Symbol Elimination for Automated Generation of Program Properties

Laura Kovács

Chalmers University of Technology
Sweden
2002 STUDY BY NIST:

- **Software bugs cost** the US economy $60 billion annually;

- 80% of software development costs are spent on identifying and correcting bugs.
Why is software verification needed?

- Software systems are becoming increasingly complex;
- Software is becoming integrated in almost every other system;
- Web and networking software is developing and changing at a fast pace and changes our life.

These facts impose higher than ever requirements on the quality, reliability and security of software.

Where do the difficulties come from?

- Increasing complexity of software (control-flow, mixed data structures, non-trivial arithmetic, etc);
- Expressiveness of the program assertions’ language;
- Expressiveness of the logic used for formulating verification conditions;
- Limitations on modern theorem provers;
- ...

Why is software verification needed?

- Software systems are becoming increasingly complex;
- Software is becoming integrated in almost every other system;
- Web and networking software is developing and changing at a fast pace and changes our life.

These facts impose higher than ever requirements on the quality, reliability and security of software.

Where do the difficulties come from?

- Increasing complexity of software (control-flow, mixed data structures, non-trivial arithmetic, etc);
- Expressiveness of the program assertions' language;
- Expressiveness of the logic used for formulating verification conditions;
- Limitations on modern theorem provers;
- ...
How do we tackle automated software verification?

1. Capture relevant aspects of the system formally (using recurrence equations and logic);

2. Design and implement algorithms for analyzing the formal model (computer algebra and theorem proving).
Assertion Synthesis — Example: Array Partition

Program

\[
\begin{align*}
a &:= 0; \quad b := 0; \quad c := 0; \\
\textbf{while} \ (a < N) \ \textbf{do} & \\
\quad \textbf{if} \ A[a] \geq 0 & \\
\quad \quad \textbf{then} \ B[b] := A[a]; \quad b := b + 1 & \\
\quad \quad \textbf{else} \ C[c] := A[a]; \quad c := c + 1; & \\
\quad a := a + 1; & \\
\textbf{end do} &
\end{align*}
\]

Loop Assertions

\[
\begin{align*}
a &= b + c & \\
&= b + c \\
&= a + b + c & \\
a &\geq 0 \land b \geq 0 \land c \geq 0 & \\
&\geq 0 \land b \geq 0 \land c \geq 0 & \\
&\leq N \lor N \leq 0 & \\
&\leq N \lor N \leq 0 & \\
&\forall p (p \geq b \Rightarrow B[p] = B_0[p]) & \\
&\forall p (0 \leq p < b \Rightarrow & \\
\quad B[p] \geq 0 \land & \\
\quad (\exists i)(0 \leq i < a \land A[a] = B[p])) &
\end{align*}
\]
Assertion Synthesis — Example: Array Partition

Program

\[ a := 0; \ b := 0; \ c := 0; \]

\textbf{while} \ (a < N) \ \textbf{do}

\quad \textbf{if} \ A[a] \geq 0

\quad \quad \textbf{then} \ B[b] := A[a]; \ b := b + 1

\quad \quad \textbf{else} \ C[c] := A[a]; \ c := c + 1;

\quad a := a + 1;

\textbf{end do}

Loop Assertions

Polynomial Equalities and Inequalities, Quantified FO properties

\[ a = b + c \]

\[ a \geq 0 \land b \geq 0 \land c \geq 0 \]

\[ a \leq N \lor N \leq 0 \]

\[ (\forall p)(p \geq b \Rightarrow B[p] = B_0[p]) \]

\[ (\forall p)(0 \leq p < b \Rightarrow \]

\[ B[p] \geq 0 \land \]

\[ (\exists i)(0 \leq i < a \land A[a] = B[p])) \]
Our Approach

Loop

Assertions

S Y M B O L E L I M I N A T I O N

Loop Properties

Extend language with extra symbols:
- loop cnt
- array update predicates

Eliminate symbols
Our Approach

Extend language with extra symbols:
loop cnt, array update predicates

Loop Properties

Assertions
Our Approach:

**SYMBOL ELIMINATION**

- **Loop**
  - Extend language with extra symbols: loop cnt, array update predicates

- **Loop Properties**

- **Assertions**
  - Eliminate symbols
Our Approach: **SYMBOL ELIMINATION**

- **Loop**
  - Extend *language* with extra symbols: loop cnt, array update predicates

- **Loop Properties**
  - Monotonicity Properties of Scalars
  - Array Update Properties

- **Assertions**
  - Eliminate symbols

- **Recurrence Solving**
  - Gröbner Basis and CAD

- **Consequence Finding**
Outline

Part I: Loop Properties and Symbol Elimination

Polynomial Invariants  (My PhD, with T. Jebelean)

Polynomial Invariants and Loop Bounds  (with T. Henzinger, J. Knoop and J. Zwirchmayr)

Quantified Invariants  (with K. Hoder and A. Voronkov)

Part II: Interpolants and Symbol Elimination  (with K. Hoder and A. Voronkov)
Outline

Part I: Loop Properties and Symbol Elimination

Polynomial Invariants  (My PhD, with T. Jebelean)

Polynomial Invariants and Loop Bounds  (with T. Henzinger, J. Knoop and J. Zwirchmayr)

Quantified Invariants  (with K. Hoder and A. Voronkov)

Part II: Interpolants and Symbol Elimination  (with K. Hoder and A. Voronkov)
Polynomial Invariants

\[ x := 1; \ y := 0; \]

while . . . do \[ x := 2 \ast x; \ y := \frac{1}{2} \ast y + 1 \] end do

1. Express state from \((n + 1)^{th}\) iteration in terms of \(n^{th}\) iteration

2. Solve recurrence relations \(\rightarrow\) closed forms of variables: functions of iteration counter \(n\)

3. Identify polynomial/algebraic dependencies among exponentials in \(n\)

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\rightarrow\)

