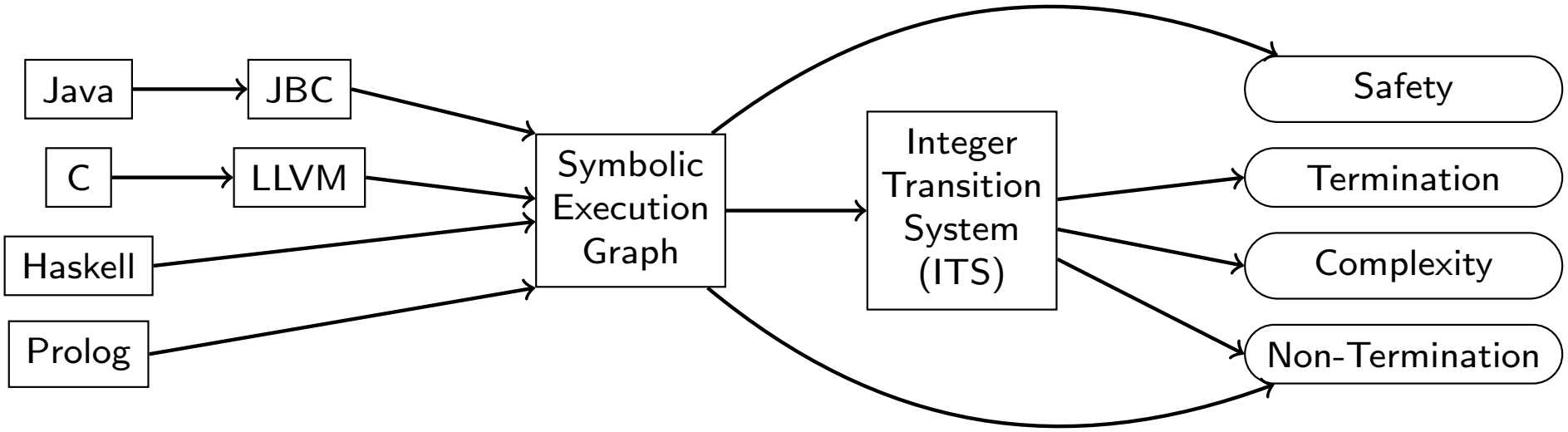
Termination Analysis of Programs with Bitvector Arithmetic by Symbolic Execution

Jürgen Giesl

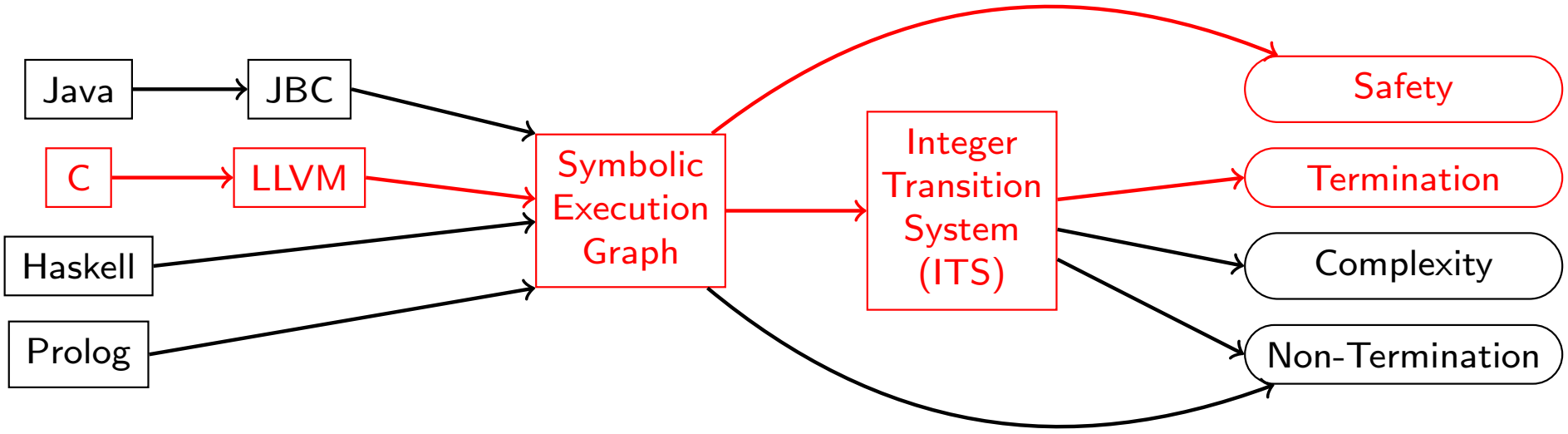
LuFG Informatik 2, RWTH Aachen University, Germany

