Experiments with Invariant Generation Using a Saturation Theorem Prover

Laura Kovács

TU Vienna, Austria

joint work with Kryštof Hoder and Andrei Voronkov

Loop	Symbol Elimination within 1s					Symbol Elimination within 10s				
	♯ SEI	♯ Min SEI	% Redundancy	∀-Inv	∀∃-Inv	♯ SEI	♯ Min SEI	% Redund	∀-Inv	∀∃-Inv
`Initialisation`	15	5	67%	yes	no	40	5	88%	yes	no
`Copy`	24	5	79%	yes	no	25	5	80%	yes	no
`Find`	151	13	91%	yes	no	474	21	96%	yes	no
`Vararg`	1	1	0%	yes	no	1	1	0%	yes	no
`Partition`	166	38	77%	yes	yes	849	59	93%	yes	yes
`Partition_Init`	168	24	86%	yes	yes	692	127	82%	yes	yes

Table: Vampire with 1 and 10 seconds of time limit on challenging benchmarks.

Loop Shape	♯ of Loops	Average ♯ of SEI	Average ♯ of Non-Redundant SEI	% of SEI Redundancy	∀-Inv	∀∃-Inv
Simple	33	168	18	89.3%	yes	no
Multi-path	5	340	46	86.4%	yes	yes

Table: Vampire on industrial benchmarks of Dassault Aviation.

Array Copy

Program

$c := 0;\ d := 0;$

while $(c < N)$ **do**

$\quad C[c] := D[d];$

$\quad c := c + 1;$

$\quad d := d + 1$

end do

Invariants

$c = d$

$c \leq N \quad \vee \quad N \leq 0$

$(\forall k)\ (0 \leq k < c \rightarrow C[k] = D[k])$

Array Copy

Program and Specification

$\texttt{Pre}: N > 0 \quad \wedge$

$\qquad (\forall k)\ (0 \leq k < N \rightarrow D[k] = 0)$

$c := 0;\ d := 0;$

while $(c < N)$ **do**

$\qquad C[c] := D[d];$

$\qquad c := c + 1;$

$\qquad d := d + 1$

end do

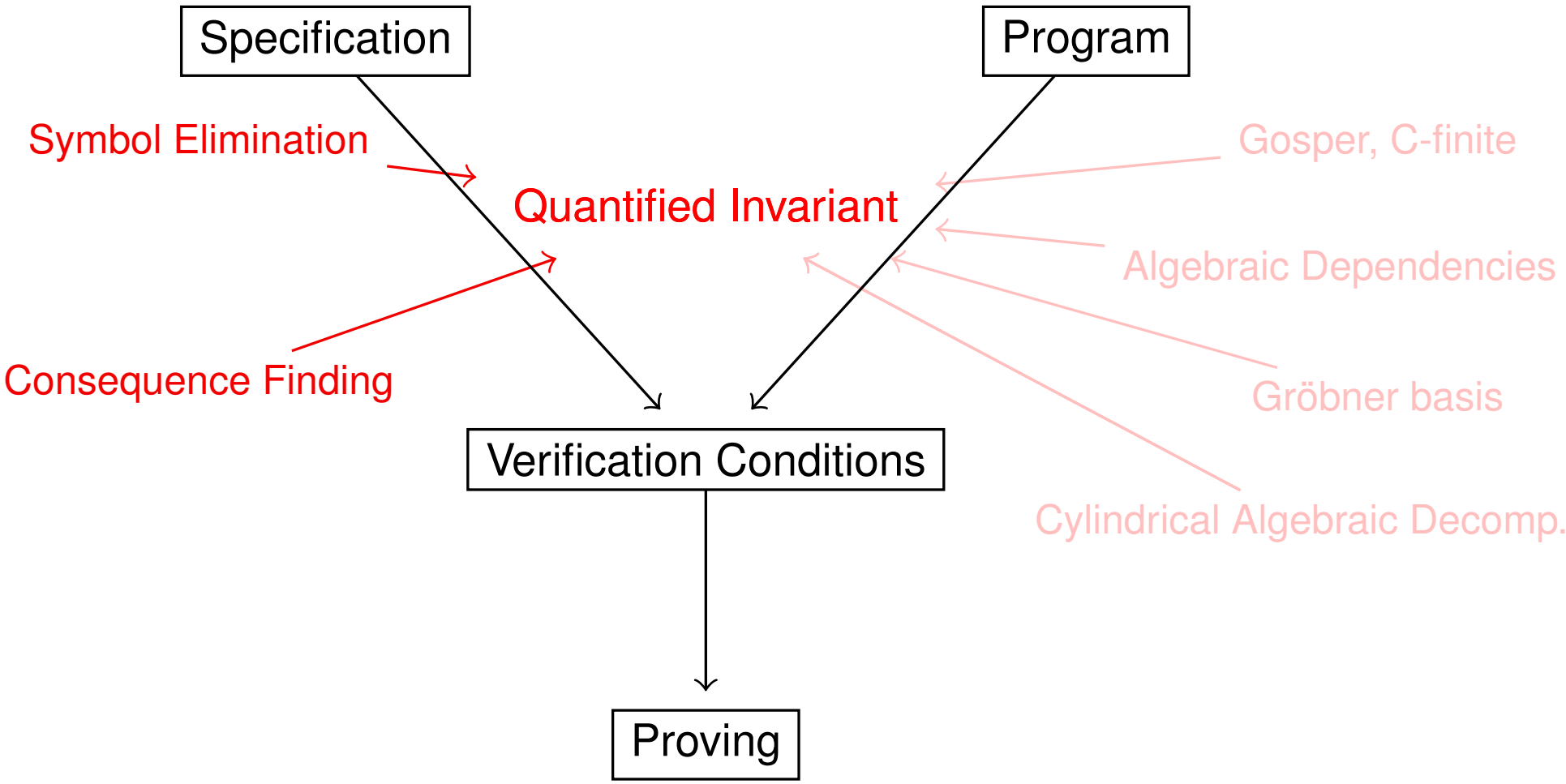
$\texttt{Post}: (\forall k)(0 \leq k < N \rightarrow C[k] = 0)$