5. Result: Polynomial Ideal \(\rightarrow\) Finite basis

\[
\begin{align*}
\begin{cases}
  x^{(n+1)} &= 2 \ast x^{(n)} \\
y^{(n+1)} &= \frac{1}{2} \ast y^{(n)} + 1
\end{cases}
\quad\left\{\begin{array}{l}
x^{(0)} = 2^n \ast x^{(0)} \\
y^{(0)} = \frac{1}{2^n} \ast y^{(0)} - \frac{2}{2^n} + 2
\end{array}\right.
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
  x^{(n)} &= a \ast x^{(n)} \\
y^{(n)} &= b \ast y^{(n)} - 2 \ast b + 2
\end{cases}
\quad0 &= x^{(n)} - 2 \ast x + 2 = 0
\end{align*}
\]
Polynomial Invariants

\[ x := 1; \ y := 0; \]

\[
\textbf{while} \ldots \textbf{do} \quad x := 2 \times x; \quad y := \frac{1}{2} \times y + 1 \quad \textbf{end do}
\]

1. Express state from \((n+1)^{th}\) iteration in terms of \(n^{th}\) iteration \(\to\) recurrence relations of variables;

2. Solve recurrence relations \(\to\) closed forms of variables: functions of iteration counter \(n\) \(\uparrow\) methods from symbolic summation;

3. Identify polynomial/algebraic dependencies among exponentials in \(n\);

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\to\) symbolic elimination by Gröbner bases;

5. Result: Polynomial Ideal \(\to\) Finite basis

\[
\begin{align*}
\text{for } n \geq 0, \quad a &= 2^n, \quad b = 2^{-n} \\
\left\{ \begin{array}{ll}
x^{(n+1)} &= 2 \times x^{(n)} \\
y^{(n+1)} &= \frac{1}{2} \times y^{(n)} + 1
\end{array} \right.
\end{align*}
\]

\[
\begin{align*}
\left\{ \begin{array}{ll}
x^{(n)} &= 2^n \times x^{(0)} \\
y^{(n)} &= \frac{1}{2^n} \times y^{(0)} - \frac{2}{2^n} + 2
\end{array} \right.
\end{align*}
\]

\[
\begin{align*}
x^{(n)} &= a \times x^{(0)} \\
y^{(n)} &= b \times y^{(0)} - 2 \times b + 2 \\
0 &= a \times b - 1 = 2^n \times \frac{1}{2^n} - 1
\end{align*}
\]

\[x \times y - 2 \times x + 2 = 0\]
Polynomial Invariants

\[ x := 1; \ y := 0; \]

\begin{verbatim}
while \ldots\ do x := 2 \ast x; \ y := \frac{1}{2} \ast y + 1 end do
\end{verbatim}

1. Express state from \((n + 1)^{th}\) iteration in terms of \(n^{th}\) iteration \(\rightarrow\) recurrence relations of variables;

2. Solve recurrence relations \(\rightarrow\) closed forms of variables: functions of iteration counter \(n\)
\(\uparrow\) methods from symbolic summation;

3. Identify polynomial/algebraic dependencies among exponentials in \(n\);

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\rightarrow\)
symbol elimination by Gröbner bases;

5. Result: Polynomial Ideal \(\rightarrow\) Finite basis

\[
\begin{align*}
\text{for } n \geq 0, \ a &= 2^n, \ b &= 2^{-n} \\
\begin{cases}
    x^{(n+1)} & = 2 \ast x^{(n)} \\
    y^{(n+1)} & = \frac{1}{2} \ast y^{(n)} + 1
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
    x^{(n)} & = 2^n \ast x^{(0)} \\
    y^{(n)} & = \frac{1}{2^n} \ast y^{(0)} - \frac{2}{2^n} + 2
\end{cases}
\end{align*}
\]

\[
\begin{align*}
x^{(n)} & = a \ast x^{(0)} \\
y^{(n)} & = b \ast y^{(0)} - 2 \ast b + 2 \\
0 & = a \ast b - 1 = 2^n \ast \frac{1}{2^n} - 1 \\
x \ast y - 2 \ast x + 2 & = 0
\end{align*}
\]
Polynomial Invariants

\[ x := 1; \ y := 0; \]

\[ \text{while } \ldots \text{ do } x := 2 \times x; \ y := \frac{1}{2} \times y + 1 \text{ end do} \]

1. Express state from \((n + 1)\)th iteration in terms of \(n\)th iteration \(\rightarrow\) recurrence relations of variables;

2. Solve recurrence relations \(\rightarrow\) closed forms of variables: functions of iteration counter \(n\) \(\uparrow\) methods from symbolic summation;

3. Identify polynomial/algebraic dependencies among exponentials in \(n\);

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\rightarrow\) symbol elimination by Gröbner bases;

5. Result: Polynomial Ideal \(\rightarrow\) Finite basis

\[ n \geq 0, \ a = 2^n, \ b = 2^{-n} \]

\[ \begin{align*}
  x^{(n+1)} &= 2 \times x^{(n)} \\
y^{(n+1)} &= \frac{1}{2} \times y^{(n)} + 1
\end{align*} \]

\[ \begin{align*}
x^{(n)} &= 2^n \times x^{(0)} \\
y^{(n)} &= \frac{1}{2^n} \times y^{(0)} - \frac{2}{2^n} + 2
\end{align*} \]

\[ \begin{align*}
x^{(n)} &= a \times x^{(0)} \\
y^{(n)} &= b \times y^{(0)} - 2 \times b + 2 \\
0 &= a \times b - 1 = 2^n \times \frac{1}{2^n} - 1
\end{align*} \]

\[ x \times y - 2 \times x + 2 = 0 \]
Polynomial Invariants

\[ x := 1; \quad y := 0; \]

\[ \text{while } \ldots \text{ do } x := 2 \times x; \quad y := \frac{1}{2} \times y + 1 \text{ end do} \]