joint work with Jera Hensel, Florian Frohn, and Thomas Ströder

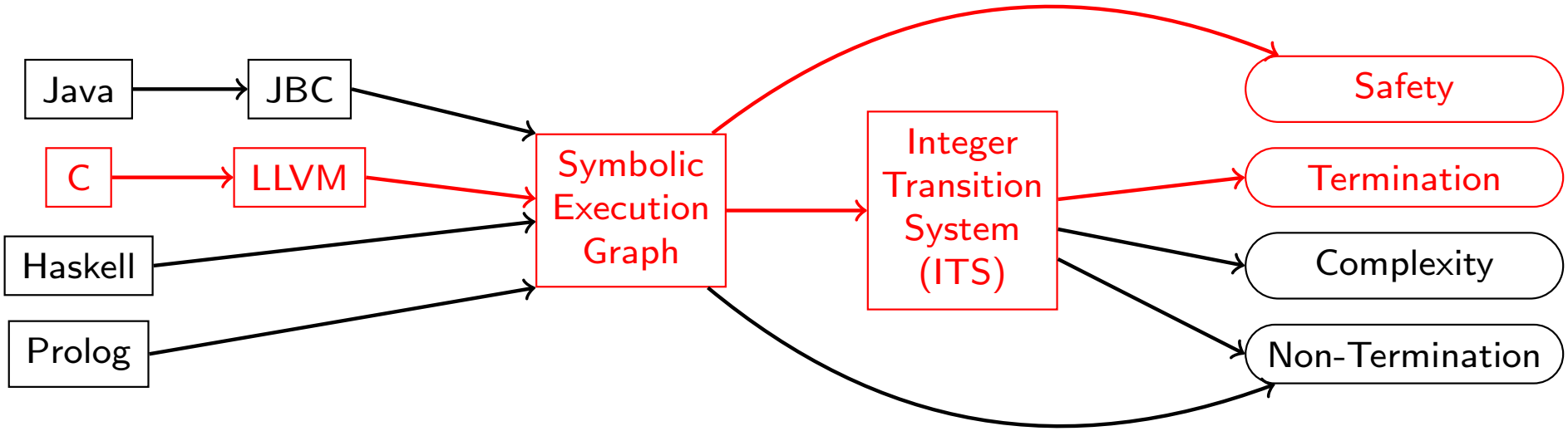
Termination Analysis in AProVE



Termination Analysis in AProVE

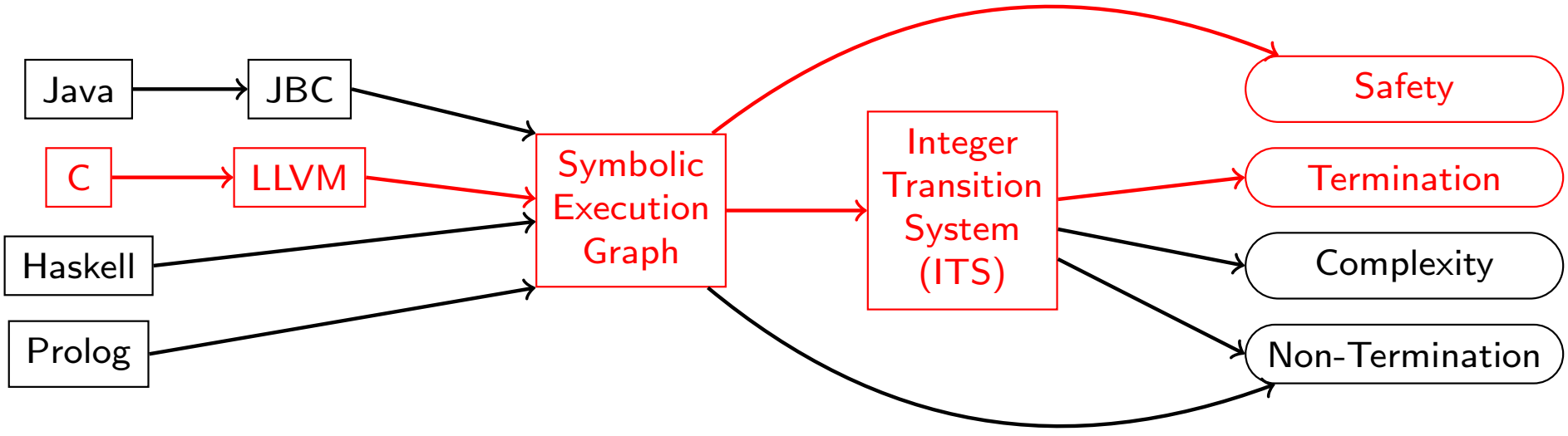


Termination Analysis in AProVE



- Termination of C programs with explicit pointer arithmetic

Termination Analysis in AProVE

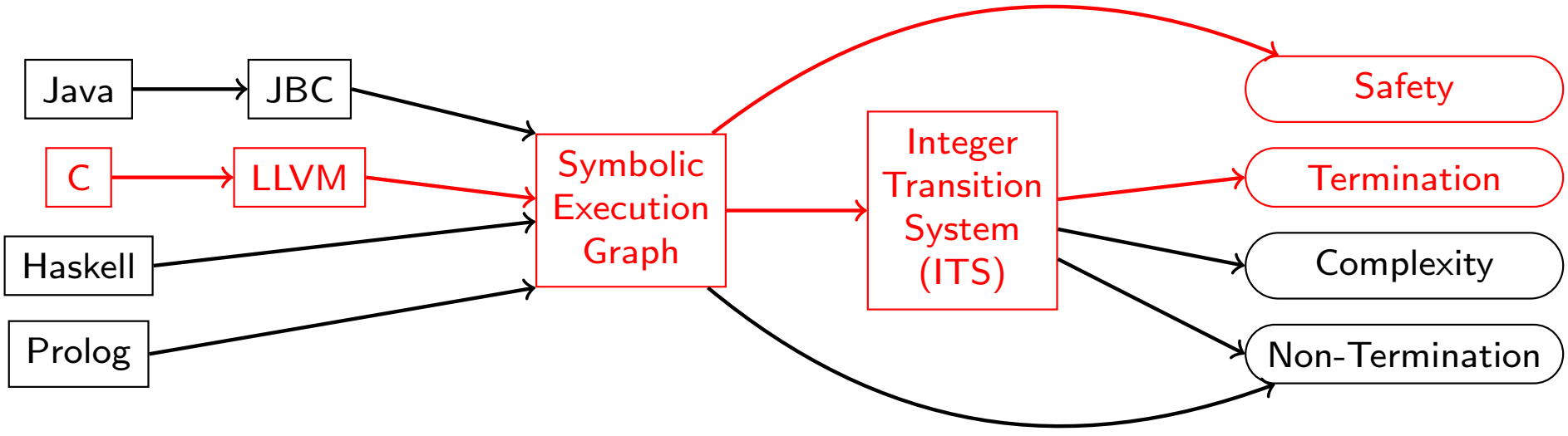


- Termination of C programs with explicit pointer arithmetic
- Winner of *SV-COMP 2015 & 2016* termination competition

+Term Comp'16

C Category

Termination Analysis in AProVE



- Termination of C programs with explicit pointer arithmetic
- Winner of *SV-COMP 2015 & 2016* termination competition
- **Drawback:** assumes mathematical integers $\mathbb{Z}$ instead of bitvectors

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

```
void g(unsigned int j)  {
    while (j > 0) j++;   }
```

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

```
void g(unsigned int j)  {
    while (j > 0) j++;   }
```

for $\mathbb{Z}$: termination

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: termination

for bitvectors: non-termination

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination

for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

- **Goal:** adapt termination analysis of C to bitvector arithmetic

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

- **Goal:** adapt termination analysis of C to bitvector arithmetic
- **Solution:** express bitvector relations by relations on $\mathbb{Z}$

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

- **Goal:** adapt termination analysis of C to bitvector arithmetic
- **Solution:** express bitvector relations by relations on $\mathbb{Z}$
  - standard SMT solving over $\mathbb{Z}$ for symbolic execution

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

- **Goal:** adapt termination analysis of C to bitvector arithmetic
- **Solution:** express bitvector relations by relations on $\mathbb{Z}$
  - standard SMT solving over $\mathbb{Z}$ for symbolic execution
  - standard ITSs over $\mathbb{Z}$ for termination proving

Mathematical Integers $\mathbb{Z}$ vs. Bitvectors

```
void f(unsigned int x)  {
    unsigned int j = 0;
    while (j <= x) j++;  }
```

for $\mathbb{Z}$: termination
for bitvectors: non-termination

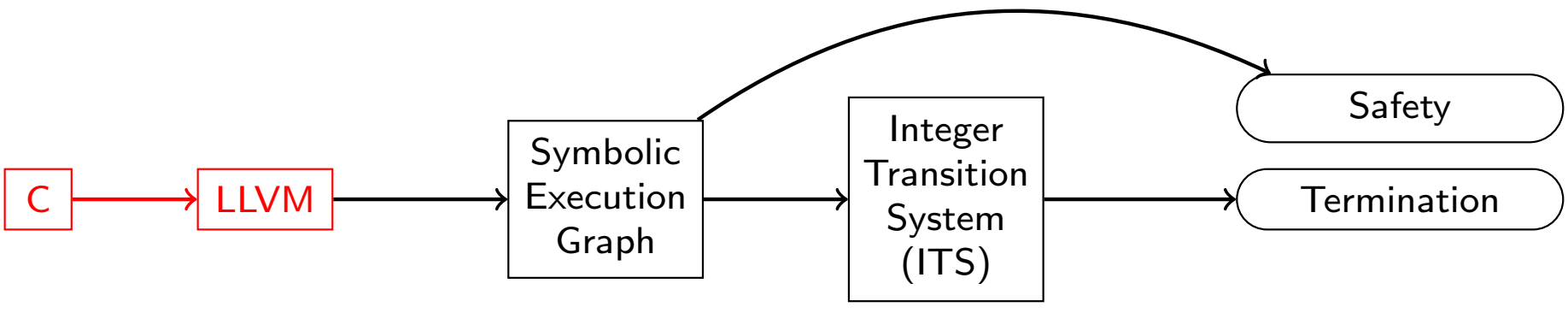
```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```

for $\mathbb{Z}$: non-termination
for bitvectors: termination

- **Goal:** adapt byte-accurate symbolic execution to bitvector arithmetic
- **Solution:** express bitvector relations by relations on $\mathbb{Z}$
  - standard SMT solving over $\mathbb{Z}$ for symbolic execution
  - standard ITSs over $\mathbb{Z}$ for termination proving

From C to LLVM

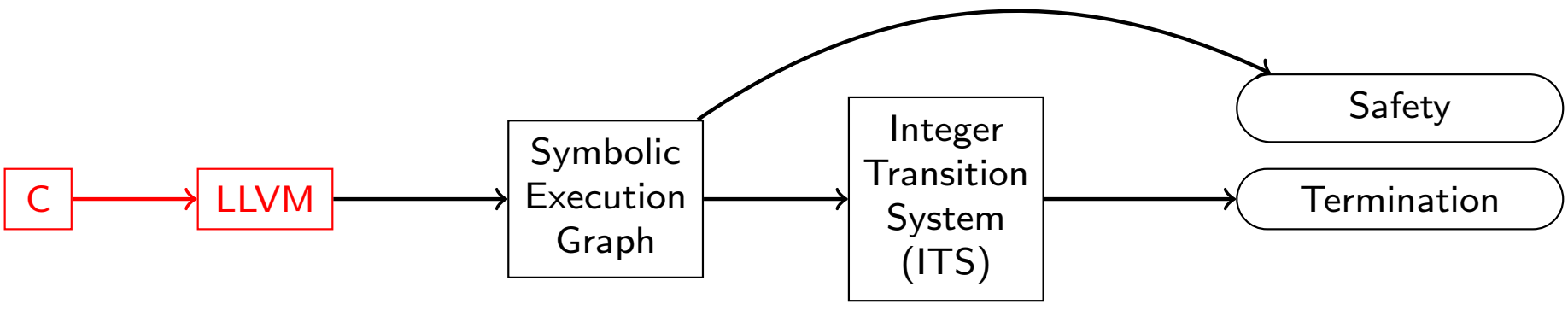
```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```



