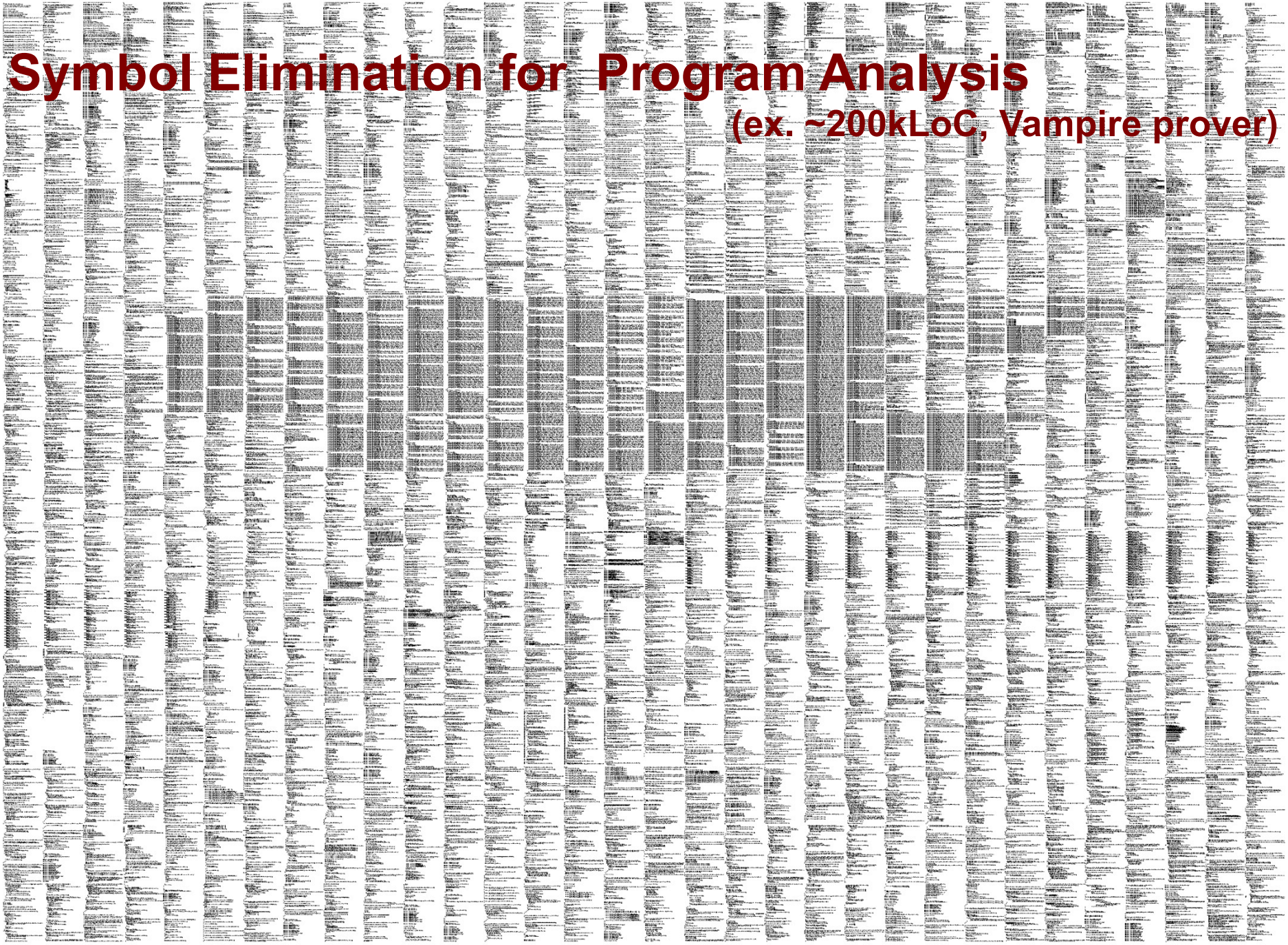
Symbol Elimination for Program Analysis

Laura Kovács

Symbol Elimination for Program Analysis

(ex. ~200kLoC, Vampire prover)



Symbol Elimination for Program Analysis

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Symbol Elimination for Program Analysis

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Program property:

$$(\forall p)(0 \le p < b \Rightarrow (\exists q)(0 \le q < a \wedge B[p]=A[q]+h(p) \wedge A[q]>0)$$

Symbol Elimination for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

h

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Symbol Elimination for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

h

Program property:

$fib1^4 + fib2^4 + 2*fib1*fib2^3 - 2\ fib1^3*fib2 - fib1^2*fib2^2 - 1 = 0$

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Symbol Elimination for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