1. Express state from \((n + 1)^{th}\) iteration in terms of \(n^{th}\) iteration \(\rightarrow\) recurrence relations of variables;

2. Solve recurrence relations \(\rightarrow\) closed forms of variables: functions of iteration counter \(n\) \(\uparrow\) methods from symbolic summation;

3. Identify polynomial/algebraic dependencies among exponentials in \(n\);

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\rightarrow\) symbol elimination by Gröbner bases;

5. Result: Polynomial Ideal \(\rightarrow\) Finite basis

\[ \begin{align*}
  n & \geq 0, \quad a = 2^n, \quad b = 2^{-n} \\
  x^{(n+1)} &= 2 \times x^{(n)} \\
  y^{(n+1)} &= \frac{1}{2} \times y^{(n)} + 1 \\
  x^{(n)} &= 2^n \times x^{(0)} \\
  y^{(n)} &= \frac{1}{2^n} \times y^{(0)} - \frac{2}{2^n} + 2 \\
  x^{(n)} &= a \times x^{(0)} \\
  y^{(n)} &= b \times y^{(0)} - 2 \times b + 2 \\
  0 &= a \times b - 1 = 2^n \times \frac{1}{2^n} - 1 \\
  x \times y - 2 \times x + 2 &= 0
\end{align*} \]
**Polynomial Invariants**

\[ x := 1; \ y := 0; \]

\[ \textbf{while} \ldots \textbf{do} \ x := 2 \ast x; \ y := \frac{1}{2} \ast y + 1 \textbf{end do} \]

1. Express state from \((n + 1)^{th}\) iteration in terms of \(n^{th}\) iteration \(\rightarrow\) recurrence relations of variables; \(n \geq 0, \ a = 2^n, \ b = 2^{-n}\)

\[ \begin{aligned} x^{(n+1)} &= 2 \ast x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} \ast y^{(n)} + 1 \\ \end{aligned} \]

2. Solve recurrence relations \(\rightarrow\) closed forms of variables: functions of iteration counter \(n\) \(\uparrow\) methods from \textit{symbolic summation};

\[ \begin{aligned} x^{(n)} &= 2^n \ast x^{(0)} \\ y^{(n)} &= \frac{1}{2^n} \ast y^{(0)} - \frac{2}{2^n} + 2 \\ \end{aligned} \]

3. Identify polynomial/algebraic dependencies among exponentials in \(n\);

4. Eliminate \(n\) and variables standing for algebraically related exponentials in \(n\) \(\rightarrow\) \textit{symbol elimination} by \Gröbner\ bases;

\[ \begin{aligned} x^{(n)} &= a \ast x^{(0)} \\ y^{(n)} &= b \ast y^{(0)} - 2 \ast b + 2 \\ 0 &= a \ast b - 1 = 2^n \ast \frac{1}{2^n} - 1 \\ x \ast y - 2 \ast x + 2 &= 0 \end{aligned} \]

5. Result: Polynomial Ideal \(\rightarrow\) Finite basis
Polynomial Invariants (TACAS08, IJCAR08, PSI09)

- Loops with assignments and nested conditionals.
  
  **Structural constraints** on assignments with polynomial rhs.

  ← Summation algorithms (Gosper, C-finite)

  Tests are ignored → non-deterministic programs

- Automated Loop Invariant Generation by Algebraic Techniques Over the Rationals for P-solvable Loops:

  ← Sound and complete;

- Polynomial invariant ideals represented by Gröbner bases;
Polynomial Invariants (TACAS08, IJCAR08, PSI09)

- Loops with assignments and nested conditionals.
  
  **Structural constraints** on assignments with polynomial rhs.

  ← Summation algorithms (Gosper, C-finite)

  **Tests are ignored** → non-deterministic programs

- Automated Loop Invariant Generation by Algebraic Techniques
  Over the Rationals for **P-solvable Loops**:

  ← **Sound** and **complete**;

- **Polynomial invariant ideals** represented by **Gröbner bases**;
Polynomial Invariants \textbf{(TACAS08, IJCAR08, PSI09)}

- Loops with assignments and nested conditionals.
  - Structural constraints on assignments with polynomial rhs.
  - Summation algorithms (Gosper, C-finite)
  - Tests are ignored → non-deterministic programs

- Automated Loop Invariant Generation by Algebraic Techniques Over the Rationals for P-solvable Loops:
  - Sound and complete;

- Polynomial invariant ideals represented by Gröbner bases;

- Implementation: \texttt{ALIGATOR} \hspace{1cm} \texttt{http://mtc.epfl.ch/software-tools/Aligator/}
Polynomial Invariants and Loop Bounds

Number of loop iterations $n$ of while $G$ do $S$ end do

$\exists n. \ n \geq 0 \ \land \ G(n) \ \land \ \neg G(n+1)$

- $G$ is a linear inequation;
- variables from $G$ have polynomial closed forms.

Invariants: Conjunction/disjunction of linear inequalities. Symbol elimination by CAD. (papers with R. Blanc, T. Henzinger, T. Hottelier)

Loop bounds: Solution to $n$. Symbol elimination by SMT $\rightarrow$ WCET. (papers with J. Knoop and J. Zwirchmayr)
Polynomial Invariants and Loop Bounds
(LPAR09, LPAR10)

Number of loop iterations $n$ of \textbf{while} $G$ do $S$ end do

\[ \exists n. \ n \geq 0 \land G(n) \land \neg G(n + 1) \]

- $G$ is a linear inequation;
- variables from $G$ have polynomial closed forms.

Invariants: Conjunction/disjunction of linear inequalities.

\textbf{Symbol elimination} by CAD. (papers with R. Blanc, T. Henzinger, T. Hottelier)

\textbf{Tools:} \textbf{ALIGATOR}

\url{http://mtc.epfl.ch/software-tools/Aligator/}

\textbf{ABC}

\url{http://mtc.epfl.ch/software-tools/ABC/}

Loop bounds: Solution to $n$.

