_2} &= \frac{b_j}{2^{j_1 + j_2}} \\
d_{j_2} &= \frac{d_j}{2^{j_1 + j_2}} \\
y_{j_2} &= y_j + \frac{d_j}{2^{j_1}}\left(1 - \frac{1}{2^{j_2}}\right)
\end{aligned}
$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops

$$\begin{aligned} a_{j_1} &= a_j \\ b_{j_1} &= \frac{b_j}{2^{j_1}} \\ d_{j_1} &= \frac{d_j}{2^{j_1}} \\ y_{j_1} &= y_j \end{aligned} \qquad \begin{aligned} a_{j_2} &= a_{j_1} + 2 * b_{j_1} - \frac{b_{j_1}}{2^{j_2-1}} \\ b_{j_2} &= \frac{b_{j_1}}{2^{j_2}} \\ d_{j_2} &= \frac{d_{j_1}}{2^{j_2}} \\ y_{j_2} &= y_{j_1} + d_{j_1} - \frac{d_{j_1}}{2^{j_2}} \end{aligned}$$

  2. Index and variable manipulation

$$\begin{aligned} a_{j_2} &= a_j + \frac{b_j}{2^{j_1-1}}\left(1 - \frac{1}{2^{j_2}}\right) \\ b_{j_2} &= \frac{b_j}{2^{j_1+j_2}} \\ d_{j_2} &= \frac{d_j}{2^{j_1+j_2}} \\ y_{j_2} &= y_j + \frac{d_j}{2^{j_1}}\left(1 - \frac{1}{2^{j_2}}\right) \end{aligned}$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops
  2. Index and variable manipulation
  3. Recurrence-counters elimination

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops
  2. Index and variable manipulation
  3. Recurrence-counters elimination

$$
\begin{aligned}
-b_{j_2} + \frac{b_j * d_{j_2}}{d_j} &= 0\\
a_{j_2} * d_j - a_j * d_j - 2b_j * y_{j_2} + 2b_j * y_j &= 0\\
-b_j d_{j_2} + b_{j_2} d_j &= 0\\
-(a_{j_2} - a_j) * d_{j_2} + b_{j_2} * (-2y_j + 2y_{j_2}) &= 0\\
-2b_j + \frac{(a_{j_2} - a_j + 2b_{j_2}) * d_j}{d_{j_2} + y_{j_2} - y_j} &= 0
\end{aligned}
$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra

  For the first nested-loop system:

  1. Recurrence solving for the inner loops
  2. Index and variable manipulation
  3. Recurrence-counters elimination

$$
\begin{aligned}
-b + \frac{b_0 * d}{d_0} &= 0 \\
a * d_0 - a_0 * d_0 - 2b_0 * y + 2b_0 * y_0 &= 0 \\
-b_0 d + b d_0 &= 0 \\
-(a - a_0) * d + b * (-2y_0 + 2y) &= 0 \\
-2b_0 + \frac{(a - a_0 + 2b) * d_0}{d} + y - y_0 &= 0.
\end{aligned}
$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
  For the second nested-loop system:

$$
\begin{aligned}
a*d-a_0*d-2by+2by_0 &= 0\\
\frac{(a-a_0)*d}{d_0}-b_0-2*d_0^{-1}*d_0+y_0-y &= 0\\
-b+\frac{b_0*d}{d_0} &= 0\\
a*d_0-a_0*d_0-2b_0*y+2b_0*y_0 &= 0\\
-b_0*d+b*d_0 &= 0\\
\frac{d*(y-y_0)}{d_0}-d_0-d_0^{-1}*d_0+y_0-y &= 0\\
2b_0+\frac{(a-2b_0-a_0)*d_0}{d_0+y_0-y} &= 0
\end{aligned}
$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
- Build the union of the obtained formulae for the nested–loop subsystems → system with 12 polynomial equations
- Check invariant property

$$\begin{aligned}
-b + \frac{b_0 * d}{d_0} &= 0\\
-b_0 * d + b * d_0 &= 0\\
-(a - a_0) * d + b * (-2y_0 + 2y) &= 0\\
\frac{d * (y - y_0)}{d_0} - d_0 - d_0^{-1} * d_0 + y_0 - y &= 0\\
\frac{(a - a_0) * d}{d_0} - b_0 - 2d_0^{-1} * d_0 + y_0 - y &= 0\\
a * d - a_0 * d - 2b * y + 2 * b * y_0 &= 0
\end{aligned}$$

- Take the minimal set of the invariant properties, by using Gröbner basis w.r.t. the loop variables

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
- Build the union of the obtained formulae for the nested–loop subsystems $\rightarrow$ system with 12 polynomial equations
- Check invariant property

$$
\begin{array}{lcl}
-b + \frac{b_0 * d}{d_0} & = & 0 \\
-b_0 * d + b * d_0 & = & 0 \\
-(a - a_0) * d + b * (-2y_0 + 2y) & = & 0 \\
\frac{d * (y - y_0)}{d_0} - d_0 - d_0^{-1} * d_0 + y_0 - y & = & 0 \\
\frac{(a - a_0) * d}{d_0} - b_0 - 2d_0^{-1} * d_0 + y_0 - y & = & 0 \\
a * d - a_0 * d - 2b * y + 2 * b * y_0 & = & 0
\end{array}
$$