From C to LLVM

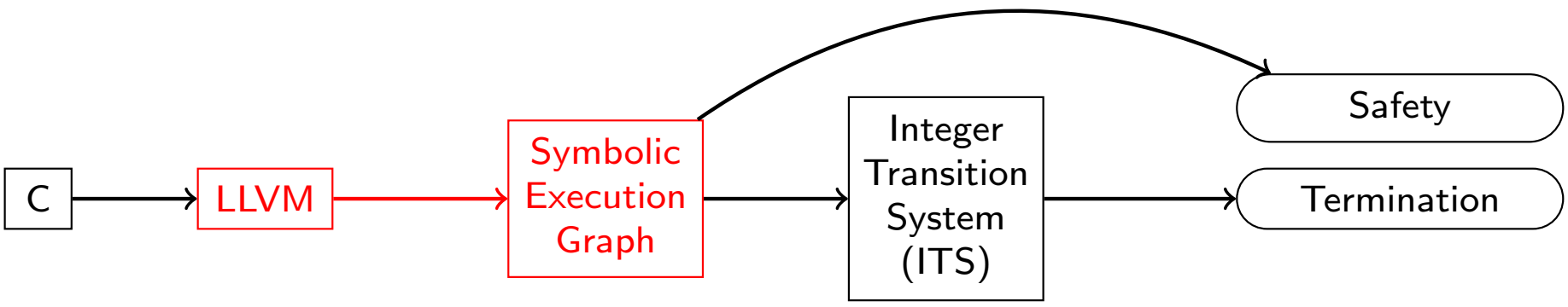
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

```
void g(unsigned int j)  {
    while (j > 0) j++;  }
```



Abstract States

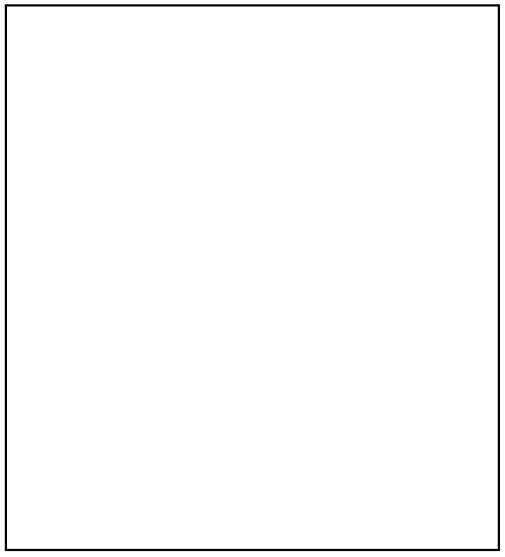
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

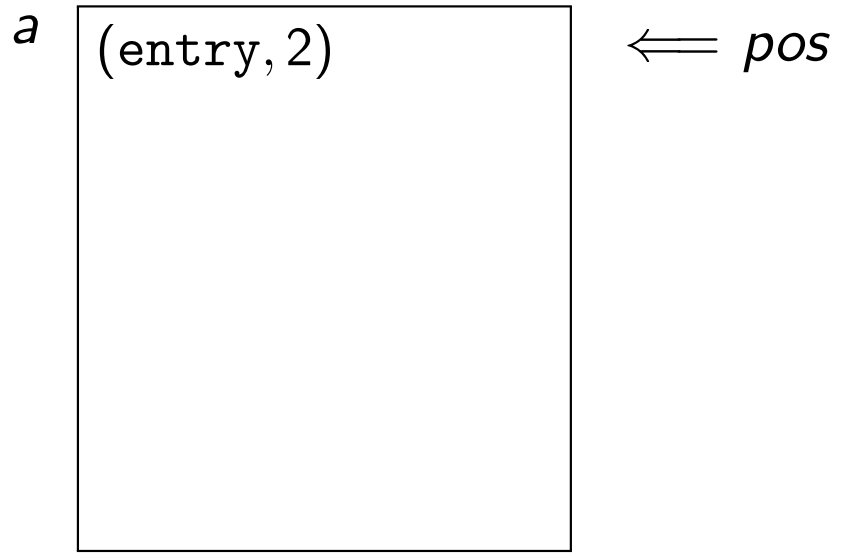
a



Abstract state *a*:

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Abstract state *a*:

pos: program position (block, next instruction)

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$ $\Longleftarrow$ *pos*

$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$ $\Longleftarrow$ *PV*

Abstract state *a*:

pos: program position (block, next instruction)

PV: program variables $\rightarrow$ symbolic variables

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$ $\Longleftarrow$ *pos*

$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$ $\Longleftarrow$ *PV*

$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$ $\Longleftarrow$ *AL*

Abstract state *a*:

- *pos*: program position (block, next instruction)
- *PV*: program variables $\rightarrow$ symbolic variables
- *AL*: allocation list $[\![v_1, v_2]\!]$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$	$\Longleftarrow$ *pos*
$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$	$\Longleftarrow$ *PV*
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$	$\Longleftarrow$ *AL*
$\{v_{end} = v_{\texttt{ad}} + 3\}$	$\Longleftarrow$ *KB*

Abstract state *a*:

- *pos*: program position (block, next instruction)
- *PV*: program variables $\rightarrow$ symbolic variables
- *AL*: allocation list $[\![v_1, v_2]\!]$
- *KB*: knowledge base (FO-(in)equalities over symbolic variables)

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$(\texttt{entry}, 2) \Longleftarrow pos$$

$$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\} \Longleftarrow PV$$

$$\{[\![v_{\texttt{ad}}, v_{end}]\!]\} \Longleftarrow AL$$

$$\{v_{end} = v_{\texttt{ad}} + 3\} \Longleftarrow KB$$

Heuristic: partition program variables into $\mathcal{U} \uplus \mathcal{S}$

$\texttt{x} \in \mathcal{U}$: $PV(\texttt{x})$ represents value of $\texttt{x}$ as unsigned integer

Abstract state *a*:

- *pos*: program position (block, next instruction)
- *PV*: program variables $\rightarrow$ symbolic variables
- *AL*: allocation list $[\![v_1, v_2]\!]$
- *KB*: knowledge base (FO-(in)equalities over symbolic variables)

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$	$\Longleftarrow pos$
$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$	$\Longleftarrow PV$
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$	$\Longleftarrow AL$
$\{v_{end} = v_{\texttt{ad}} + 3\}$	$\Longleftarrow KB$
$\{v_{\texttt{ad}} \hookrightarrow_{\texttt{i32},u} v_{\texttt{j}}\}$	$\Longleftarrow PT$

Abstract state a:

- pos: program position (block, next instruction)
- PV: program variables $\rightarrow$ symbolic variables
- AL: allocation list $[\![v_1, v_2]\!]$
- KB: knowledge base (FO-(in)equalities over symbolic variables)
- PT: points-to atoms $v_1 \hookrightarrow_{\texttt{type},u} v_2$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$	$\Longleftarrow$ *pos*
$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$	$\Longleftarrow$ *PV*
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$	$\Longleftarrow$ *AL*
$\{v_{end} = v_{\texttt{ad}} + 3\}$	$\Longleftarrow$ *KB*
$\{v_{\texttt{ad}} \hookrightarrow_{\texttt{i32},u} v_{\texttt{j}}\}$	$\Longleftarrow$ *PT*

Abstract state *a*: *ERR* or

- *pos*: program position (block, next instruction)
- *PV*: program variables $\rightarrow$ symbolic variables
- *AL*: allocation list $[\![v_1, v_2]\!]$
- *KB*: knowledge base (FO-(in)equalities over symbolic variables)
- *PT*: points-to atoms $v_1 \hookrightarrow_{\texttt{type},u} v_2$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