\textbf{Symbol elimination} by SMT \rightarrow WCET. (papers with J. Knoop and J. Zwirchmayr)
Polynomial Invariants and Loop Bounds
(LPARD9, LPARD10, PSI11, RTNS13)

Number of loop iterations $n$ of $\textbf{while } G \textbf{ do } S \textbf{ end do}$

$$\exists n. n \geq 0 \land G(n) \land \neg G(n + 1)$$

- $G$ is a linear inequation;
- variables from $G$ have polynomial closed forms.

Invariants: Conjunction/disjunction of linear inequalities.

**Symbol elimination** by CAD.  
(papers with R. Blanc, T. Henzinger, T. Hottelier)

**Tools:**
- **ALIGATOR**  
  http://mtc.epfl.ch/software-tools/Aligator/
- **ABC**  
  http://mtc.epfl.ch/software-tools/ABC/

Loop bounds: Solution to $n$.

**Symbol elimination** by SMT $\rightarrow$ WCET.  
(papers with J. Knoop and J. Zwirchmayr)

**Tool:** **rTuBound**  
http://costa.tuwien.ac.at/tubound.html
Quantified Invariants

\[ a := 0; b := 0; c := 0; \]
\[ \textbf{while} (a \leq k) \textbf{do} \]
\[ \textbf{if} A[a] \geq 0 \]
\[ \quad \textbf{then} B[b] := A[a]; b := b + 1; \]
\[ \quad \textbf{else} C[c] := A[a]; c := c + 1; \]
\[ \quad a := a + 1; \]
\[ \textbf{end do} \]

\[
\begin{array}{cccccccc}
A: & 1 & 3 & -1 & -5 & 8 & 0 & -2 \\
\hline
a &= 0 \\
B: & * & * & * & * & * & * & * & * \\
b &= 0 \\
C: & * & * & * & * & * & * & * & * \\
c &= 0 \\
\end{array}
\]
Quantified Invariants

\[
a := 0; \quad b := 0; \quad c := 0;
\]

\begin{algorithm}
\textbf{while} (a \leq k) \textbf{do}
  \quad \textbf{if} A[a] \geq 0
  \quad \quad \textbf{then} B[b] := A[a]; b := b + 1;
  \quad \quad \textbf{else} C[c] := A[a]; c := c + 1;
  \quad a := a + 1;
\textbf{end do}
\end{algorithm}

\[
A : \begin{bmatrix} 1 & 3 & -1 & -5 & 8 & 0 & -2 \end{bmatrix}
\]
\[a = 7\]

\[
B : \begin{bmatrix} 1 & 3 & 8 & 0 & * & * & * \end{bmatrix}
\]
\[b = 4\]

\[
C : \begin{bmatrix} -1 & -5 & -2 & * & * & * & * \end{bmatrix}
\]
\[c = 3\]
Quantified Invariants

\[ a := 0; \quad b := 0; \quad c := 0; \]

\[
\textbf{while} \ (a \leq k) \ \textbf{do} \\
\quad \textbf{if} \ A[a] \geq 0 \\
\quad \quad \textbf{then} \ B[b] := A[a]; \ b := b + 1; \\
\quad \quad \textbf{else} \ C[c] := A[a]; \ c := c + 1; \\
\quad \quad a := a + 1; \\
\textbf{end do}
\]

\[
A:\begin{array}{cccccccc}
1 & 3 & -1 & -5 & 8 & 0 & -2 \\
a = 7
\end{array}
\]

\[
B:\begin{array}{cccccccc}
1 & 3 & 8 & 0 & * & * & * \\
b = 4
\end{array}
\]

\[
C:\begin{array}{cccccccc}
-1 & -5 & -2 & * & * & * & * \\
c = 3
\end{array}
\]

Invariants with \( \forall \exists \)

- Each of \( B[0], \ldots, B[b - 1] \) is non-negative and equal to one of \( A[0], \ldots, A[a - 1] \).

\[
(\forall p)(0 \leq p < b \Rightarrow B[p] \geq 0 \land (\exists i)(0 \leq i < a \land A[i] = B[p]))
\]
Quantified Invariants

\[
a := 0; b := 0; c := 0;
\]
\[\textbf{while} (a \leq k) \textbf{do}
\]
\[\textbf{if} \ A[a] \geq 0
\]
\[\textbf{then} \ B[b] := A[a]; b := b + 1;
\]
\[\textbf{else} \ C[c] := A[a]; c := c + 1;
\]
\[a := a + 1;
\]
\[\textbf{end do}
\]

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>3</th>
<th>-1</th>
<th>-5</th>
<th>8</th>
<th>0</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>1</th>
<th>3</th>
<th>8</th>
<th>0</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>-1</th>
<th>-5</th>
<th>-2</th>
<th>*</th>
<th>*</th>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Invariants with \(\forall \exists\)

- Each of \(B[0], \ldots, B[b - 1]\) is non-negative and equal to one of \(A[0], \ldots, A[a - 1]\).

\[(\forall p)(0 \leq p < b \Rightarrow B[p] \geq 0 \land (\exists i)(0 \leq i < a \land A[i] = B[p]))\]
Quantified Invariants

\[ a := 0; \ b := 0; \ c := 0; \]

**while** \( a \leq k \) **do**

**if** \( A[a] \geq 0 \)

**then** \( B[b] := A[a]; \ b := b + 1; \)

**else** \( C[c] := A[a]; \ c := c + 1; \)

\( a := a + 1; \)

**end do**

\[ A := \begin{array}{ccccccc}
1 & 3 & -1 & -5 & 8 & 0 & -2 \\
\end{array} \]

\( a = 7 \)

\[ B := \begin{array}{cccccccc}
1 & 3 & 8 & 0 & * & * & * & \ *
\end{array} \]

\( b = 4 \)

\[ C := \begin{array}{cccccccc}
-1 & -5 & -2 & * & * & * & * & * \\
\end{array} \]

\( c = 3 \)

**Invariants with** \( \forall \exists \)

- Each of \( B[0], \ldots, B[b - 1] \) is non-negative and equal to one of \( A[0], \ldots, A[a - 1] \).
- Each of \( C[0], \ldots, C[c - 1] \) is negative and equal to one of \( A[0], \ldots, A[a - 1] \).