Invariants

$c = d$

$c \leq N \quad \vee \quad N \leq 0$

$(\forall k)\ (0 \leq k < c \rightarrow C[k] = D[k])$


Specification
Program
Symbol Elimination
Gosper, C-finite
Quantified Invariant
Algebraic Dependencies
Consequence Finding
Gröbner basis
Verification Conditions
Cylindrical Algebraic Decomp.
Proving

Example: Array Partition

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

Example: Array Partition

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

$a = 0$

B :

*	*	*	*	*	*	*

$b = 0$

C :

*	*	*	*	*	*	*

$c = 0$

Example: Array Partition

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

$c = 3$

Example: Array Partition

$a := 0; b := 0; c := 0;$
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 $A[0], \ldots, A[a-1]$.

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

Example: Array Partition

$a := 0; b := 0; c := 0;$
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 $A[0], \ldots, A[a-1]$.

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

Example: Array Partition

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

Example: Array Partition - Vampire Experiments

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

1. B doesn't change at positions after final value of b (1s):

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

2. Each $B[0], \ldots, B[b-1]$ is a positive value in $\{A[0], \ldots, A[a-1]\}$ (1s):

$$\forall p(b > p \wedge p \geq 0 \rightarrow B[p] \geq 0 \wedge \exists k(a > k \wedge k \geq 0 \wedge A[k] = B[p])$$

Outline

Symbol Elimination and Invariant Generation

Experiments with Invariant Generation in Vampire

Programming Model

$$\begin{array}{lcl} G_1 & \to & \alpha_{11}; \ldots; \alpha_{1m} \\ & \vdots & \\ G_p & \to & \alpha_{p1}; \ldots; \alpha_{pm} \end{array}$$

with

$$G_i \wedge G_j \to \mathit{false}$$
$$G_1 \vee \cdots \vee G_p \to \mathit{true}$$

if for some i $\begin{cases} \alpha_{ik} : A[e_1] := e_2 \\ \alpha_{il} : A[e_3] := e_4 \end{cases}$ then $e_1 \neq e_3$

$$\begin{array}{rcl} A[a] \geq 0 & \to & B[b] := A[a];\ b := b+1;\ a := a+1 \\ \neg A[a] \geq 0 & \to & C[c] := A[a];\ c := c+1;\ a := a+1. \end{array}$$

Notation: Loop counter n

Programming Model

$$\begin{array}{lcl} G_1 & \rightarrow & \alpha_{11}; \ldots; \alpha_{1m} \\ & \vdots & \\ G_p & \rightarrow & \alpha_{p1}; \ldots; \alpha_{pm} \end{array}$$

with

$$G_i \wedge G_j \rightarrow \mathit{false}$$
$$G_1 \vee \cdots \vee G_p \rightarrow \mathit{true}$$

$$\text{if for some } i \begin{cases} \alpha_{ik} : A[e_1] := e_2 \\ \alpha_{il} : A[e_3] := e_4 \end{cases} \quad \text{then } e_1 \neq e_3$$

$$\begin{array}{rcl} A[a] \geq 0 & \rightarrow & B[b] := A[a];\ b := b+1;\ a := a+1 \\ \neg A[a] \geq 0 & \rightarrow & C[c] := A[a];\ c := c+1;\ a := a+1. \end{array}$$

Notation: Loop counter n

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

- $upd_V(i,p)$: at iteration i, V is updated at position p;
- $upd_V(i,p,x)$: at iteration i, V is updated at position p by value x.

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$
2. Collect loop properties:

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

2. Collect loop properties:
   - Polynomial scalar properties

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Increasing/decreasing (strictly): $(\forall i \in iter)(v^{(i+1)} \geq v^{(i)})$
   - Dense: $(\forall i \in iter)(v^{(i+1)} = v^{(i)} \vee v^{(i+1)} = v^{(i)} + 1)$

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}$, $i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter$, $upd_V(i,p)$, $upd_V(i,p,x)$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
     - Stability
       no array update at $p \rightarrow$ (final) value of $V[p]$ is unchanged
     - Last update
       $V[p]$ updated at iteration i and no further $\rightarrow$ final value $V[p]$