- $\langle a \rangle$: FO formula

a

$$
\begin{array}{|l|l}
\hline
(\mathtt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{\mathit{end}}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{\mathit{end}} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l|}
\hline
(\mathtt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{end}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{end} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula containing
  - KB and consequences of AL and PT

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$	$\Longleftarrow pos$
$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$	$\Longleftarrow PV$
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$	$\Longleftarrow AL$
$\{v_{end} = v_{\texttt{ad}} + 3\}$	$\Longleftarrow KB$
$\{v_{\texttt{ad}} \hookrightarrow_{\texttt{i32},u} v_{\texttt{j}}\}$	$\Longleftarrow PT$

- $\langle a \rangle$: FO formula containing
  - KB and consequences of AL and PT
  - information on ranges of integers:

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l}
\hline
(\texttt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{end}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{end} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula containing
  - KB and consequences of AL and PT
  - information on ranges of integers:

    $\mathtt{j} \in \mathcal{U}$ has type $\mathtt{i32} \quad \Rightarrow \quad 0 \leq \underbrace{PV(\mathtt{j})}_{v_{\mathtt{j}}} \leq \underbrace{\mathrm{umax}_{32}}_{2^{32}-1}$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l|}
\hline
(\mathtt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{end}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{end} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula
- *a concrete*:

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l}
\hline
(\texttt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{\mathit{end}}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{\mathit{end}} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula
- *a concrete*: $\forall$ symbolic variables v $\exists\, n \in \mathbb{Z}$ such that $\models \langle a \rangle \Rightarrow v = n$

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l|}
\hline
(\texttt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{\mathit{end}}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{\mathit{end}} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula
- *a concrete*: $\forall$ symbolic variables v $\exists\, n \in \mathbb{Z}$ such that $\models \langle a \rangle \Rightarrow v = n$
- $\langle a \rangle_{SL}$: separation logic formula, extends $\langle a \rangle$ by details on memory

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$(\texttt{entry}, 2)$	$\Longleftarrow pos$
$\{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\}$	$\Longleftarrow PV$
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$	$\Longleftarrow AL$
$\{v_{end} = v_{\texttt{ad}} + 3\}$	$\Longleftarrow KB$
$\{v_{\texttt{ad}} \hookrightarrow_{\texttt{i32},u} v_{\texttt{j}}\}$	$\Longleftarrow PT$

- $\langle a \rangle$: FO formula
- *a concrete*: $\forall$ symbolic variables v $\exists\, n \in \mathbb{Z}$ such that $\models \langle a \rangle \Rightarrow v = n$
- $\langle a \rangle_{SL}$: separation logic formula, extends $\langle a \rangle$ by details on memory
- abstract state a *represents* concrete state

Abstract States

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

a

$$
\begin{array}{|l|l}
\hline
(\texttt{entry}, 2) & \Longleftarrow \mathit{pos} \\
\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} & \Longleftarrow \mathit{PV} \\
\{[\![v_{\mathtt{ad}}, v_{\mathit{end}}]\!]\} & \Longleftarrow \mathit{AL} \\
\{v_{\mathit{end}} = v_{\mathtt{ad}} + 3\} & \Longleftarrow \mathit{KB} \\
\{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

- $\langle a \rangle$: FO formula
- *a concrete*: $\forall$ symbolic variables v $\exists\, n \in \mathbb{Z}$ such that $\models \langle a \rangle \Rightarrow v = n$
- $\langle a \rangle_{SL}$: separation logic formula, extends $\langle a \rangle$ by details on memory
- abstract state a *represents* concrete state iff
  $\langle a \rangle_{SL}$ is satisfied by instantiation corresponding to concrete state

Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

A

$(\texttt{entry}, 0)$
$\{\mathrm{j} = v_{\mathrm{j}}, \ldots\}$
$\varnothing$
$\{0 \leq v_{\mathrm{j}} \leq \mathrm{umax}, \ldots\}$
$\varnothing$

Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

A

$$
\begin{array}{|l|l}
\hline
(\texttt{entry}, 0) & \Longleftarrow \mathit{pos} \\
\{\mathrm{j} = v_{\mathrm{j}}, \ldots\} & \Longleftarrow \mathit{PV} \\
\varnothing & \Longleftarrow \mathit{AL} \\
\{0 \leq v_{\mathrm{j}} \leq \mathrm{umax}, \ldots\} & \Longleftarrow \mathit{KB} \\
\varnothing & \Longleftarrow \mathit{PT} \\
\hline
\end{array}
$$

Symbolic Execution

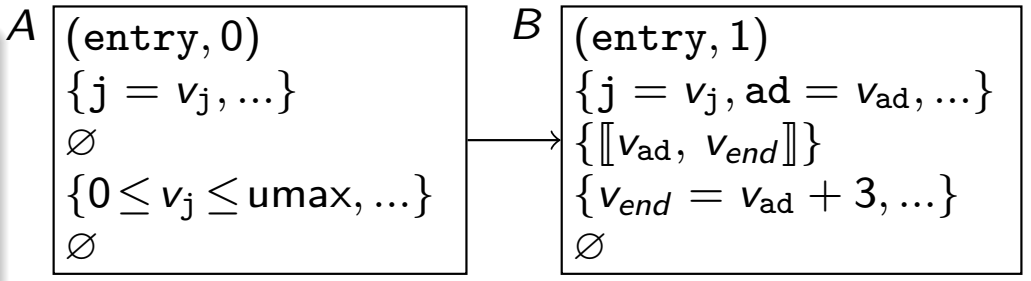
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

A

$(\texttt{entry}, 0)$	$\Longleftarrow pos$
$\{\texttt{j} = v_{\texttt{j}}, \ldots\}$	$\Longleftarrow PV$
$\varnothing$	$\Longleftarrow AL$
$\{0 \leq v_{\texttt{j}} \leq \texttt{umax}, \ldots\}$	$\Longleftarrow KB$
$\varnothing$	$\Longleftarrow PT$

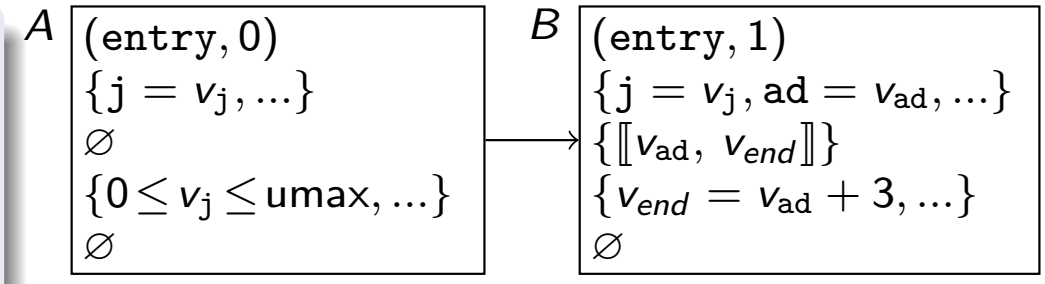
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