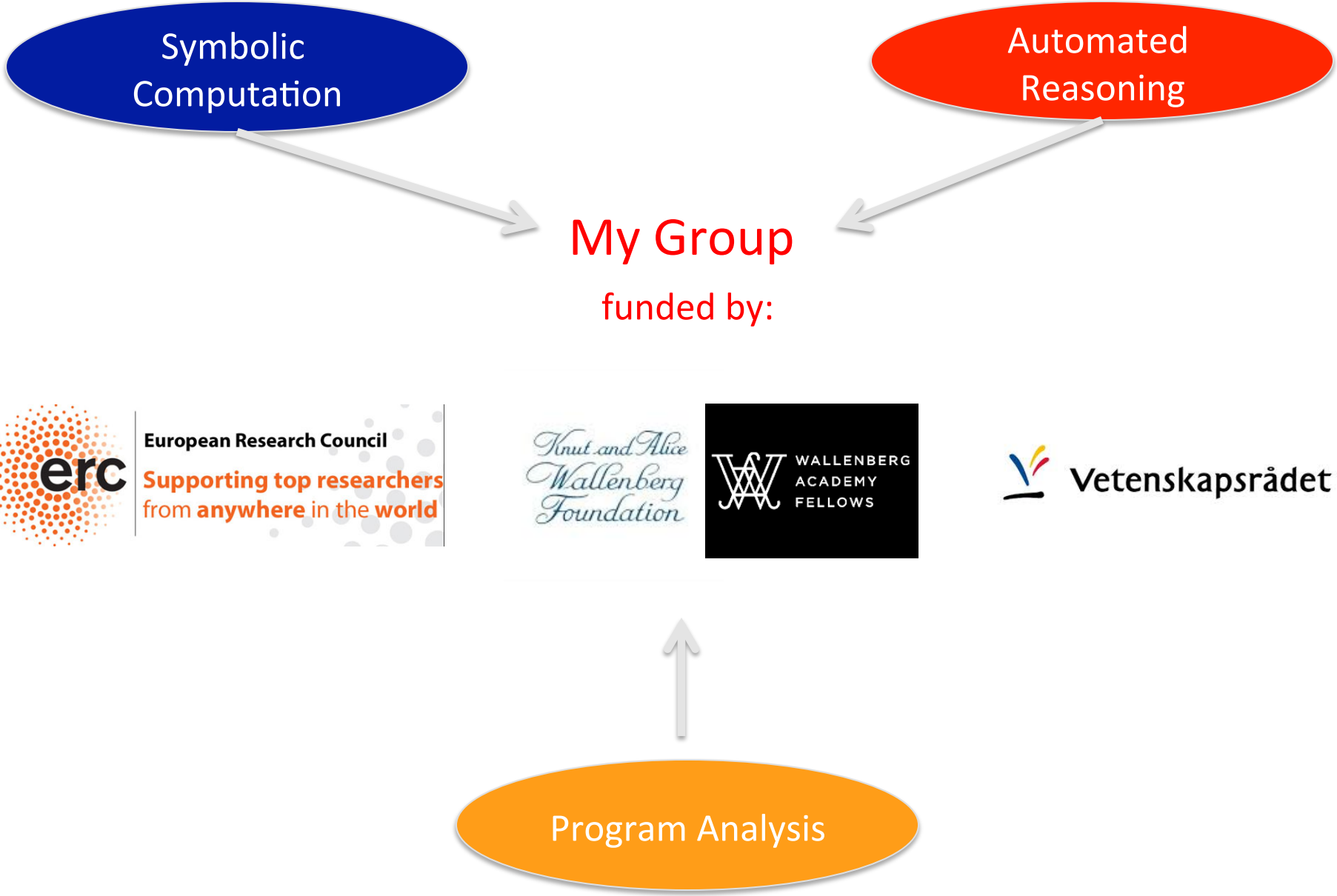
h

$fib1^4 + fib2^4 + 2*fib1*fib2^3 - 2\ fib1^3*fib2 - fib1^2*fib2^2 - 1 = 0$

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