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

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

3. Eliminate symbols $\rightarrow$ Invariants

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation - The Method

```
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:
   - variables as functions of n: $v^{(i)}, i :: F$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

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

3. Eliminate symbols $\rightarrow$ Invariants

   Symbol Elimination

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

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

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

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

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

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

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

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

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

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

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

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

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

Invariant Generation by 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 Vampire

1. Reasoning with quantifiers and linear arithmetic [Korovin&V07]

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

- 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 (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

- 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 (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

- 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 (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

- 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 (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses: $x + y = y + x$ is not useful
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

- 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 (variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by Vampire;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

3. Turning USEFUL clauses into INVARIANTS
   - Replace v' by v for all loop variables v;
   - No skolem functions in the clause $\rightarrow$ Invariant;

$$\neg p \geq b' \vee B'[p] = B_0[p] \quad \rightarrow \quad \forall p(p \geq b \rightarrow B[p] = B_0[p])$$

   - Skolem function in the clause $\rightarrow$ (De-skolemise) $\rightarrow$ Invariant.

$$\neg b' > p \vee \neg p \geq 0 \vee A'[\$i(p)] = B'[p]$$
$$\downarrow$$
$$\forall p(b > p \wedge p \geq 0 \rightarrow (\exists k)A[k] = B[p])$$

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

3. Turning USEFUL clauses into INVARIANTS
   - Replace v' by v for all loop variables v;
   - No skolem functions in the clause $\rightarrow$ Invariant;

     $\neg p \geq b' \vee B'[p] = B_0[p] \quad \rightarrow \quad \forall p(p \geq b \rightarrow B[p] = B_0[p])$
   - Skolem function in the clause $\rightarrow$ (De-skolemise) $\rightarrow$ Invariant.

     $\neg b' > p \vee \neg p \geq 0 \vee A'[\$i(p)] = B'[p]$

     $\downarrow$

     $\forall p(b > p \wedge p \geq 0 \rightarrow (\exists k)A[k] = B[p])$

Symbol Elimination by Vampire

1. Reasoning with quantifiers and linear arithmetic

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$s(x) > x$$
$$x \geq s(y) \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses

3. Turning USEFUL clauses into INVARIANTS
   - Replace v' by v for all loop variables v;
   - No skolem functions in the clause $\rightarrow$ Invariant;

     $\neg p \geq b' \vee B'[p] = B_0[p] \quad \rightarrow \quad \forall p(p \geq b \rightarrow B[p] = B_0[p])$
   - Skolem function in the clause $\rightarrow$ (De-skolemise) $\rightarrow$ Invariant.

     $\neg b' > p \vee \neg p \geq 0 \vee A'[\$i(p)] = B'[p]$

     $\downarrow$

     $\forall p(b > p \wedge p \geq 0 \rightarrow (\exists k)A[k] = B[p])$

Example: Array Partition - Vampire Experiments

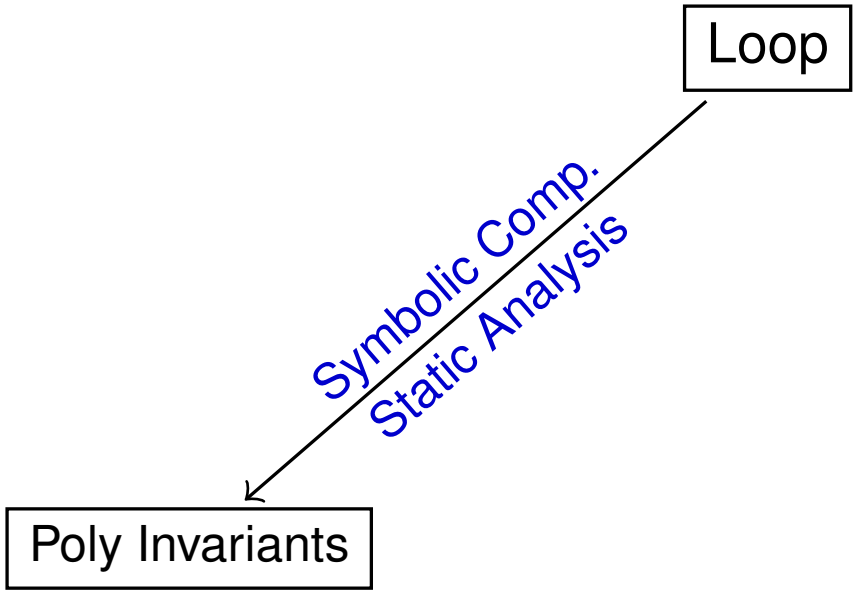
$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

1. B doesn't change at positions after final value of b (1s):

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

2. Each $B[0], \ldots, B[b-1]$ is a positive value in $\{A[0], \ldots, A[a-1]\}$ (1s):

$$\forall p(b > p \wedge p \geq 0 \rightarrow B[p] \geq 0 \wedge \exists k(a > k \wedge k \geq 0 \wedge A[k] = B[p])$$



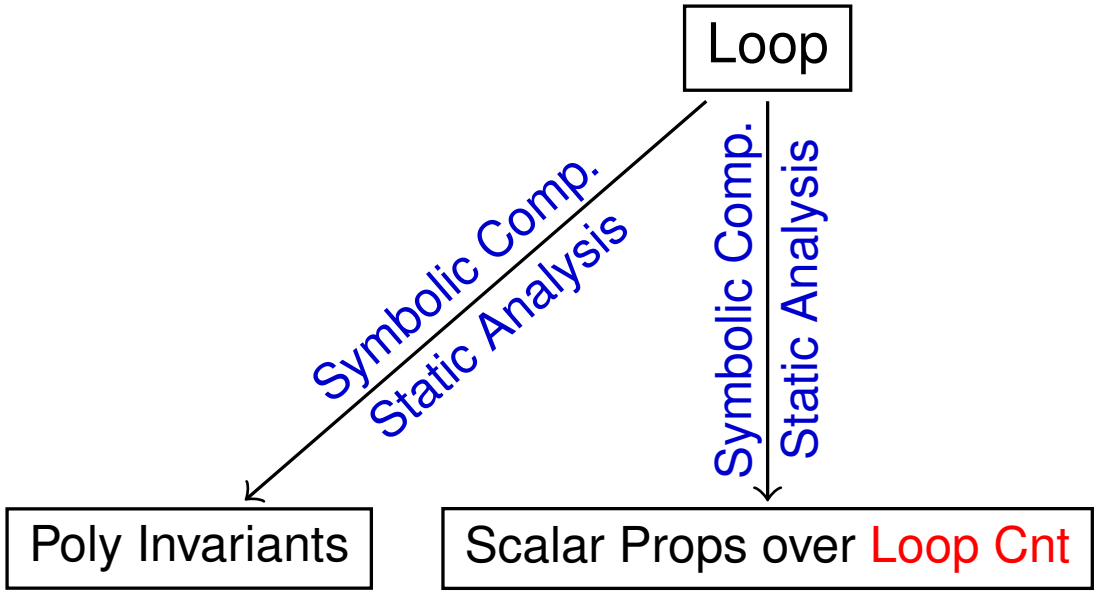

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

$a = b + c$

$b \geq 0$

$c \geq 0$

$a \geq 0$