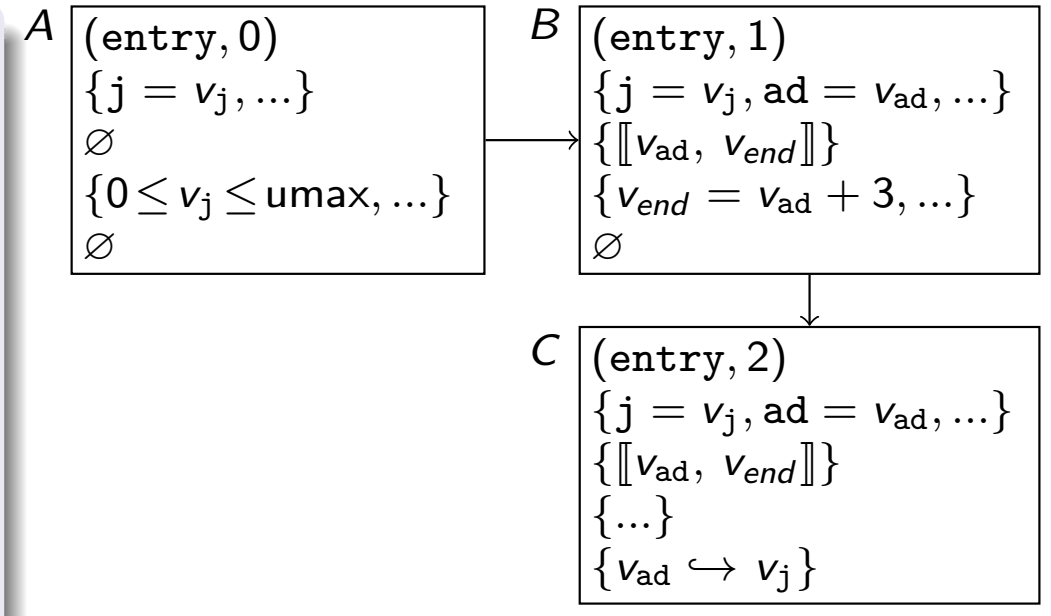
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: store i32 inc, i32* ad
       3: br label cmp
done:  0: ret void }
```



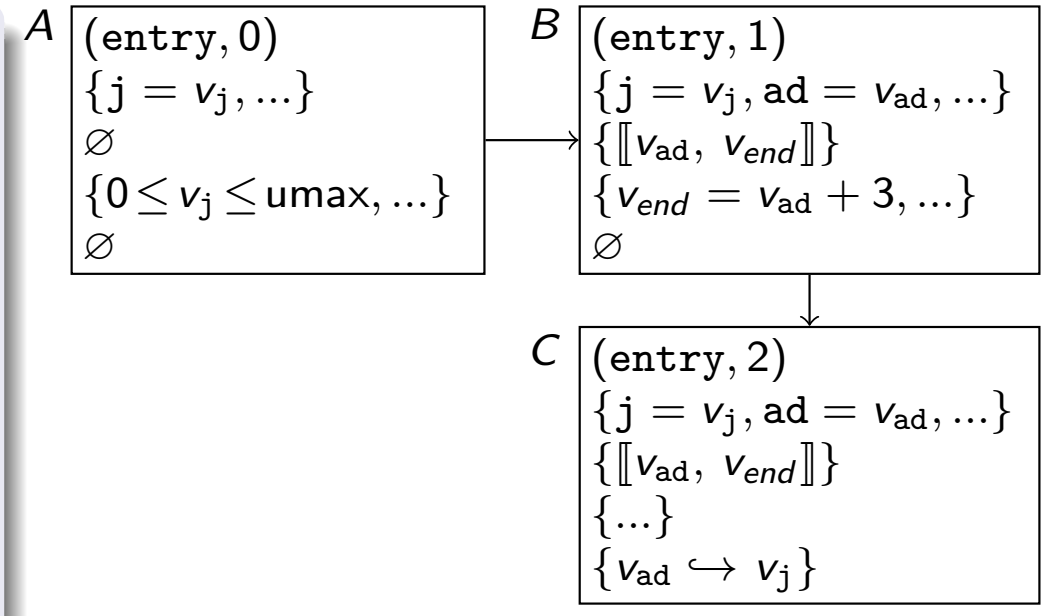
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



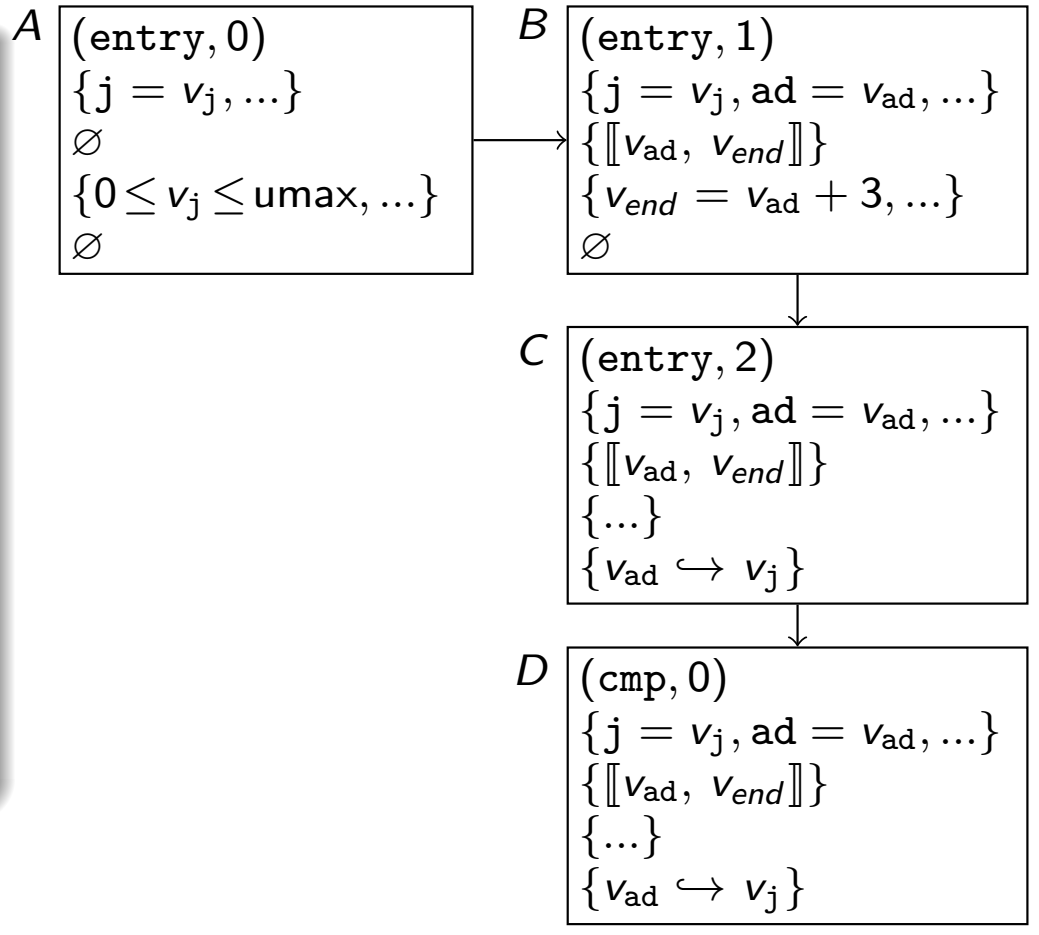
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



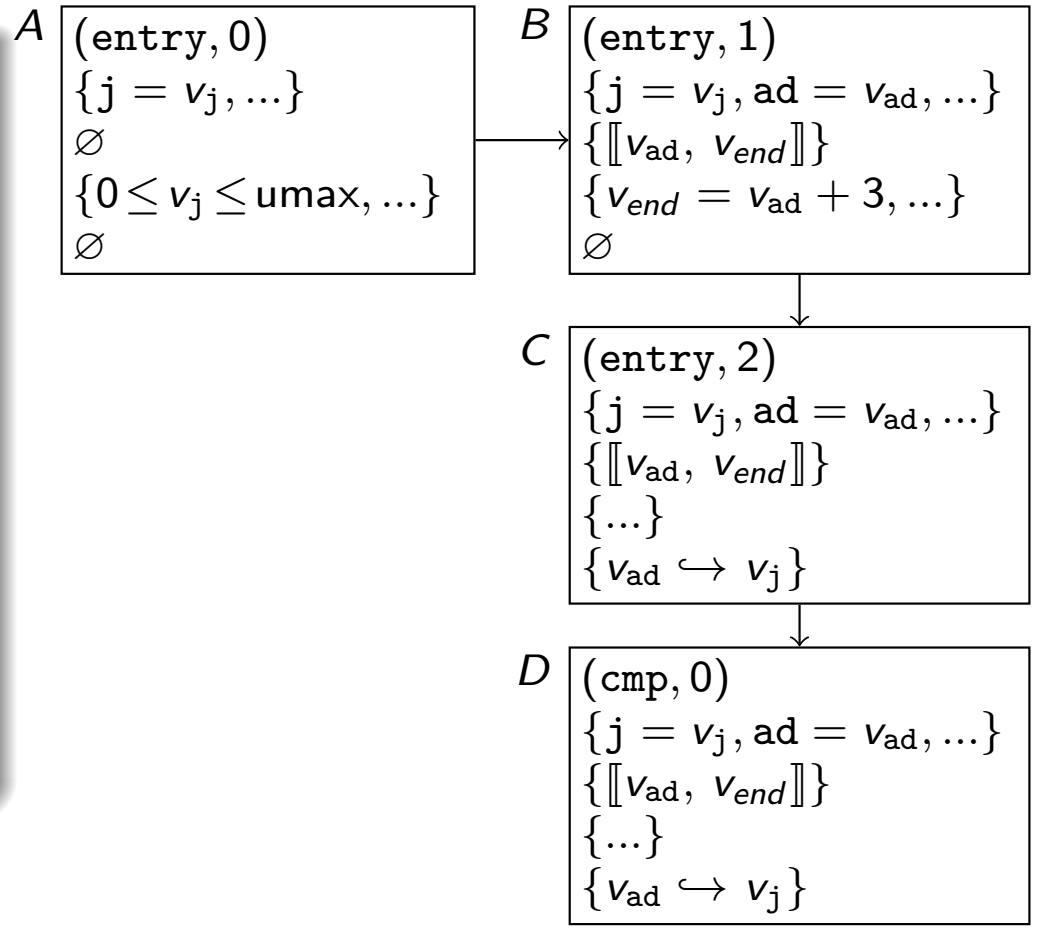
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



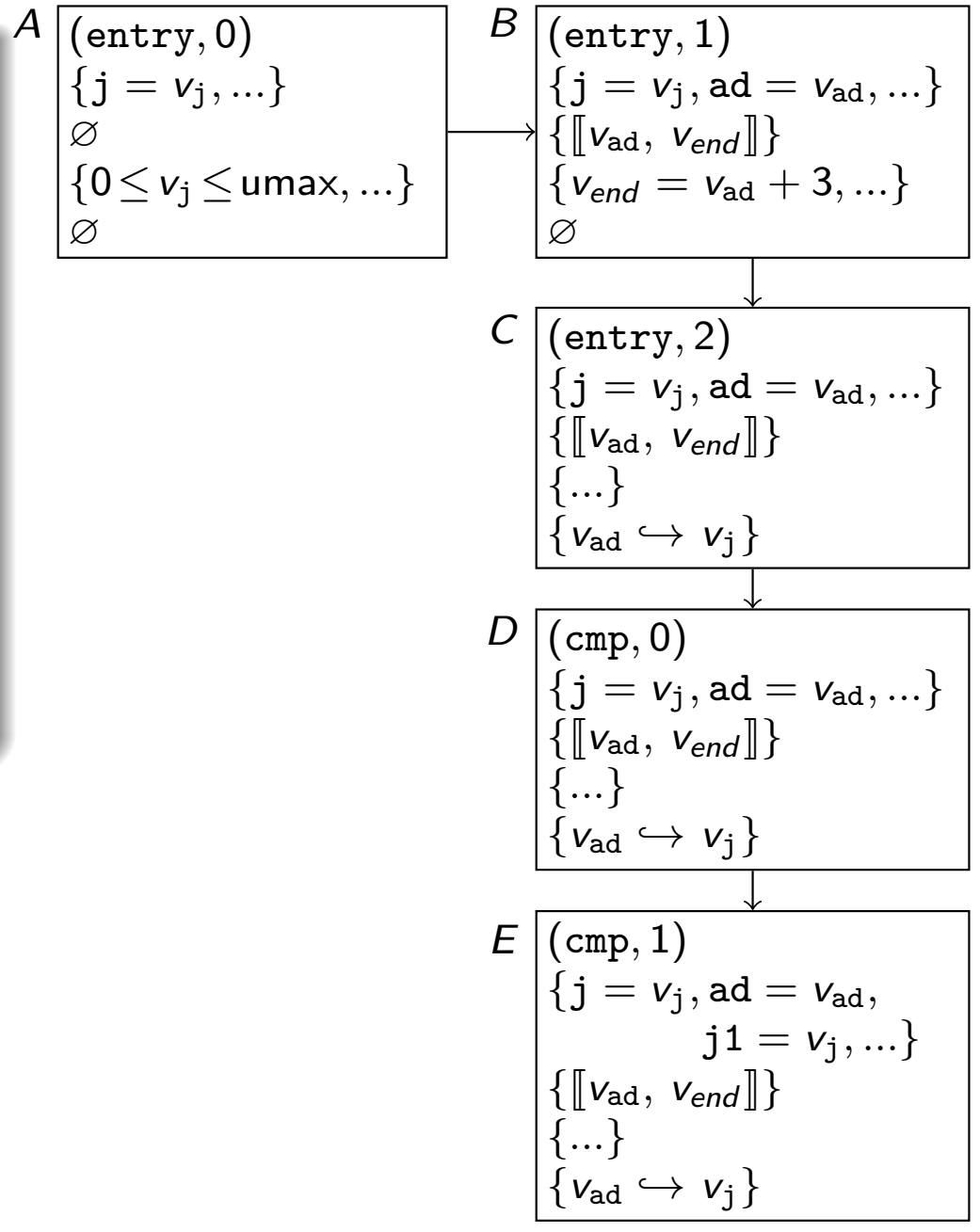
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: store i32 inc, i32* ad
       3: br label cmp
done:  0: ret void }
```