$(\forall p)(0 \le p < b \Rightarrow$

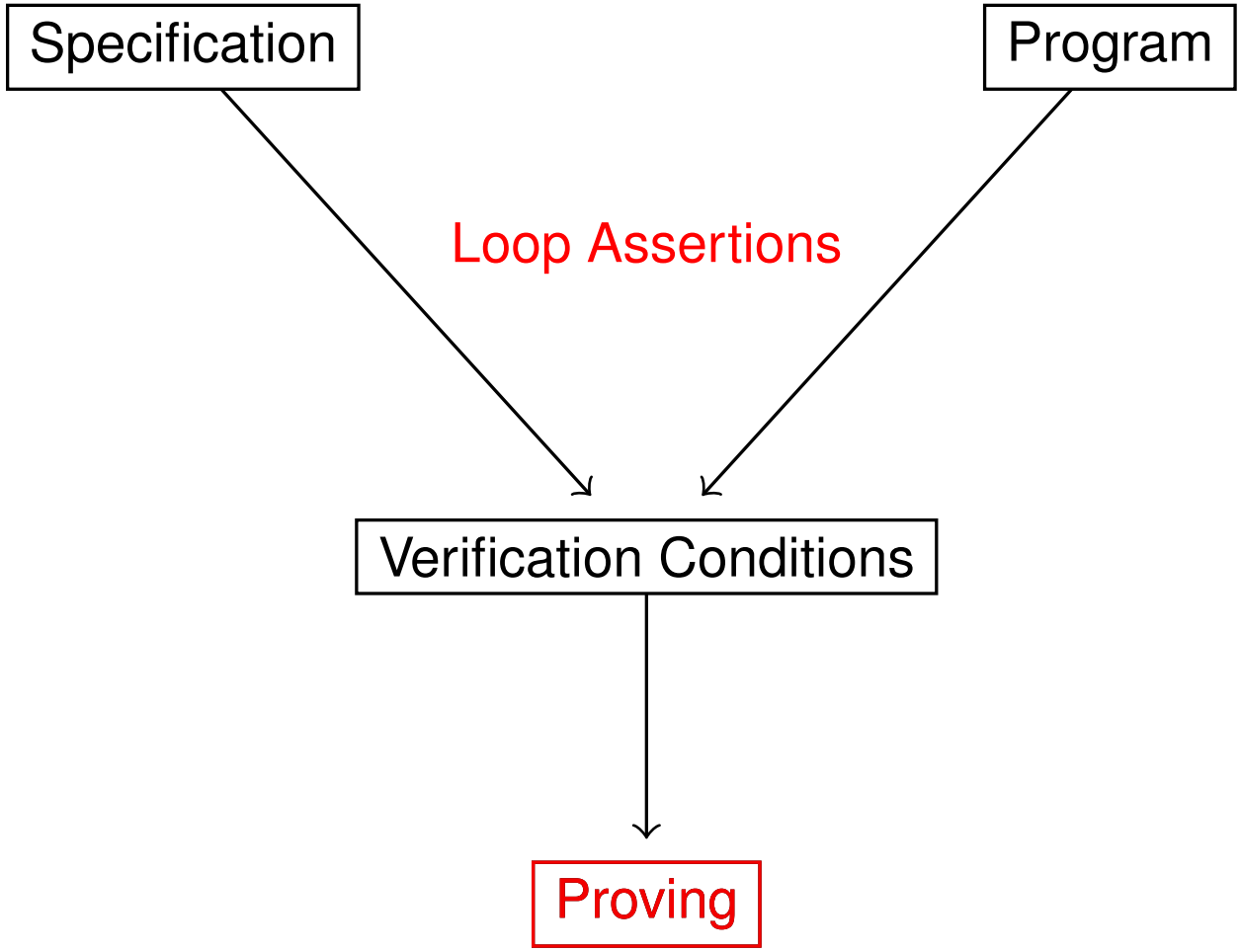
$(\exists q)(0 \le q < a \wedge B[p]=A[q]+h(p) \wedge A[q]>0)$