```
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
```

$a = b + c$

$b \geq 0$

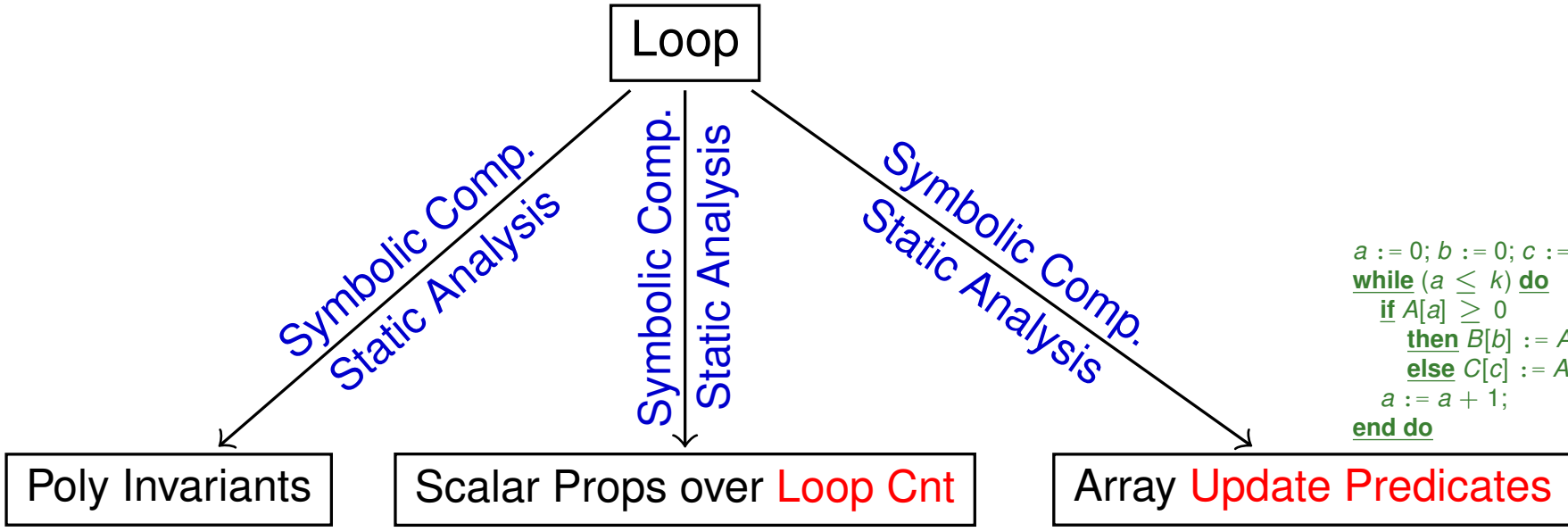
$c \geq 0$

$a \geq 0$

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

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

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