**Invariants with** \( \forall \)

- For every \( p \geq b \), the value of \( B[p] \) is equal to its initial value.
- For every \( p \geq c \), the value of \( C[p] \) is equal to its initial value.
Invariants and Symbol Elimination: The General Picture

- **Loop**
  - Extend language with extra symbols: loop cnt, array update predicates

- **Loop Properties**

- **Invariants**
  - Eliminate symbols
Quant. Invariants and Symbol Elimination

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   ▶ loop counter $n$;
   ▶ variables as functions of $n$: $v^{(i)}$ with $0 \leq i < n$
   ▶ loop property predicates: $\text{iter}$, $\text{upd}_V(i, p)$, $\text{upd}_V(i, p, x)$

- $\text{upd}_V(i, p)$: at iteration $i$, $V$ is updated at position $p$;
- $\text{upd}_V(i, p, x)$: at iteration $i$, $V$ is updated at position $p$ by value $x$.

```
a := 0; b := 0; c := 0;
while (a \leq k) do
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1$
    else $C[c] := A[a]; c := c + 1$
  a := a + 1;
end do
```

$(\forall i)(i \in \text{iter} \Leftrightarrow 0 \leq i \land i < n)$

$\text{upd}_B(i, p) \Leftrightarrow i \in \text{iter} \land p = b^{(i)} \land A[a^{(i)}] \geq 0$

$\text{upd}_B(i, p, x) \Leftrightarrow \text{upd}_B(i, p) \land x = A[a^{(i)}]$

$a = b + c$, $a \geq 0$, $b \geq 0$, $c \geq 0$

$(\forall i \in \text{iter})(a^{(i+1)} > a^{(i)})$

$(\forall i \in \text{iter})(b^{(i+1)} = b^{(i)} \lor b^{(i+1)} = b^{(i)} + 1)$

$(\forall i \in \text{iter})(a^{(i)} = a^{(0)} + i)$

$(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$

$(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(i)} + k \geq b^{(k)} + j)$

$(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in \text{iter})(b^{(i)} = p \land A[a^{(i)}] \geq 0))$

$(\forall i)\neg \text{upd}_B(i, p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$

$\text{upd}_B(i, p, x) \land (\forall j > i)\neg \text{upd}_B(j, p) \Rightarrow B^{(n)}[p] = x$

$(\forall i \in \text{iter})(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \land b^{(i+1)} = b^{(i)} + 1 \land c^{(i+1)} = c^{(i)})$
Quant. Invariants and Symbol Elimination

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - loop counter $n$;
   - variables as functions of $n$: $v^{(i)}$ with $0 \leq i < n$
   - loop property predicates: $\text{iter}$, $\text{upd}_V(i, p)$, $\text{upd}_V(i, p, x)$

2. Collect loop properties in $\mathcal{L}'$:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

3. Eliminate symbols $\rightarrow$ Invariants

```plaintext
a := 0; b := 0; c := 0;
while (a ≤ k) do
    if $A[a] \geq 0$
        then $B[b] := A[a]; b := b + 1$
        else $C[c] := A[a]; c := c + 1$
    a := a + 1;
end do
```

$(\forall i)(i \in \text{iter} \iff 0 \leq i \land i < n)$

$\text{upd}_B(i, p) \iff i \in \text{iter} \land p = b^{(i)} \land A[a^{(i)}] \geq 0$

$\text{upd}_B(i, p, x) \iff \text{upd}_B(i, p) \land x = A[a^{(i)}]$
Quant. Invariants and Symbol Elimination

1. Extend the language $L$ to $L'$:
   - loop counter $n$;
   - variables as functions of $n$: $v(i)$ with $0 \leq i < n$;
   - loop property predicates: $\text{iter}$, $\text{upd}_V(i, p)$, $\text{upd}_V(i, p, x)$

2. Collect loop properties in $L'$:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

3. Eliminate symbols $\rightarrow$ Invariants

---

While $(a \leq k)$ do
  if $A[a] \geq 0$ then $B[b] := A[a]; b := b + 1;$
  else $C[c] := A[a]; c := c + 1;$
end do

$a := 0; b := 0; c := 0;

(\forall i)(i \in \text{iter} \Leftrightarrow 0 \leq i \land i < n)$

$\text{upd}_B(i, p) \Leftrightarrow i \in \text{iter} \land p = b^{(i)} \land A[a^{(i)}] \geq 0$

$\text{upd}_B(i, p, x) \Leftrightarrow \text{upd}_B(i, p) \land x = A[a^{(i)}]$

$a = b + c, \ a \geq 0, \ b \geq 0, \ c \geq 0$

$(\forall i \in \text{iter})(a^{(i+1)} > a^{(i)})$

$(\forall i \in \text{iter})(b^{(i+1)} = b^{(i)} \lor b^{(i+1)} = b^{(i)} + 1)$

$(\forall i \in \text{iter})(a^{(i)} = a^{(0)} + i)$

$(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$

$(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$

$(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in \text{iter})(b^{(i)} = p \land A[a^{(i)}] \geq 0))$

$(\forall i)\neg \text{upd}_B(i, p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$

$\text{upd}_B(i, p, x) \land (\forall j > i)\neg \text{upd}_B(j, p) \Rightarrow B^{(n)}[p] = x$