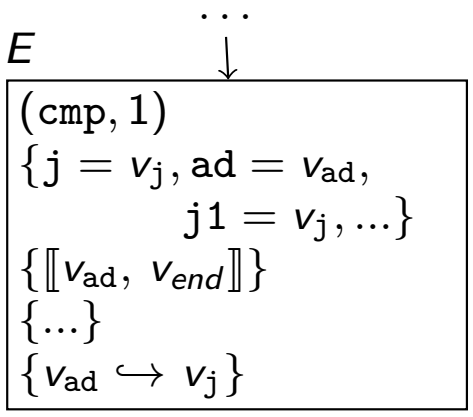
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



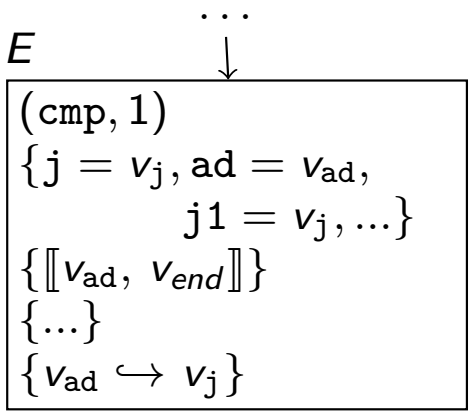
Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Integer Comparison

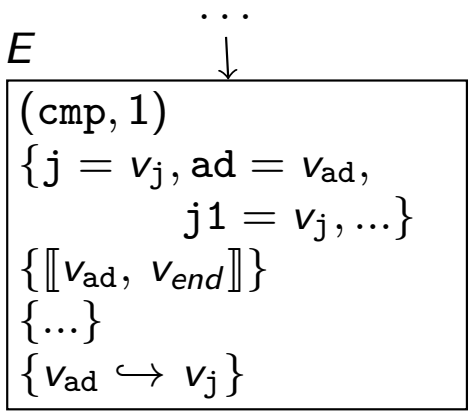
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

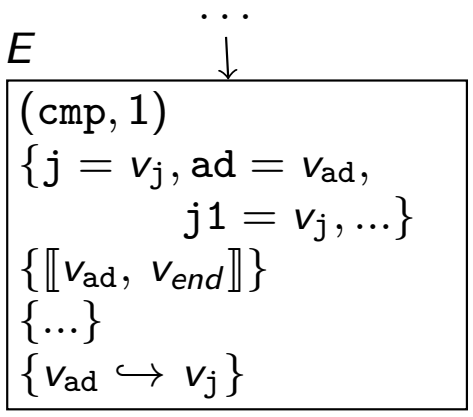


Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

- set `x` to `1` if $\models \langle a \rangle \implies (PV_u(t_1) > PV_u(t_2))$

Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

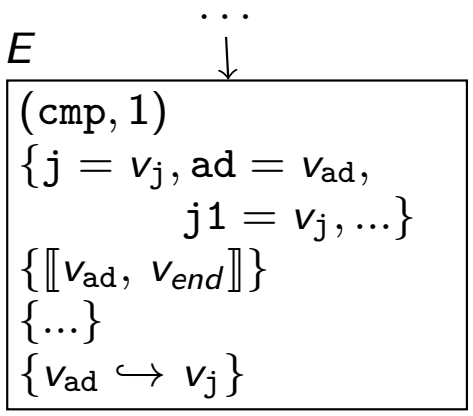


Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

- set `x` to 1 if $\models \langle a \rangle \implies (PV_u(t_1) > PV_u(t_2))$
- set `x` to 0 if $\models \langle a \rangle \implies (PV_u(t_1) \leq PV_u(t_2))$

Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

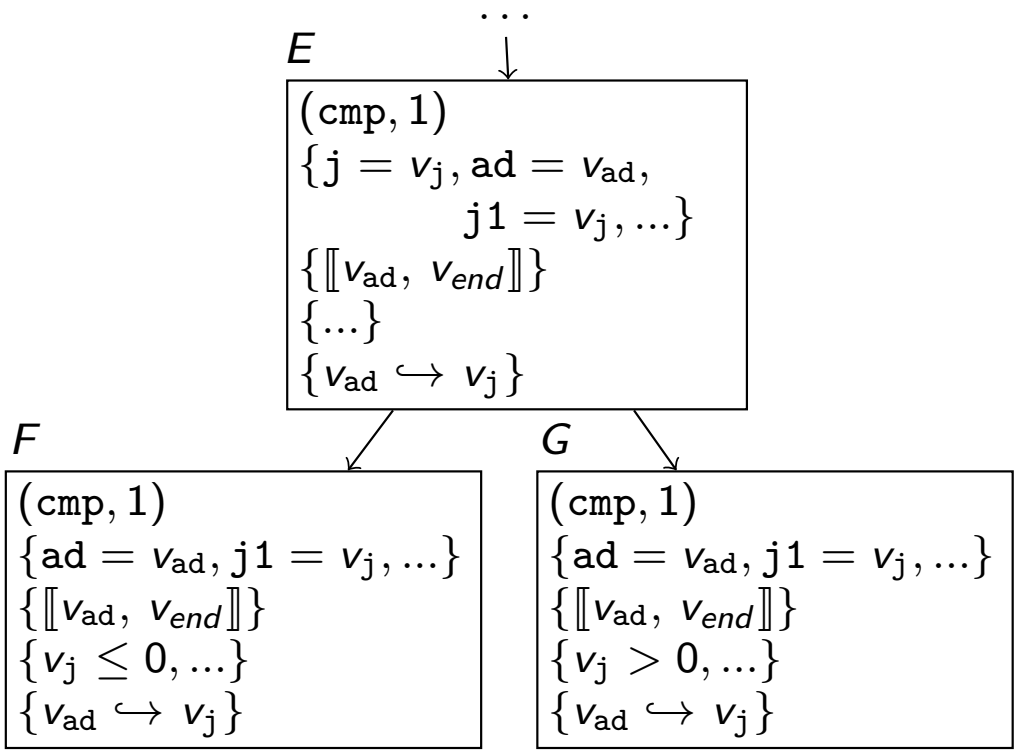


Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

- set `x` to 1 if $\models \langle a \rangle \implies (PV_u(t_1) > PV_u(t_2))$
- set `x` to 0 if $\models \langle a \rangle \implies (PV_u(t_1) \leq PV_u(t_2))$
- otherwise: case analysis

Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

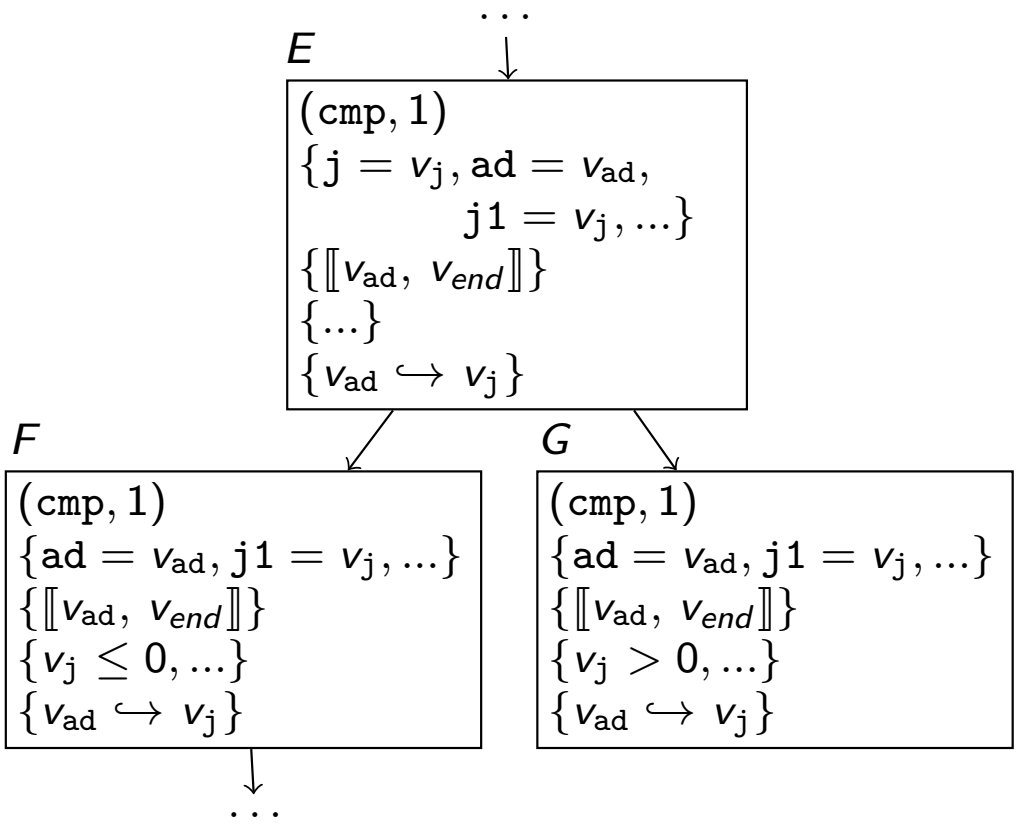


Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

- set `x` to 1 if $\models \langle a \rangle \implies (PV_u(t_1) > PV_u(t_2))$
- set `x` to 0 if $\models \langle a \rangle \implies (PV_u(t_1) \leq PV_u(t_2))$
- otherwise: case analysis

Integer Comparison

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

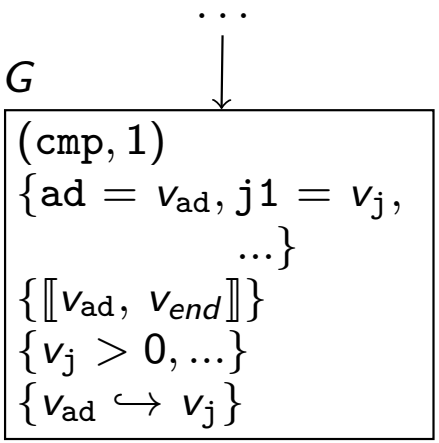


Symbolic execution rule for `x = icmp ugt i32` t_1, t_2

- set `x` to 1 if $\models \langle a \rangle \implies (PV_u(t_1) > PV_u(t_2))$
- set `x` to 0 if $\models \langle a \rangle \implies (PV_u(t_1) \leq PV_u(t_2))$
- otherwise: case analysis