```
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
```

$a = b + c$

$b \geq 0$

$c \geq 0$

$a \geq 0$

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

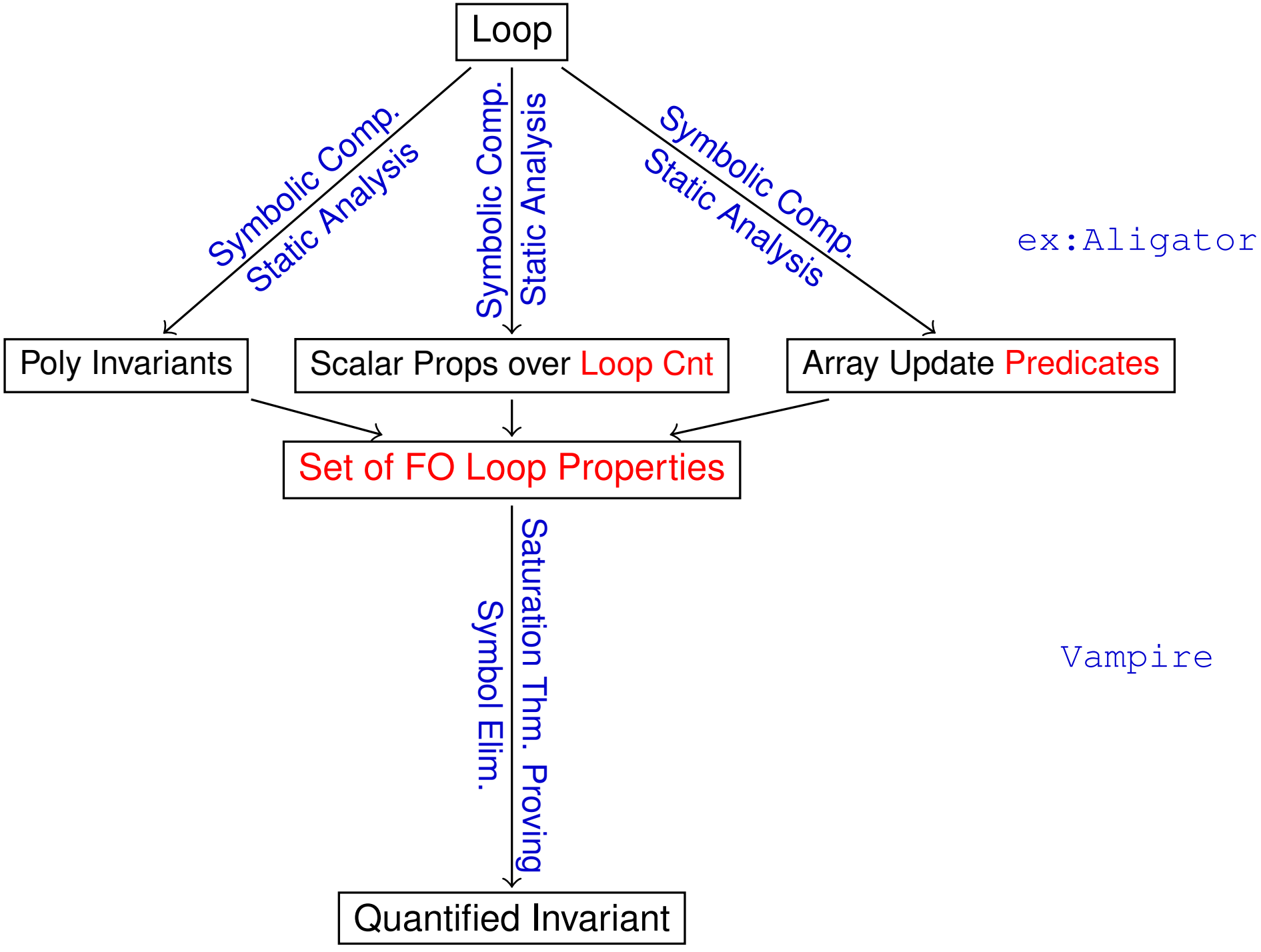
$(\forall j, k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$

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

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

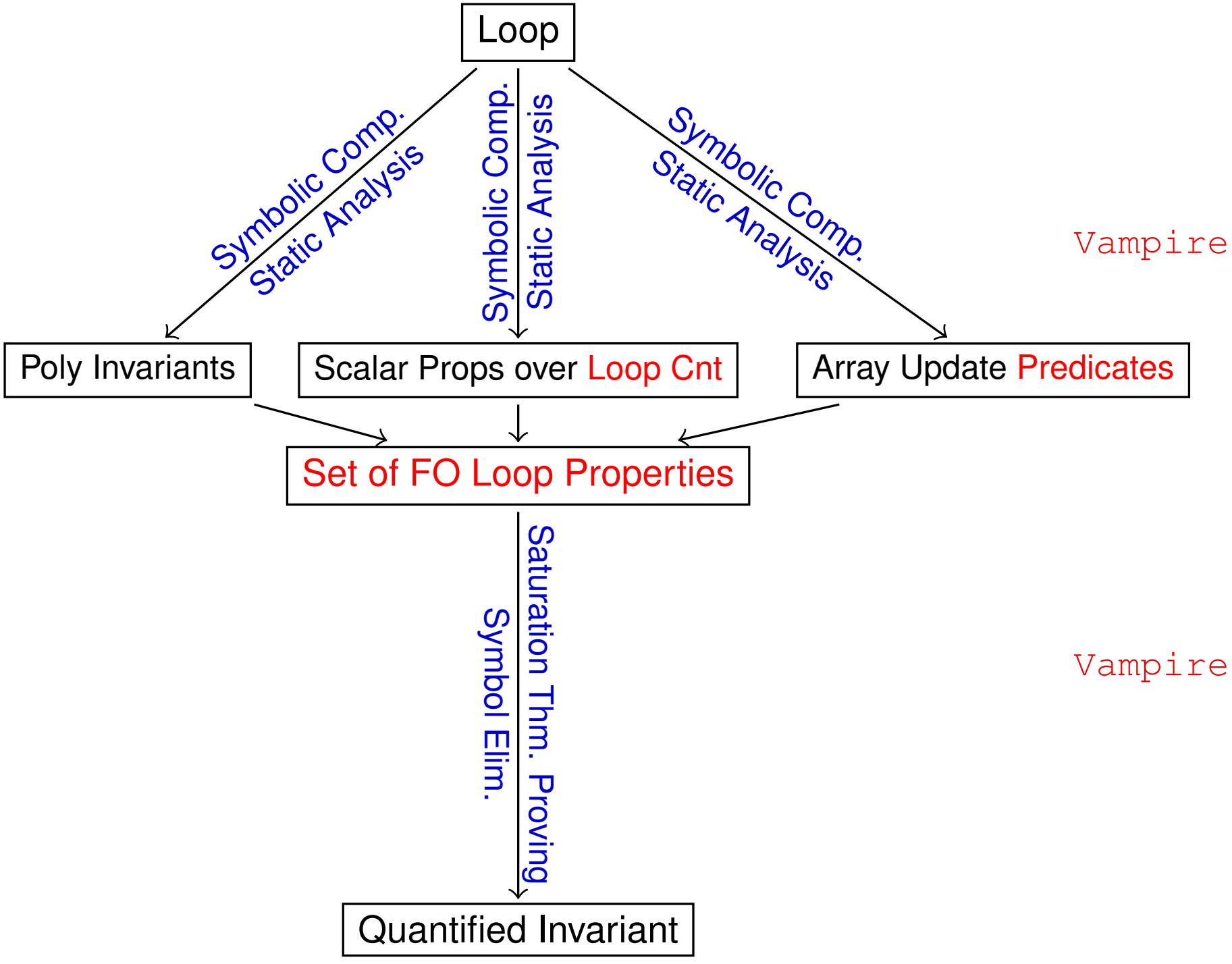
$$(\forall v)(b^{(0)} \leq v < b^{(n)} \Rightarrow (\exists i \in iter)(b^{(i)} = v \wedge (A[a^{(i)}] \geq 0))$$

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

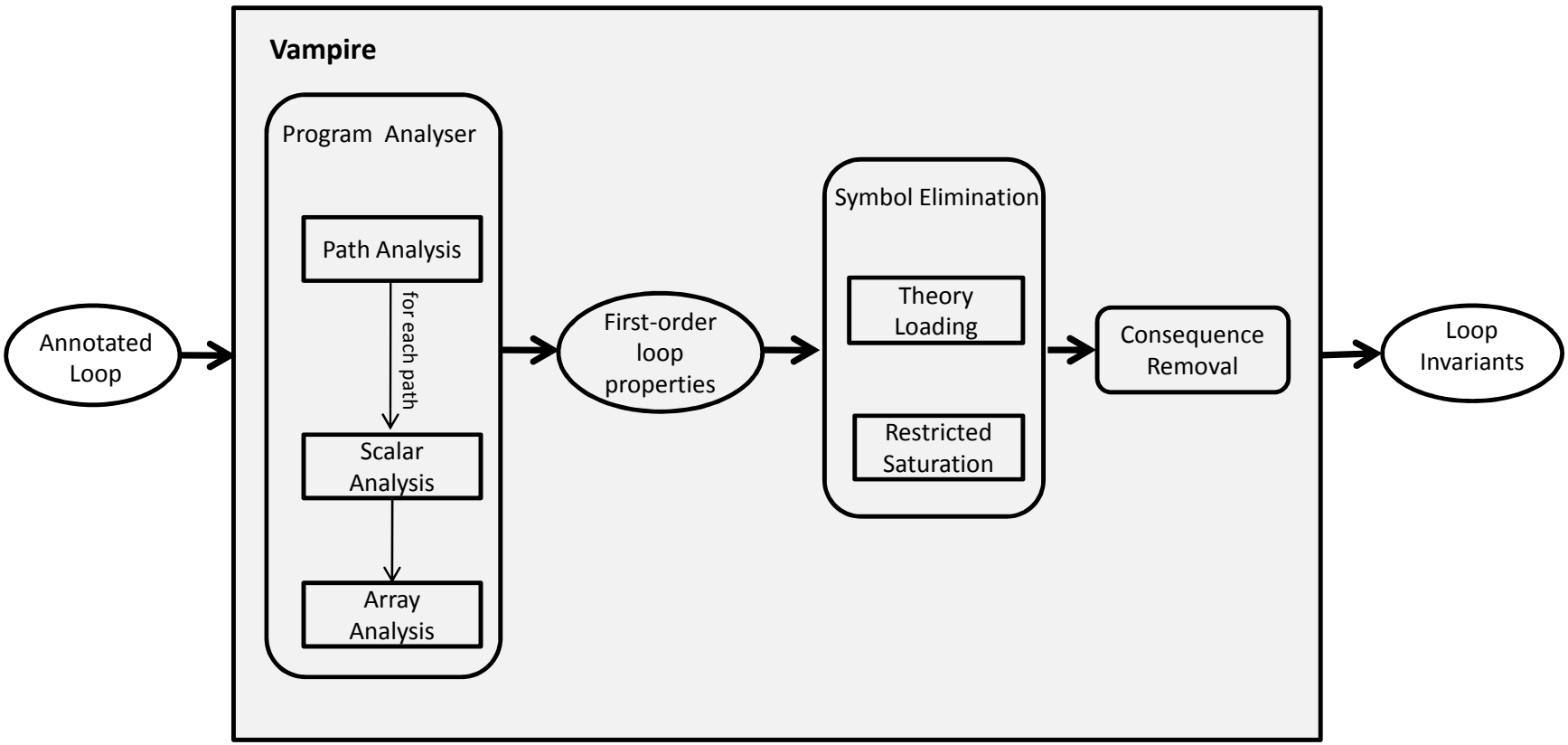


ex:Aligator

Vampire