$(\forall i \in \text{iter})(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \land b^{(i+1)} = b^{(i)} + 1 \land c^{(i+1)} = c^{(i)})$
Quant. Invariants and Symbol Elimination

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - loop counter $n$;
   - variables as functions of $n$: $v^{(i)}$ with $0 \leq i < n$;
   - loop property predicates: $\mathrm{iter}$, $\mathrm{upd}_V(i, p)$, $\mathrm{upd}_V(i, p, x)$

2. Collect loop properties in $\mathcal{L}'$:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

3. Eliminate symbols $\rightarrow$ Invariants
   
   HOW?
Quantified Invariants and Symbol Elimination

\[
(\forall i)(i \in \text{iter} \iff 0 \leq i \land i < n)
\]

\[
\text{upd}_B(i, p) \iff i \in \text{iter} \land p = b^{(i)} \land A[a^{(i)}] \geq 0
\]

\[
\text{upd}_B(i, p, x) \iff \text{upd}_B(i, p) \land x = A[a^{(i)}]
\]

\[
a = b + c, \ a \geq 0, \ b \geq 0, \ c \geq 0
\]

\[
(\forall i \in \text{iter})(a^{(i+1)} > a^{(i)})
\]

\[
(\forall i \in \text{iter})(b^{(i+1)} = b^{(i)} \lor b^{(i+1)} = b^{(i)} + 1)
\]

\[
(\forall i \in \text{iter})(a^{(i)} = a^{(0)} + i)
\]

\[
(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})
\]

\[
(\forall j, k \in \text{iter})(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)
\]

\[
(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in \text{iter})(b^{(i)} = p \land A[a^{(i)}] \geq 0))
\]

\[
(\forall i)(\neg \text{upd}_B(i, p) \Rightarrow B^{(n)}[p] = B^{(0)}[p])
\]

\[
\text{upd}_B(i, p, x) \land (\forall j > i)(\neg \text{upd}_B(j, p) \Rightarrow B^{(n)}[p] = x)
\]

\[
(\forall i \in \text{iter})(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \land
b^{(i+1)} = b^{(i)} + 1 \land
\]

\[
c^{(i+1)} = c^{(i)}
\]

Saturation Theorem Proving \[ l_1, l_2, l_3, l_4, l_5, \ldots \]
Symbol Elimination by First-Order Theorem Proving

1. Reasoning in first-order theories

\[
\begin{align*}
x \geq y & \iff x > y \lor x = y \\
x > y & \Rightarrow x \neq y \\
x \geq y \land y \geq z & \Rightarrow x \geq z \\
s(x) > x & \\
x \geq s(y) & \iff x > y
\end{align*}
\]

2. Procedures for eliminating symbols → INVARIA NTS
Symbol Elimination by First-Order Theorem Proving

1. Reasoning in first-order theories

\[
x \geq y \iff x > y \lor x = y \\
x > y \Rightarrow x \neq y \\
x \geq y \land y \geq z \Rightarrow x \geq z \\
s(x) > x \\
x \geq s(y) \iff x > y
\]

2. Procedures for eliminating symbols

▷ For every loop variable \( v \) \( \rightarrow \) TARGET SYMBOLS \( v_0 \) and \( v \):

\[
\nu^{(0)} = v_0 \quad \text{and} \quad \nu^{(n)} = v
\]

▷ USABLE symbols:
- target or interpreted symbols;
- skolem functions introduced by Vampire;

▷ Reduction (elimination) ordering \( \succ \):
  useless symbols \( \succ \) usable symbols.
Symbol Elimination by First-Order Theorem Proving

1. Reasoning in first-order theories

\[
\begin{align*}
  x \geq y & \iff x > y \lor x = y \\
  x > y & \Rightarrow x \neq y \\
  x \geq y \land y \geq z & \Rightarrow x \geq z \\
  s(x) > x & \\
  x \geq s(y) & \iff x > y
\end{align*}
\]

2. Procedures for eliminating symbols \( \rightarrow \) INVARIANTS

- For every loop variable \( v \rightarrow \) TARGET SYMBOLS \( v_0 \) and \( v \):

\[
\begin{align*}
  v^{(0)} &= v_0 \\
  v^{(n)} &= v
\end{align*}
\]

- **Usable** symbols:
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;

- Reduction (elimination) ordering \( \succ \):
  - useless symbols \( \succ \) usable symbols.
<table>
<thead>
<tr>
<th>Loop</th>
<th># SEI</th>
<th># Min SEI</th>
<th>Inv of interest</th>
<th>Generated invariants implying Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>24</td>
<td>5</td>
<td>∀x : 0 ≤ x &lt; a ⇒ B[x] = A[x]</td>
<td>inv8: ∀x₀, x₁ : A[x₀] = B[x₁] ∨ x₀ ≠ x₁ ∨ ¬a &gt; x₀ ∨ ¬x₀ ≥ 0</td>
</tr>
<tr>
<td>while (a &lt; m) do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[a] = A[a]; a = a + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>151</td>
<td>13</td>
<td>spot = m ∨ A[spot] ≠ 0</td>
<td>inv3: a ≥ spot</td>
</tr>
<tr>
<td>a = 0; spot = m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while (a &lt; m) do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if (spot = m &amp;&amp; A[a] ≠ 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>then spot = a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B[a] = (A[a] ≠ 0);</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = a + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition</td>
<td>166</td>
<td>38</td>
<td>∀x : 0 ≤ x &lt; b ⇒ B[x] ≥ 0 ∧ ∃y : B[x] = A[y]</td>
<td>inv1: ∀x₀ : A(sk₂(x₀)) ≥ 0 ∨ ¬b &gt; x₀ ∨ ¬x₀ ≥ 0</td>
</tr>
<tr>
<td>a = 0; b = 0; c = 0;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while (a &lt; m) do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if (A[a] == 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>then B[b] = A[a]; b = b + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>else C[c] = A[a]; c = c + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = a + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition_Init</td>
<td>168</td>
<td>24</td>
<td>∀x : 0 ≤ x &lt; c ⇒ A[C[x]] = B[C[x]]</td>
<td>inv0: ∀x₀ : A(sk₁(x₀)) = B(sk₁(x₀)) ∨ ¬c &gt; x₀ ∨ ¬x₀ ≥ 0</td>
</tr>
<tr>
<td>a = 0; c = 0;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while (a &lt; m) do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if (A[a] == B[a])</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>then C[c] = a; c = c + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end if</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = a + 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end do</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: Vampire with 1 second time limit.
<table>
<thead>
<tr>
<th>Loop Shape</th>
<th># of Loops</th>
<th>Average # of SEI</th>
<th>Average # of Non-Redundant SEI</th>
<th>% of SEI Redundancy</th>
<th>∀-Inv</th>
<th>∀∃-Inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>33</td>
<td>168</td>
<td>18</td>
<td>89.3%</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Multi-path</td>
<td>5</td>
<td>340</td>
<td>46</td>
<td>86.4%</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Table**: Vampire on industrial benchmarks of Dassault Aviation.
Quantified Invariants and Symbol Elimination