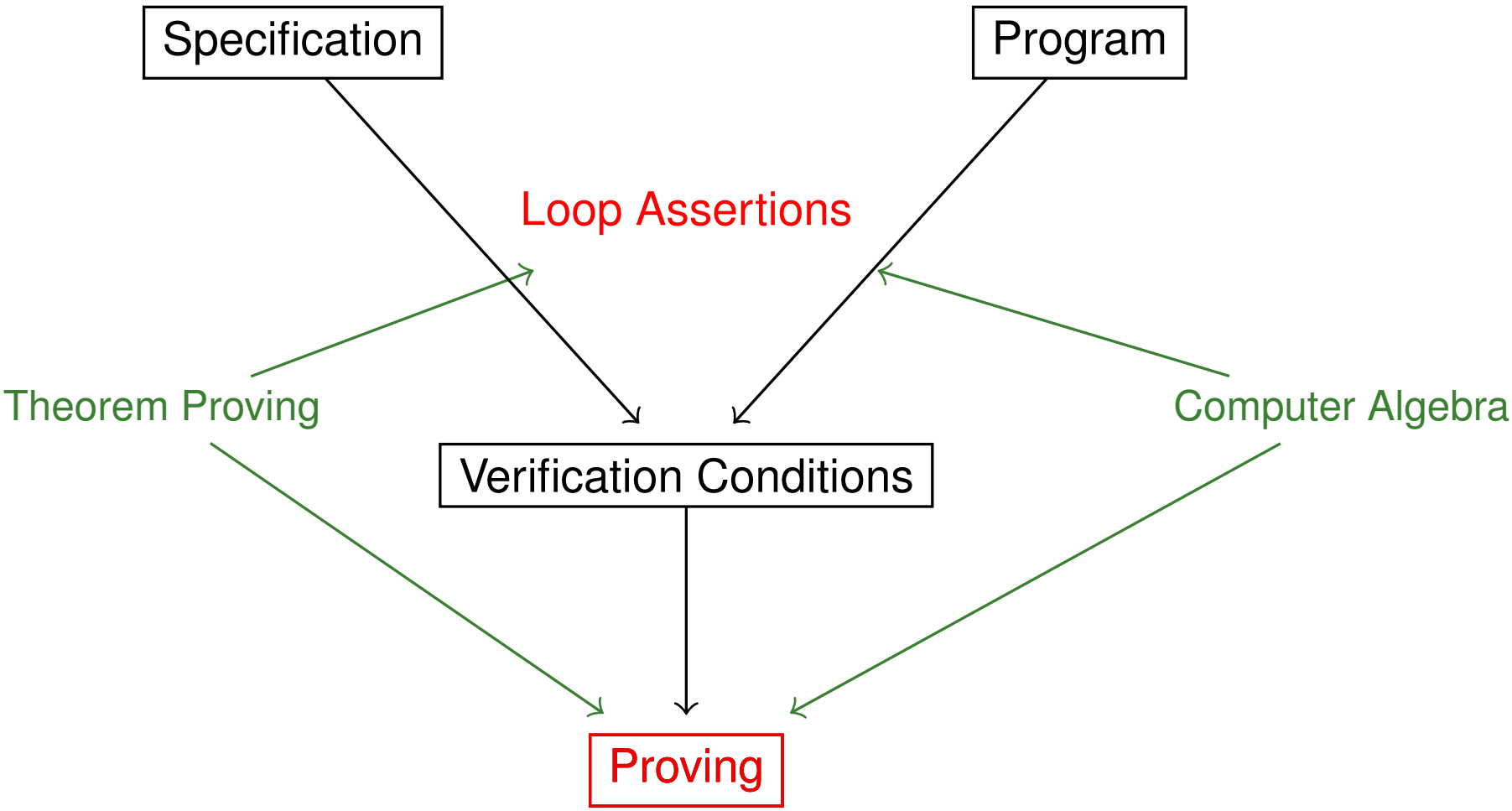

Symbolic Computation
Automated Reasoning
My Group
funded by:
European Research Council
Supporting top researchers from anywhere in the world
Knut and Alice Wallenberg Foundation
WALLENBERG ACADEMY FELLOWS
Vetenskapsrådet
Program Analysis

How do we tackle automated program analysis?

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).


Specification
Program
Loop Assertions
Verification Conditions
Proving


Specification
Program
Loop Assertions
Theorem Proving
Computer Algebra
Verification Conditions
Proving

Assertion Synthesis — Example: Array Partition

Program

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

while $(a < N)$ **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

Loop Assertions

$a = b + c$

$a \geq 0 \ \wedge \ b \geq 0 \ \wedge \ c \geq 0$

$a \leq N \quad \vee \quad N \leq 0$

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

$(\forall p)(0 \leq p < b \Rightarrow$

$\quad B[p] \geq 0 \ \wedge$

$\quad (\exists i)(0 \leq i < a \ \wedge \ A[a] = B[p]))$

Assertion Synthesis — Example: Array Partition

Program

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

while $(a < N)$ **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

Loop Assertions

Polynomial Equalities and Inequalities, Quantified FO properties

$$a = b + c$$

$$a \geq 0 \ \wedge \ b \geq 0 \ \wedge \ c \geq 0$$

$$a \leq N \quad \vee \quad N \leq 0$$

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

$$(\forall p)(0 \leq p < b \Rightarrow$$
$$B[p] \geq 0 \ \wedge$$
$$(\exists i)(0 \leq i < a \ \wedge \ A[a] = B[p]))$$