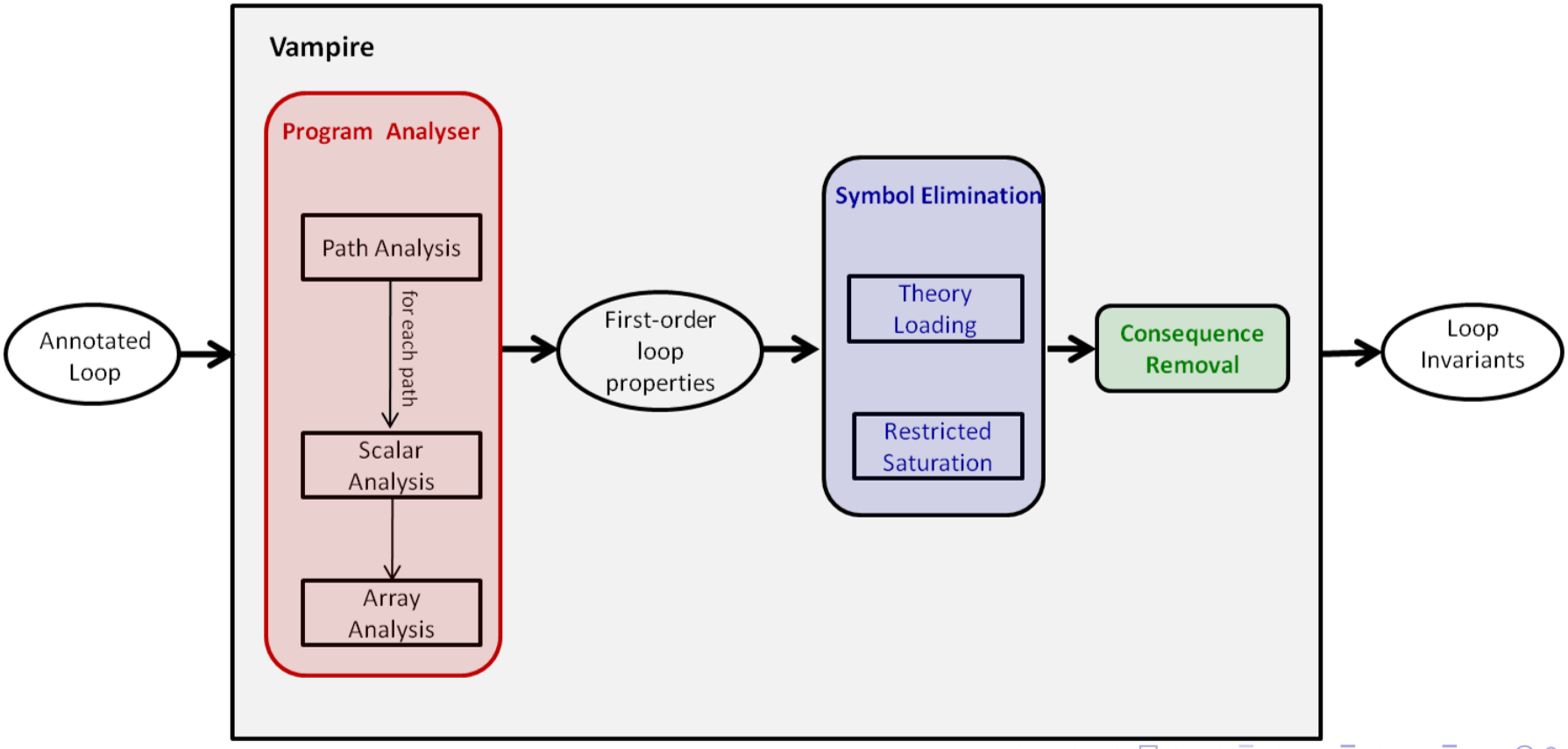

Loop
Symbolic Comp.
Static Analysis
Symbolic Comp.
Static Analysis
Symbolic Comp.
Static Analysis
Vampire
Poly Invariants
Scalar Props over Loop Cnt
Array Update Predicates
Set of FO Loop Properties
Saturation Thm. Proving
Symbol Elim.
Vampire
Quantified Invariant

Experiments with Invariant Generation in Vampire



Experiments with Invariant Generation in Vampire

1. Program analysis (new Vampire mode);
2. Theory loading (new Vampire opion);
3. Elimination of “colored” symbols (new Vampire option);
4. Generation of “minimal” set of invariants (new Vampire mode).



1. Sample Output for Program Analysis in Vampire