(FASE09, IJCAR10, TACAS11, CAV13, PSI14, ATVA14)

Symbol elimination by saturation-based theorem proving.

(papers with K. Hoder and A. Voronkov)

Tool: Vampire http://vprover.org
Outline

Part I: Loop Properties and Symbol Elimination

Polynomial Invariants  (My PhD, with T. Jebelean)

Polynomial Invariants and Loop Bounds  (with T. Henzinger, J. Knoop and J. Zwirchmayr)

Quantified Invariants  (with K. Hoder and A. Voronkov)

Part II: Interpolants and Symbol Elimination  (with K. Hoder and A. Voronkov)
Symbol Elimination and Interpolation

\[
\begin{align*}
\{\text{Pre}\} & \quad \text{Loop} \\
A & \\
P_1 & \\
P_2 & \\
\vdots & \\
P_n & \implies \text{Invariant/Interpolant} \\
& \quad \implies \\
\{\text{Post}\} & \\
B & 
\end{align*}
\]
Symbol Elimination and Interpolation

\{\text{Pre}\} \quad \text{Loop} \quad \begin{align*}
P_1 \\
P_2 \\
\vdots \\
P_n
\end{align*} \quad \implies \text{Invariant/Interpolant} \quad \implies \quad \begin{align*}
A \\
B
\end{align*} \quad \{\text{Post}\}
Symbol Elimination and Interpolation

\[
\{\text{Pre}\} \quad \text{Loop} \quad \begin{array}{c}
P_1 \\
P_2 \\
\vdots \\
P_n \\
\end{array} \quad \Rightarrow \quad \text{Invariant/Interpolant} \quad \Rightarrow \quad \{\text{Post}\} \\
\begin{array}{c}
Q_1 \\
Q_2 \\
\vdots \\
Q_m \\
\end{array}
\]
Invariants, Symbol Elimination, and Interpolation

\[ A : \quad \{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0) \}\]

\textbf{while} \ (c < N) \ \textbf{do}
\begin{align*}
C[c] & := D[d]; \\
c & := c + 1; \\
d & := d + 1
\end{align*}
\textbf{end do}

\[ B : \quad \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}\]
Reachability of $B$ in ONE iteration: $A \land \text{One Loop Iteration} \Rightarrow B$

$A : \quad \{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0)\}$

while $(c < N)$ do

$C[c] : = D[d]$;
$c : = c + 1$;
$d : = d + 1$

end do

$B : \quad \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$
Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A: \{ c = d = 0 \land N > 0 \land (\forall k)(0 \leq k < N \Rightarrow D[k] = 0)\}$

```
while (c < N) do
    C[c] := D[d];
    c := c + 1;
    d := d + 1
end do
```

$B: \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A : \{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0) \}$

$(c < N) \land$

$C[c] = D[d] \land$

$c' = c + 1 \land$

$d' = d + 1 \land$

$(c' \geq N)$

$B : \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$
Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A : \{ c = d = 0 \land N > 0 \land (\forall k)(0 \leq k < N \Rightarrow D[k] = 0) \}$

\[(c < N) \land \]
\[C[c] = D[d] \land \]
\[c' = c + 1 \land \]
\[d' = d + 1 \land \]
\[(c' \geq N)\]

$B : \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$

Refutation: $A \land \text{One\_Loop\_Iteration} \land \neg B$

- The formula is over 2 states: $(c, d)$ and $(c', d')$.
- Need a state formula $I_{c',d'}$ over the FINAL state $(c', d')$ such that:
  \[A \land \text{One\_Loop\_Iteration} \Rightarrow I_{c',d'} \quad \text{and} \quad I_{c',d'} \land \neg B \Rightarrow \bot\]
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A: \{ \begin{array}{c} c = d = 0 \land N > 0 \land (\forall k)(0 \leq k < N \Rightarrow D[k] = 0) \end{array} \}
\begin{array}{c} (c < N) \land \\ C[c] = D[d] \land \\ c' = c + 1 \land \\ d' = d + 1 \land \\ (c' \geq N) \end{array}$

$B: \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$

Refutation: $A \land \text{One\_Loop\_Iteration} \land \neg B$

- The formula is over 2 states: $(c, d)$ and $(c', d')$.
- Need a state formula $I_{c',d'}$ over the FINAL state $(c', d')$ such that:
  
  $A \land \text{One\_Loop\_Iteration} \Rightarrow I_{c',d'}$ and $I_{c',d'} \land \neg B \Rightarrow \bot$
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A: \{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0)\}$

$(c < N) \land$

$C[c] = D[d] \land$

$c' = c + 1 \land$

$d' = d + 1 \land$

$(c' \geq N)$

$B: \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$

Refutation: $A \land \text{One\_Loop\_Iteration} \land \neg B$

- The formula is over 2 states: $(c, d)$ and $(c', d')$.
- Need a state formula $I_{c',d'}$ over the FINAL state $(c', d')$ such that:

$A \land \text{One\_Loop\_Iteration} \Rightarrow I_{c',d'}$ and $I_{c',d'} \land \neg B \Rightarrow \bot$

Taks: Compute interpolant $I_{c',d'}$ by eliminating symbols $c, d$. 
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in ONE iteration: $A \land \text{One\_Loop\_Iteration} \Rightarrow B$

$A: \{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0)\}$

$(c < N) \land C[c] = D[d] \land c' = c + 1 \land d' = d + 1 \land (c' \geq N)$

$B: \{(\forall k)(0 \leq k < N \Rightarrow C[k] = 0)\}$

$I_{c',d'} \equiv 0 < c' = 1 \land C[0] = D[0]$

Taks: Compute interpolant $I_{c',d'}$ by eliminating symbols $c, d$. 
Reachability of $B$ in TWO iteration: $A \land \text{Two\_Loop\_Iteration} \Rightarrow B$

$A: \{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0)\}$

$(c < N) \land \lnot N \land C[c] = D[d] \land C[c'] = D[d'] \land c' = c + 1 \land c'' = c' + 1 \land d' = d + 1 \land d'' = d' + 1 \land (c' < N) \land c'' \geq N$

$B: \{(\forall k) (0 \leq k < N \Rightarrow C[k] = 0)\}$
Invariants, Symbol Elimination, and Interpolation

Reachability of \( B \) in TWO iteration: \( A \land \text{Two\_Loop\_Iteration} \Rightarrow B \)

\( A : \{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0) \} \)

\[
\begin{align*}
(c < N) & \land \\
C[c] = D[d] & \land \\
C[c'] = D[d'] & \\
c' = c + 1 & \land \\
d' = d + 1 & \land \\
c'' = c' + 1 & \land \\
d'' = d' + 1 & \land \\
(c' < N) & \land \\
c'' \geq N & \land
\end{align*}
\]

\( B : \{ (\forall k)(0 \leq k < N \Rightarrow C[k] = 0) \} \)

\[
\begin{align*}
l_{c',d'} & \equiv 0 < c' = 1 \land C[0] = D[0] \\
l_{c'',d''} & \equiv 0 < c'' = 2 \land C[0] = D[0] \land C[1] = D[1]
\end{align*}
\]

Taks: Compute interpolant \( l_{c'',d''} \) by eliminating symbols \( c, d, c', d' \).
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in TWO iteration: $A \land \text{Two\_Loop\_Iteration} \Rightarrow B$

$A:$ \{ $c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \Rightarrow D[k] = 0)$ \}

\[
(c < N) \land \\
C[c] = D[d] \land \\
c' = c + 1 \land \\
d' = d + 1 \land \\
(c' < N) \land \\
\]

$B:$ \{ $\forall k)(0 \leq k < N \Rightarrow C[k] = 0)$ \}

\[
I_{c',d'} \equiv (\forall k)0 \leq k < c' \Rightarrow C[k] = D[k] \\
I_{c'',d''} \equiv (\forall k)0 \leq k < c'' \Rightarrow C[k] = D[k]
\]

Taks: Compute interpolant / implying invariant in any state.
Extracting Interpolants

- Given $A$ and $B$ such that $A \Rightarrow B$;
- Compute an interpolant $I$ from a proof of $A \Rightarrow B$. 
Extracting Interpolants by Symbol Elimination

- Given $A$ and $B$ such that $A \Rightarrow B$;
- Compute an interpolant $I$ from a local proof of $A \Rightarrow B$. 
Shape of local proofs for $A \Rightarrow B$
Let $\Pi$ be a local proof of $A \Rightarrow B$.

Then:

- An interpolant $I$ of $A$ and $B$ can be extracted from $\Pi$ in linear time.
- $I$ is ground if all formulas in $\Pi$ are ground.
- $I$ is a boolean combination of conclusions of symbol-eliminating inferences of $\Pi$.

Tool: Vampire  
[http://vprover.org](http://vprover.org)
Let $\Pi$ be a local proof of $A \Rightarrow B$.

Then:

- An interpolant $I$ of $A$ and $B$ can be extracted from $\Pi$ in linear time.
- $I$ is ground if all formulas in $\Pi$ are ground.
- $I$ is a boolean combination of conclusions of symbol-eliminating inferences of $\Pi$.

Tool: Vampire [http://vprover.org]
Interpolants and Symbol Elimination
(CADE09, IJCAR10, POPL12, CAV13)

(papers with K. Hoder and A. Voronkov)

Let $\Pi$ be a local proof of $A \Rightarrow B$.

Then:

- An interpolant $I$ of $A$ and $B$ can be extracted from $\Pi$ in linear time.
- $I$ is ground if all formulas in $\Pi$ are ground.
- $I$ is a boolean combination of conclusions of symbol-eliminating inferences of $\Pi$.

Tool: Vampire http://vprover.org
Summary: Symbol Elimination in Program Analysis

Given the proof obligation $A \Rightarrow B$:

1. Run a theorem prover and eliminate extra symbols;
2. Generate an interpolant from a proof;
3. Interpolants are boolean combinations of symbol-eliminating inferences.

Given a loop:

1. Express loop properties in a language containing extra symbols;
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover and/or a decision procedure for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are consequences of symbol-eliminating inferences.
Summary: Symbol Elimination in Program Analysis

Given the proof obligation $A \Rightarrow B$:

1. Run a theorem prover and eliminate extra symbols;
2. Generate a interpolant from a proof;
3. Interpolants are boolean combinations of symbol-eliminating inferences.

Given a loop:

1. Express loop properties in a language containing extra symbols;
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover and/or a decision procedure for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are consequences of symbol-eliminating inferences.