Our Approach

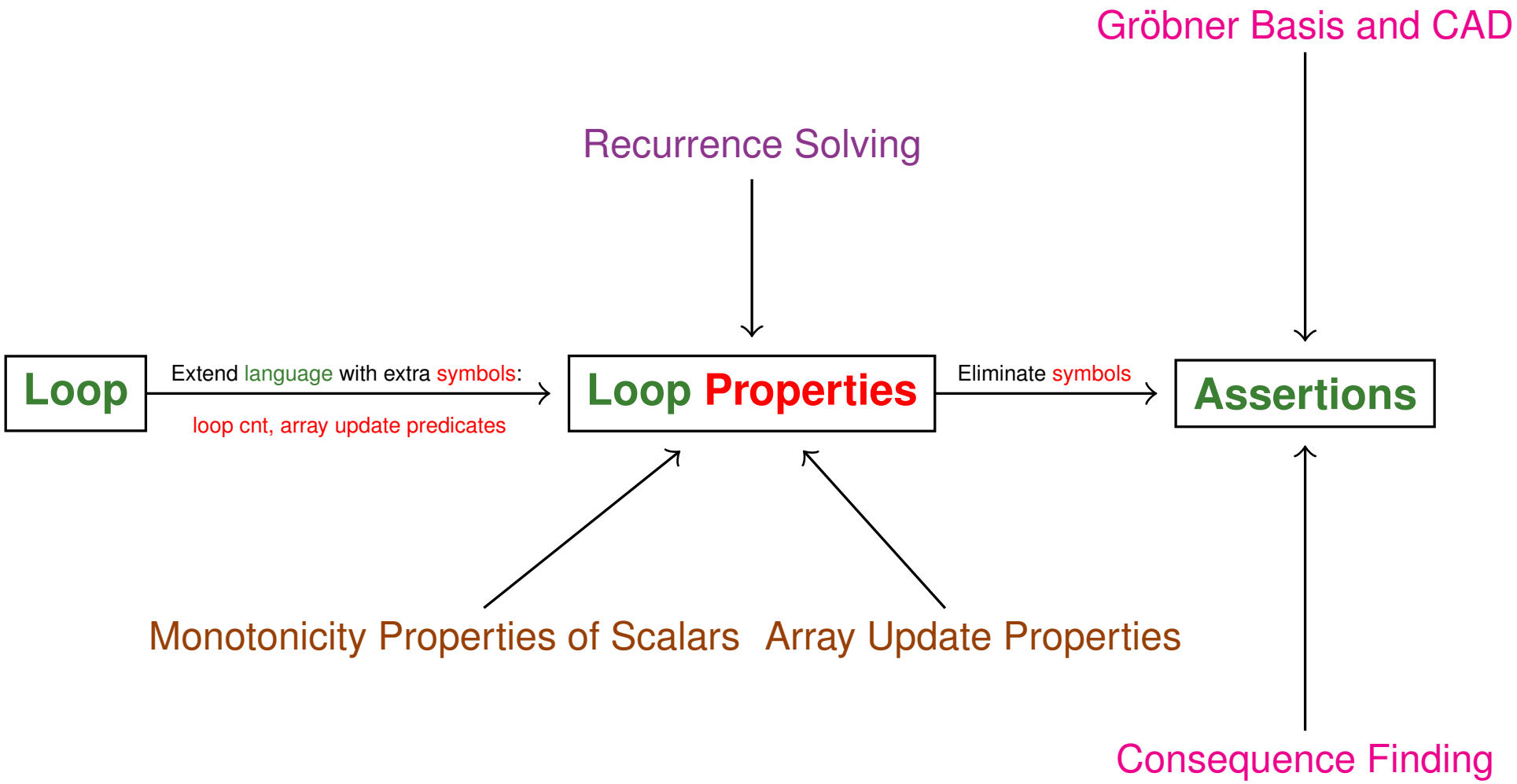
Loop

Assertions

Our Approach

Our Approach:
SYMBOL ELIMINATION

Our Approach:
SYMBOL ELIMINATION



Outline

Outline

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **end do**

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **end do**

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

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

$$\begin{cases} x^{(n+1)} &= 2 * x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} * y^{(n)} + 1 \end{cases}$$

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

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

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

$$\begin{cases} x^{(n)} &= a * x^{(0)} \\ y^{(n)} &= b * y^{(0)} - 2 * b + 2 \\ 0 &= a * b - 1 = 2^n * \frac{1}{2^n} - 1 \end{cases}$$

4. Eliminate n and variables standing for algebraically related exponentials in $n \rightarrow$
   $\uparrow$ elimination by Gröbner bases;

$$x * y - 2 * x + 2 = 0$$

5. Result: Polynomial Ideal $\rightarrow$ Finite basis

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **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

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

$$\begin{cases} x^{(n+1)} &= 2 * x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} * y^{(n)} + 1 \end{cases}$$

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

$$\begin{cases} x^{(n)} &= a * x^{(0)} \\ y^{(n)} &= b * y^{(0)} - 2 * b + 2 \\ 0 &= a * b - 1 = 2^n * \frac{1}{2^n} - 1 \end{cases}$$

$$x * y - 2 * x + 2 = 0$$

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **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

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

$$\begin{cases} x^{(n+1)} &= 2 * x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} * y^{(n)} + 1 \end{cases}$$

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

$$\begin{cases} x^{(n)} &= a * x^{(0)} \\ y^{(n)} &= b * y^{(0)} - 2 * b + 2 \\ 0 &= a * b - 1 = 2^n * \frac{1}{2^n} - 1 \end{cases}$$

$$x * y - 2 * x + 2 = 0$$

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **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

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

$$\begin{cases} x^{(n+1)} &= 2 * x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} * y^{(n)} + 1 \end{cases}$$