```
vampire --mode program_analysis
```

```
Loops found: 1
Analyzing loop...
--------------------
while (a < m)
{
if (A[a] >= 0)
{
B[b] = A[a]; b = b + 1;
}
else
{
C[c] = A[a]; c = c + 1;
}
a = a + 1;
}
--------------------
Variable: B: (updated)
Variable: a: (updated)
Variable: b: (updated)
Variable: m: constant
Variable: A: constant
Variable: C: (updated)
Variable: c: (updated)
Counter: a
Counter: b
Counter: c
```

```
Path:
false: A[a] >= 0
C[c] = A[a];
c = c + 1;
a = a + 1;
Path:
true: A[a] >= 0
B[b] = A[a];
b = b + 1;
a = a + 1;
Counter a: 1 min, 1 max, 1 gcd
Counter b: 0 min, 1 max, 1 gcd
Counter c: 0 min, 1 max, 1 gcd
7. ![X1,X0,X3]:(X1>X0 & c(X1)>X3 & X3>c(X0)) =>
   ?[X2]:(c(X2)=X3 & X2>X0 & X1>X2) [program analysis]
6. ![X0]:c(X0)>=c0 (0:4) [program analysis]
5. ![X0]:c(X0)<=c0+X0 (0:6) [program analysis]
4. ![X1,X0,X3]:(X1>X0 & b(X1)>X3 & X3>b(X0))
   => ?[X2]:(b(X2)=X3 & X2>X0 & X1>X2) [program
analysis]
3. ![X0]:b(X0)>=b0 (0:4) [program analysis]
2. ![X0]:b(X0)<=b0+X0 (0:6) [program analysis]
1. ![X0]:a(X0)=a0+X0 (0:6) [program analysis]
```

Figure: Partial output of Vampire's program analyser on the `Partition` program.

2. Theory Loading in Vampire

We use incomplete but sound theory axiomatisation.

Example: Integers in Vampire

- 0, 1, 2, etc;
- Integer predicates/funcions:
  - addition;
  - subtraction;
  - multiplication;
  - successor;
  - division;
  - inequality relations;
- Automated theory loading:

```
vampire(interpreted_symbol, geq, integer_greater_equal)
```

2. Theory Loading in Vampire

```
vampire(option, theory_axioms, on).
```

```
vampire --theory_axioms on
```

We use incomplete but sound theory axiomatisation.

Example: Integers in Vampire

- 0, 1, 2, etc;
- Integer predicates/funcions:
  - addition;
  - subtraction;
  - multiplication;
  - successor;
  - division;
  - inequality relations;
- Automated theory loading:

```
vampire(interpreted_symbol, geq, integer_greater_equal)
```

3. Elimination of Colored Symbols

vampire(option,show_symbol_elimination,on).

```
vampire(option,time_limit,1).

vampire(option, theory_axioms,on).
vampire(interpreted_symbol,geq, integer_greater_equal).
vampire(interpreted_symbol,greater, integer_greater).
vampire(interpreted_symbol,plus,integer_plus).
vampire(interpreted_symbol,s,successor).
vampire(interpreted_symbol,minus,integer_minus).

vampire(symbol,function,n,0,left).
vampire(symbol,function,a,1,left).
vampire(symbol,function,b,1,left).
vampire(symbol,function,c,1,left).
vampire(symbol,function,bb,2,left).
vampire(symbol,function,cc,2,left).
vampire(symbol,predicate,updB,2,left).
vampire(symbol,predicate,updB,3,left).
vampire(symbol,predicate,updC,2,left).
vampire(symbol,predicate,updC,3,left).
vampire(symbol,predicate,iter,1,left).
vampire(symbol,predicate,geq,2,skip).