- Take the minimal set of the invariant properties, by using Gröbner basis w.r.t. the loop variables

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
- Build the union of the obtained formulae for the nested–loop subsystems $\rightarrow$ system with 12 polynomial equations
- Check invariant property

$$
\begin{array}{lcl}
-b + \frac{b_0 * d}{d_0} & = & 0 \\
-b_0 * d + b * d_0 & = & 0 \\
-(a - a_0) * d + b * (-2y_0 + 2y) & = & 0 \\
\frac{d * (y - y_0)}{d_0} - d_0 - d_0^{-1} * d_0 + y_0 - y & = & 0 \\
\frac{(a - a_0) * d}{d_0} - b_0 - 2d_0^{-1} * d_0 + y_0 - y & = & 0 \\
a * d - a_0 * d - 2b * y + 2 * b * y_0 & = & 0
\end{array}
$$

- Take the minimal set of the invariant properties, by using Gröbner basis w.r.t. the loop variables

$$
-b + \frac{b_0 * d}{d_0} = 0 \bigwedge a * d - a_0 * d - 2b_0 * d * d_0^{-1} * y + \frac{2 * b_0 * d * y_0}{d_0} = 0
$$

The Algorithm

- Program Transformation → Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
- Build the union of the obtained formulae for the nested–loop subsystems → system with 12 polynomial equations
- Check invariant property

$$
\begin{array}{lcl}
-b + \frac{b_0 * d}{d_0} & = & 0 \\
-b_0 * d + b * d_0 & = & 0 \\
-(a - a_0) * d + b * (-2y_0 + 2y) & = & 0 \\
\frac{d * (y - y_0)}{d_0} - d_0 - d_0^{-1} * d_0 + y_0 - y & = & 0 \\
\frac{(a - a_0) * d}{d_0} - b_0 - 2d_0^{-1} * d_0 + y_0 - y & = & 0 \\
a * d - a_0 * d - 2b * y + 2 * b * y_0 & = & 0
\end{array}
$$

- Take the minimal set of the invariant properties, by using Gröbner basis w.r.t. the loop variables

$$-b + \frac{1}{2} * d * Q = 0 \bigwedge a * d - d * y * Q = 0$$

The Algorithm

- Program Transformation $\rightarrow$ Loop with only assignments
- Invariant generation for each system of nested loops by combinatorics and algebra
- Build the union of the obtained formulae for the nested–loop subsystems
- Check invariant property
- Take the minimal set of the invariant properties, by using Gröbner basis w.r.t. the loop variables
- Invariant $\equiv$ generated algebraic property $\wedge$ asserted non-algebraic properties

$$-b+\frac{1}{2}*d*Q=0 \wedge a*d-d*y*Q=0 \wedge y \leq P/q < y+d \wedge 0 < d \leq 1$$

Application to Program Verification

Specification["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],
  Pre $\rightarrow (Q > P) \wedge (P \geq 0) \wedge (Tol \geq 0)$,
  Post $\rightarrow (P/Q < r + Tol) \wedge (r \leq P/Q)$]

Program["ReDiv", ReDiv[$\downarrow P, \downarrow Q, \downarrow Tol, \uparrow r$],
  Module[$\{a, b, d, y\}$,
    $a := 0;\ b := Q/2;\ d := 1;\ y := 0;$
    While[$d \geq Tol$,
      If[$P < a + b$,
        $b := b/2; d := d/2$,
        $a := a + b; y := y + d/2; b := b/2; d := d/2$,
          $Assert \rightarrow y \leq P/q < y + d \ \wedge\ 0 < d \leq 1$,
          $Invariant \rightarrow -b + \frac{1}{2} * d * Q = 0 \ \wedge\ a * d - d * y * Q = 0$]];
    $r := y$]]

Outline

Automated Invariant Generation - Related Work

Affine Relationships among Program Variables → **Gröbner basis**

- B.Elspas, M.W.Green, K.N.Lewitt and R.J.Waldinger (1972);
- M.Karr (1974);
- M.Müller-Olm and H.Seidl (2002, 2004);
- S.Sankaranaryanan, B.S.Henry and Z.Manna (2004);
- E.Rodriguez-Carbonell and D.Kapur (2004).

Automated Invariant Generation - Related Work

Affine Relationships among Program Variables → Gröbner basis

- B.Elspas, M.W.Green, K.N.Lewitt and R.J.Waldinger (1972);
- M.Karr (1974);
- M.Müller-Olm and H.Seidl (2002, 2004);
- S.Sankaranaryanan, B.S.Henry and Z.Manna (2004);
- E.Rodriguez-Carbonell and D.Kapur (2004).

Outline

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.

Conclusion and Further work

- Powerful tool for Invariant Generation by algebraic and combinatorial methods
  1. Loops only with assignments:
     - recurrence solving by: Gosper Alg., geometric series, Generating Function;
     - future work: integrate new recurrence solving techniques;
     - also applicable for non–linear invariant generation;
  2. Loops with conditionals:
     - Novel-algorithm based on algorithms from combinatorics and polynomial algebra;
     - future work: reduction of obtained non-linear invariant properties;
  3. future work: slight modifications $\longrightarrow$ invariant generation for nested loops;
- Treatment of other type of recurrences and solving techniques;
- Generation of invariant linear inequalities, non–algebraic invariant properties;
- Generation of termination terms.