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

$$\begin{cases} x^{(n)} &= a * x^{(0)} \\ y^{(n)} &= b * y^{(0)} - 2 * b + 2 \\ 0 &= a * b - 1 = 2^n * \frac{1}{2^n} - 1 \end{cases}$$

$$x * y - 2 * x + 2 = 0$$

Polynomial Invariants

$x := 1;\ y := 0;$

while $\ldots$ **do** $x := 2 * x;\ y := \frac{1}{2} * y + 1$ **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

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

$$\begin{cases} x^{(n+1)} &= 2 * x^{(n)} \\ y^{(n+1)} &= \frac{1}{2} * y^{(n)} + 1 \end{cases}$$

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

$$\begin{cases} x^{(n)} &= a * x^{(0)} \\ y^{(n)} &= b * y^{(0)} - 2 * b + 2 \\ 0 &= a * b - 1 = 2^n * \frac{1}{2^n} - 1 \end{cases}$$

$$x * y - 2 * x + 2 = 0$$

Polynomial Invariants (TACAS08, ISSAC17)

Symbol elimination by Gröbner bases computation:

- Loops with assignments and nested conditionals.

  Structural constraints on assignments with polynomial rhs.

  $\leftarrow$ Recurrence solving (C-finite, Hypergeometric)

  $\leftarrow$ Restricted multiplication among variables;

  Tests are ignored $\rightarrow$ non-deterministic programs

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

  $\leftarrow$ Sound and complete - termination bound: #vars · #branches;

- Polynomial invariant ideals represented by Gröbner bases;

Polynomial Invariants (TACAS08, ISSAC17)

Symbol elimination by Gröbner bases computation:

- Loops with assignments and nested conditionals.

  Structural constraints on assignments with polynomial rhs.

  $\leftarrow$ Recurrence solving (C-finite, Hypergeometric)

  $\leftarrow$ Restricted multiplication among variables;

  Tests are ignored $\rightarrow$ non-deterministic programs

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

  $\leftarrow$ Sound and complete - termination bound: #vars $\cdot$ #branches;

- Polynomial invariant ideals represented by Gröbner bases;

Polynomial Invariants (TACAS08, ISSAC17)

Symbol elimination by Gröbner bases computation:

- Loops with assignments and nested conditionals.

  Structural constraints on assignments with polynomial rhs.

  $\leftarrow$ Recurrence solving (C-finite, Hypergeometric)

  $\leftarrow$ Restricted multiplication among variables;

  Tests are ignored $\rightarrow$ non-deterministic programs

- **A**utomated **L**oop **I**nvariant **G**eneration by **A**lgebraic **T**echniques **O**ver the **R**ationals for P-solvable Loops:

  $\leftarrow$ Sound and complete - termination bound: #vars $\cdot$ #branches;

- Polynomial invariant ideals represented by Gröbner bases;

- Implementation: `ALIGATOR` https://ahumenberger.github.io/aligator/

Example from Sharon's Talk

while $\ldots$ **do** $x := x + y;\ y := y + 2$ **end do**

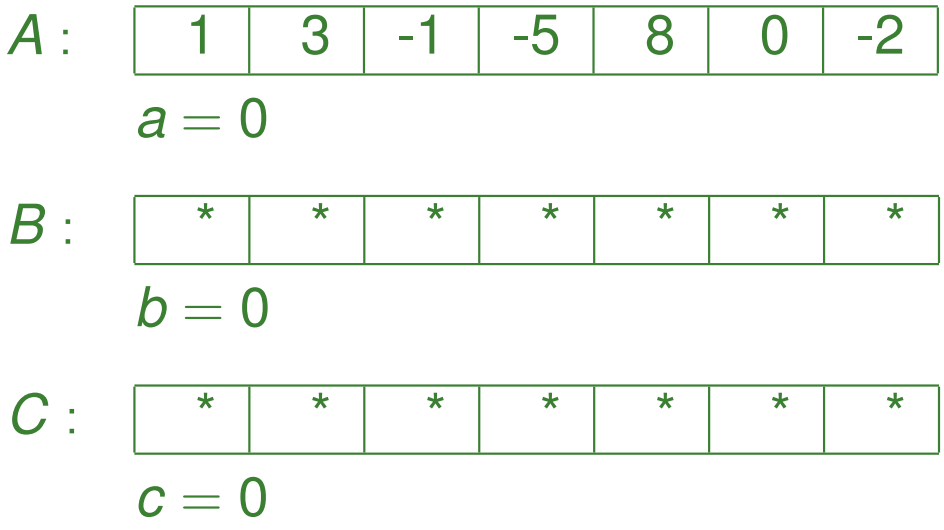
Inductive Polynomial Invariant:

$$4x - y^2 + 2y = 4x_0 - y_0^2 + 2 * y_0$$

Outline

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



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 :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

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 :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

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

$$(\forall p)(0 \leq p < b \Rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \wedge A[i] = B[p]))$$

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 :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall \exists$

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

$$(\forall p)(0 \leq p < b \Rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \ \wedge \ A[i] = B[p]))$$

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 :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

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

Quant. Invariants and Symbol Elimination

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

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: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

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

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

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

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

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

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

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

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

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \Rightarrow B^{(n)}[p] = x$$

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

Quant. Invariants and Symbol Elimination

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

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: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

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

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \Rightarrow B^{(n)}[p] = x$$

3. Eliminate symbols $\rightarrow$ Invariants

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

Quant. Invariants and Symbol Elimination

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

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: $iter, upd_V(i,p), upd_V(i,p,x)$

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

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \Rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Quant. Invariants and Symbol Elimination

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

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: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

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

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \Rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \Rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

3. Eliminate symbols $\rightarrow$ Invariants

HOW?