vampire(symbol,predicate,greater,2,skip).
vampire(symbol,function,s,1,skip).
vampire(symbol,function,zero,0,skip).
vampire(symbol,function,plus,2,skip).
vampire(symbol, function, minus,2,skip).
vampire(symbol,function,a,0,skip).
vampire(symbol,function,b,0,skip).
vampire(symbol,function,c,0,skip).
vampire(symbol,function,m,0,left).
vampire(symbol,function,aa,1,skip).
vampire(symbol,function,bb0,2,skip).
vampire(symbol,function,bb0,1,skip).
vampire(symbol,function,cc0,2,skip).
vampire(symbol,function,cc0,1,skip).
```

Figure: Partial input for symbol elimination in Vampire.

4. Generation of Minimal Set of Invariants

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

Experiments: Invariant Generation in Vampire

- Challenging academic benchmarks
- Industrial Benchmarks

Loop	♯ SEI	♯ Min SEI	Inv of interest	Generated invariants implying Inv

`Copy`
$a = 0;$
while $(a < m)$ **do**
$B[a] = A[a]; a = a + 1$
end do
	24	5	$\forall x: \quad 0 \leq x < a \rightarrow B[x] = A[x]$	`inv8`:
				$\forall x_0, x_1: \quad A[x_0] = B[x_1] \vee x0 \neq x_1 \vee \neg a > x_0 \vee \neg x_0 \geq 0$

`Find`
$a = 0; spot = m$
while $(a < m)$ **do**
if $(spot = m \&\& A[a] \neq 0)$
then $spot = a$
end if;
$B[a] = (A[a] \neq 0);$
$a = a + 1$
end do
	151	13	$spot = m \vee A[spot] \neq 0$	`inv3`:
				$a \geq spot$
				`inv39`:
				$spot = sk_1 \vee a = spot$
				`inv25`:
				$0 \geq sk_1 \vee a = spot$
				`inv5`:
				$\forall x_1: \quad \neg a > x_1 \vee \neg x_1 \geq 0 \vee a = spot \vee A(spot) \neq 0$

`Partition`
$a = 0; b = 0; c = 0;$
while $(a < m)$ **do**
if $(A[a] >= 0)$
then $B[b] = A[a]; b = b + 1$
else $C[c] = A[a]; c = c + 1$
end if;
$a = a + 1$
end do
	166	38	$\forall x: \quad 0 \leq x < b \rightarrow B[x] \geq 0 \wedge$
			$\exists y: \quad B[x] = A[y]$
				`inv1`:
				$\forall x_0: \quad A(sk_2(x_0)) \geq 0 \vee \neg b > x_0 \vee \neg x_0 \geq 0$
				`inv81`:
				$\forall x_0: \quad \neg b > x_0 \vee \neg x_0 \geq 0 \vee A(sk_2(x_0)) = B(x_0)$

`Partition_Init`
$a = 0; c = 0;$
while $(a < m)$ **do**
if $(A[a] == B[a])$
then $C[c] = a; c = c + 1$
end if;
$a = a + 1$
end do
	168	24	$\forall x: \quad 0 \leq x < c \rightarrow A[C[x]] = B[C[x]]$	`inv0`:
				$\forall x_0: \quad A(sk_1(x_0)) = B(sk_1(x_0)) \vee \neg c > x_0 \vee \neg x_0 \geq 0$
				`inv30`:
				$\forall x_0, x_1, x_2: \quad sk_1(x_0) \neq x_1 \vee x_0 \neq x_2 \vee \neg c > x_0 \vee \neg x_0 \geq 0 \vee C(x_2) = x_1$

Table: Vampire with 1 second time limit on challenging benchmarks.

Loop	Symbol Elimination within 1s					Symbol Elimination within 10s				
	♯ SEI	♯ Min SEI	% Redundancy	∀-Inv	∀∃-Inv	♯ SEI	♯ Min SEI	% Redund	∀-Inv	∀∃-Inv
`Initialisation`	15	5	67%	yes	no	40	5	88%	yes	no
`Copy`	24	5	79%	yes	no	25	5	80%	yes	no
`Find`	151	13	91%	yes	no	474	21	96%	yes	no
`Vararg`	1	1	0%	yes	no	1	1	0%	yes	no
`Partition`	166	38	77%	yes	yes	849	59	93%	yes	yes
`Partition_Init`	168	24	86%	yes	yes	692	127	82%	yes	yes

Table: Vampire with 1 and 10 seconds of time limit on challenging benchmarks.

Loop Shape	♯ of Loops	Average ♯ of SEI	Average ♯ of Non-Redundant SEI	% of SEI Redundancy	∀-Inv	∀∃-Inv
Simple	33	168	18	89.3%	yes	no
Multi-path	5	340	46	86.4%	yes	yes

Table: Vampire on industrial benchmarks of Dassault Aviation.

Summary: Invariant Generation and Symbol Elimination

Given a loop:

1. Express loop properties in a language containing extra symbols (loop counter, predicates expressing array updates, etc.);
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover 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 (SEI)**.

SEI: premise contains extra symbols, conclusion is in the loop language.

Conclusions: Experiments with Invariant Generation using Vampire

Vampire for Invariant Generation

Conclusions: Experiments with Invariant Generation using Vampire

Vampire for Invariant Generation

- Complex invariants were generated, already within 1s;
- Invariants implied intended loop properties;
- Consequence elimination significantly reduces the number of interesting invariants (by an 80% average);
- Theory reasoning and program analysis needs further improvements.

Conclusions: Experiments with Invariant Generation using Vampire

Vampire for Invariant Generation

- Complex invariants were generated, already within 1s;
- Invariants implied intended loop properties;
- Consequence elimination significantly reduces the number of interesting invariants (by an 80% average);
- Theory reasoning and program analysis needs further improvements.