Quantified Invariants and Symbol Elimination

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

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

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

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

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

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

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

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

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

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \Rightarrow B^{(n)}[p] = x$$

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

$$\xrightarrow[\text{Theorem Proving}]{\text{Saturation}} I_1, I_2, I_3, I_4, I_5, \ldots$$

Symbol Elimination by First-Order Theorem Proving

1. Reasoning in first-order theories

$$x \geq y \iff x > y \vee x = y$$
$$x > y \Rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \Rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols → INVARIANTS

Symbol Elimination by First-Order Theorem Proving

1. Reasoning in first-order theories

$$x \geq y \iff x > y \vee x = y$$
$$x > y \Rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \Rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols → INVARIANTS

- For every loop variable v → TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- 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

$$x \geq y \iff x > y \vee x = y$$
$$x > y \Rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \Rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ INVARIANTS

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

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols:
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- Reduction (elimination) ordering $\succ$:

  useless symbols $\succ$ usable symbols.

Quantified Invariants and Symbol Elimination

(CAV13, LPAR15)

Symbol elimination by saturation-based theorem proving:

(joint work with Y. Chen, S. Robillard and A. Voronkov)

- Express loop properties in a language containing extra symbols;
- Every logical consequence of these properties is a valid loop property, but not an invariant;
- Run a saturation theorem prover for eliminating extra symbols;
- Derived formulas in the language of the loop are loop invariants.

Quantified Invariants and Symbol Elimination

(CAV13, LPAR15)

Symbol elimination by saturation-based theorem proving:

(joint work with Y. Chen, S. Robillard and A. Voronkov)

- Express loop properties in a language containing extra symbols;
- Every logical consequence of these properties is a valid loop property, but not an invariant;
- Run a saturation theorem prover for eliminating extra symbols;
- Derived formulas in the language of the loop are loop invariants.

Implementation: `Vampire` http://vprover.org

Symbol Elimination for Program Analysis

New Extensions in Theorem Proving

- First class boolean sort
  - example: boolean arrays, boolean flags;
- Theory of polymorphic arrays
  - example: programs with integer arrays and boolean arrays;
- If-then-Else and Let-in constructs
  - example: partial correctness of programs with nested conditionals.

Symbol Elimination for Program Analysis

New Extensions in Theorem Proving

- First class boolean sort
  - example: boolean arrays, boolean flags;
- Theory of polymorphic arrays
  - example: programs with integer arrays and boolean arrays;
- If-then-Else and Let-in constructs
  - example: partial correctness of programs with nested conditionals.

Symbol Elimination for Program Analysis

New Extensions in Theorem Proving

- First class boolean sort
  - example: boolean arrays, boolean flags;
- Theory of polymorphic arrays
  - example: programs with integer arrays and boolean arrays;
- If-then-Else and Let-in constructs
  - example: partial correctness of programs with nested conditionals.

Symbol Elimination for Program Analysis (CPP16)

New Extensions in Theorem Proving: **the FOOL Logic**

(joint work with E. Kotelnikov, M. Suda, A. Voronkov)

- First class boolean sort
  - example: boolean arrays, boolean flags;
- Theory of polymorphic arrays
  - example: programs with integer arrays and boolean arrays;
- If-then-Else and Let-in constructs
  - example: partial correctness of programs with nested conditionals.

Cookies for Program Analysis (CPP16)

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r[a]) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

Cookies for Program Analysis (CPP16)

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r[a]) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

The next state function for `a`:

```
a1 =
  if r[a]
  then let a=a+1 in a
  else let a=a+q(a) in a
```

Cookies for Program Analysis (CPP16)

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r[a]) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

The next state function for `a`:

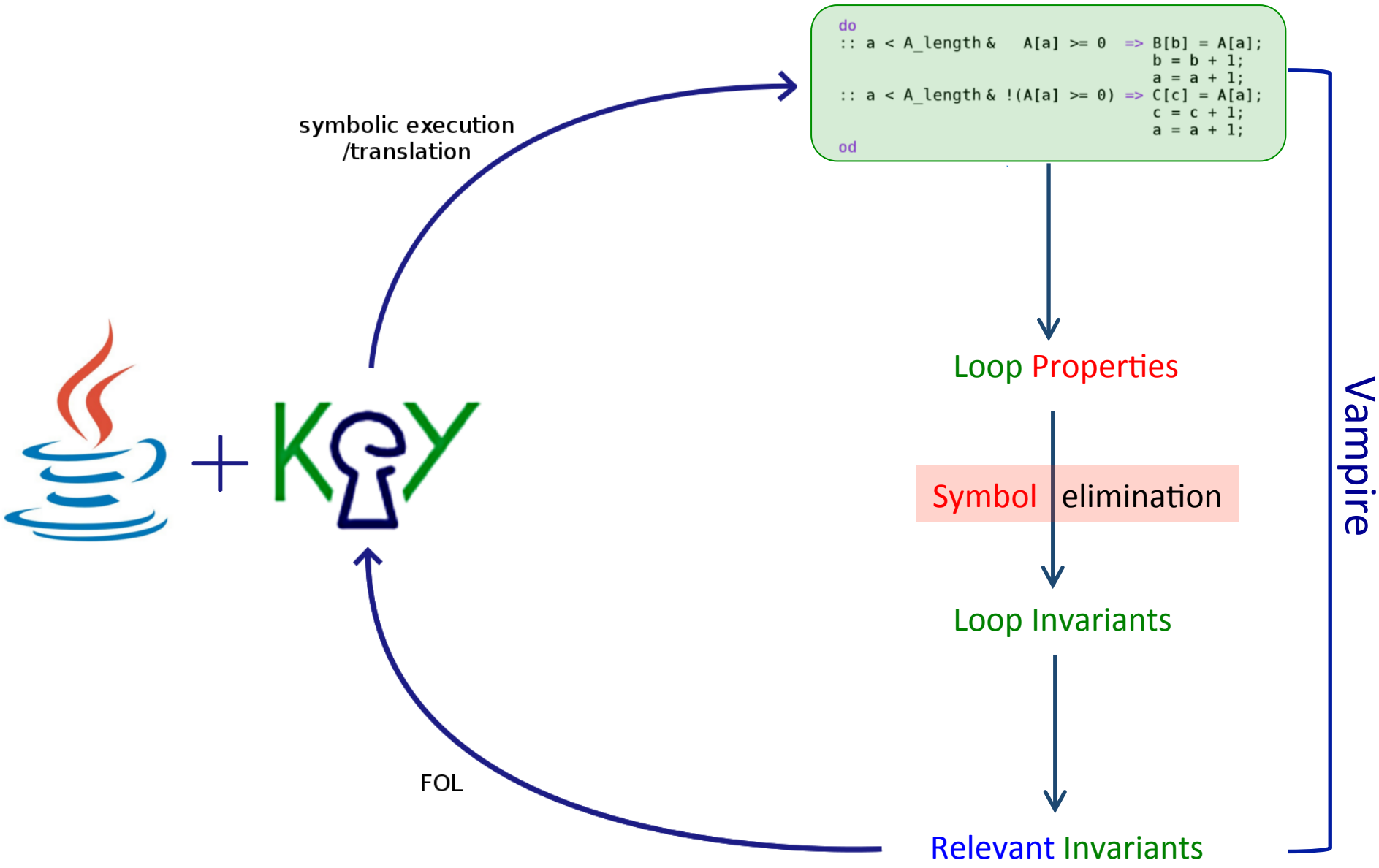
```
a1 =
  if r[a]
  then let a=a+1 in a
  else let a=a+q(a) in a
```

In Vampire:

```
thf(1,type,p : $int > $o).
thf(2,type,q : $int > $int).
thf(3,type,r : $array($int, $o)).
thf(4,type,a : $int).
thf(5,type,a1: $int).

thf(6,hypothesis,! [X:$int] :
  (p(X) => $greatereq(X,0))).
thf(7,hypothesis,! [X:$int] :
  ($greater(q(X),0))).
thf(8,hypothesis,p(a)).

thf(9,hypothesis,
  a1 = $ite($select(r,a),
    $let(a:=$sum(a,1),a),
    $let(a:=$sum(a,q(a)),a)
)).

tff(10,conjecture,$greater(a1,0)).
```

Symbol Elimination in Program Verification (LPAR15)

(joint work with W. Ahrendt and S. Robillard)



Conclusions and Further Work

Symbol Elimination for Program Analysis

- Combines symbolic computation and saturation theorem proving;
- Fully automatic;
- Polynomial properties with quantifier alternations.

Further Work

- Programs with more complex flow and/or operations over the heap;
- Inductive reasoning in first-order proving.

Conclusions and Further Work

Symbol Elimination for Program Analysis

- Combines symbolic computation and saturation theorem proving;
- Fully automatic;
- Polynomial properties with quantifier alternations.

Further Work

- Programs with more complex flow and/or operations over the heap;
- Inductive reasoning in first-order proving.