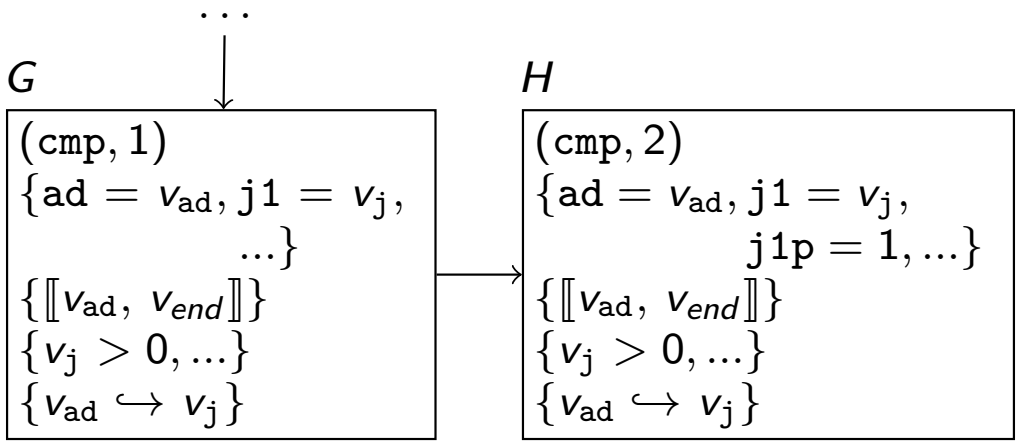
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



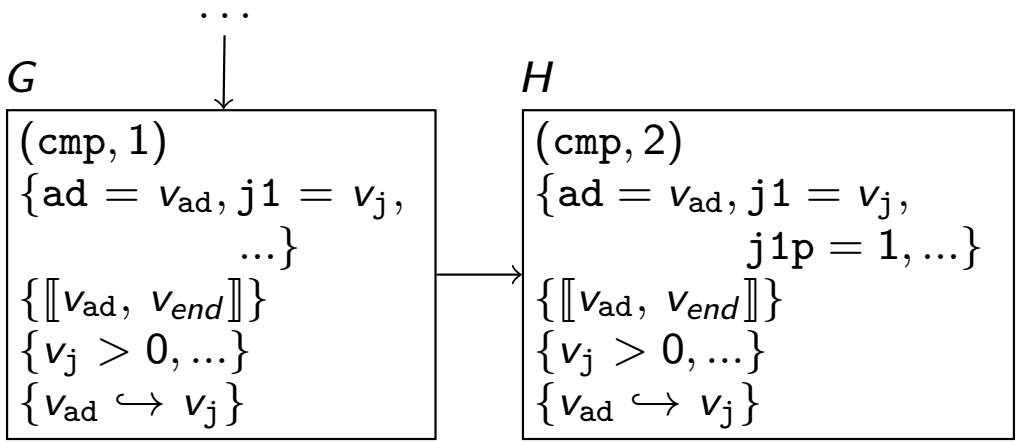
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



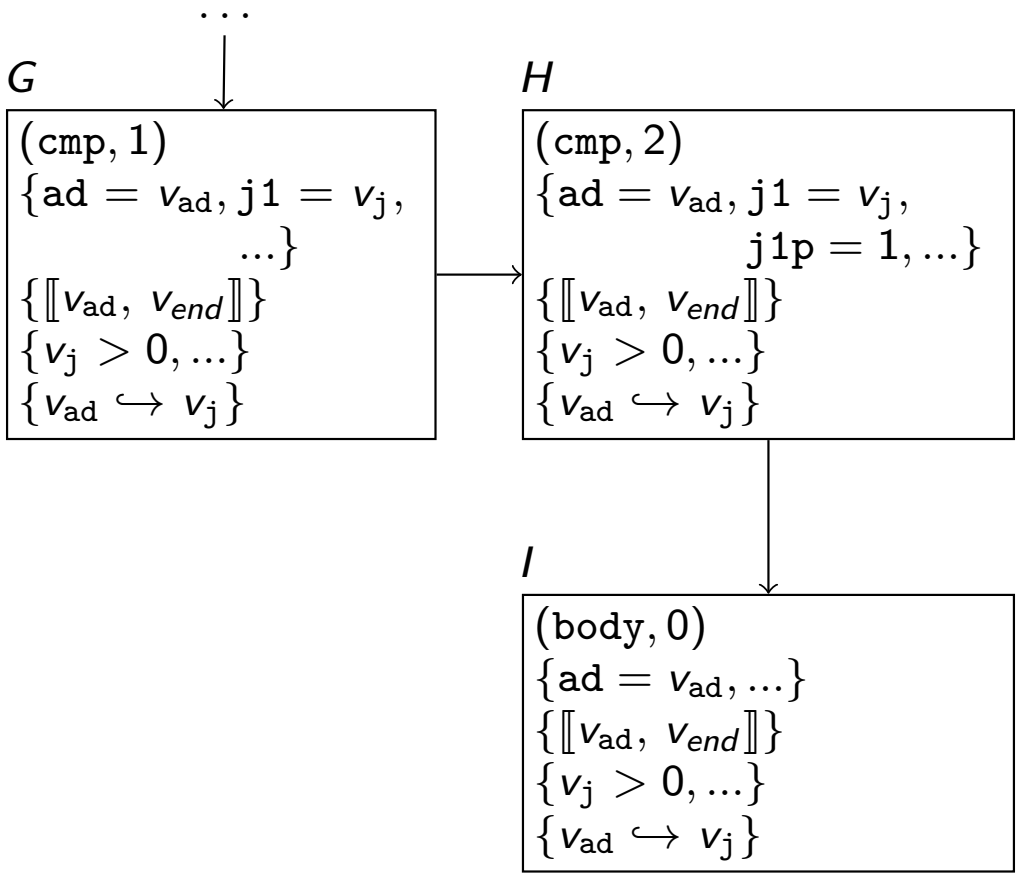
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



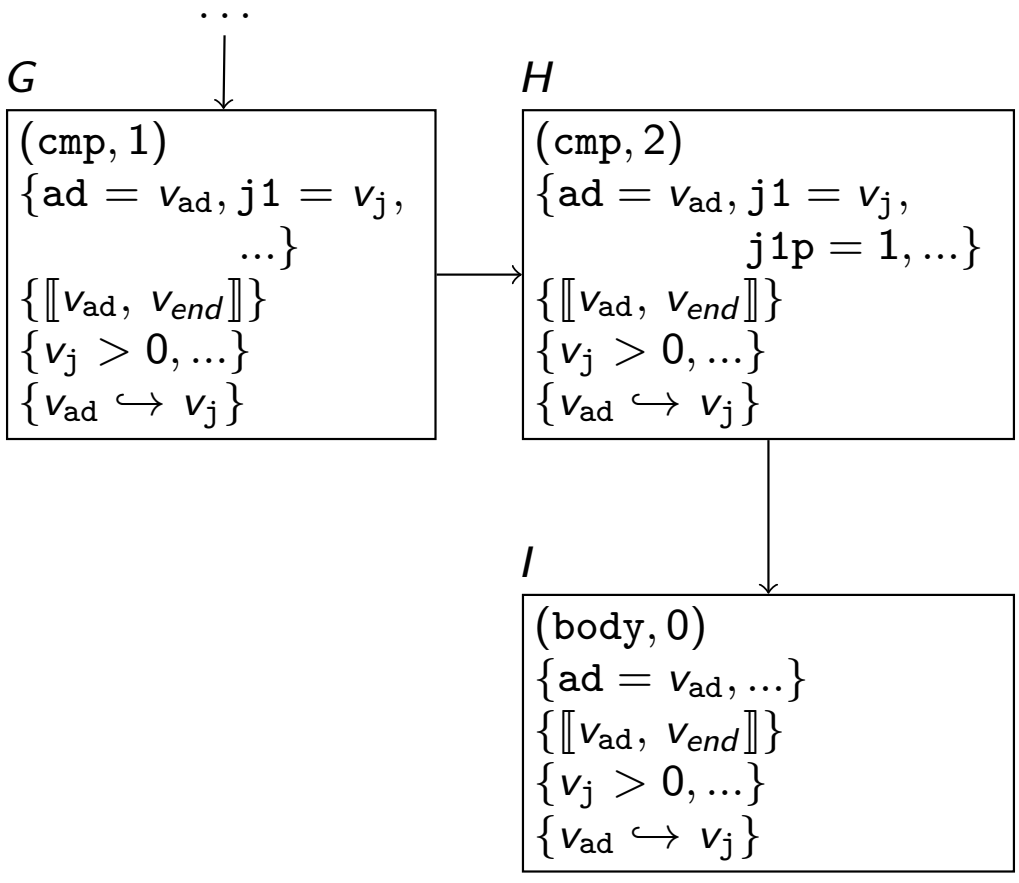
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



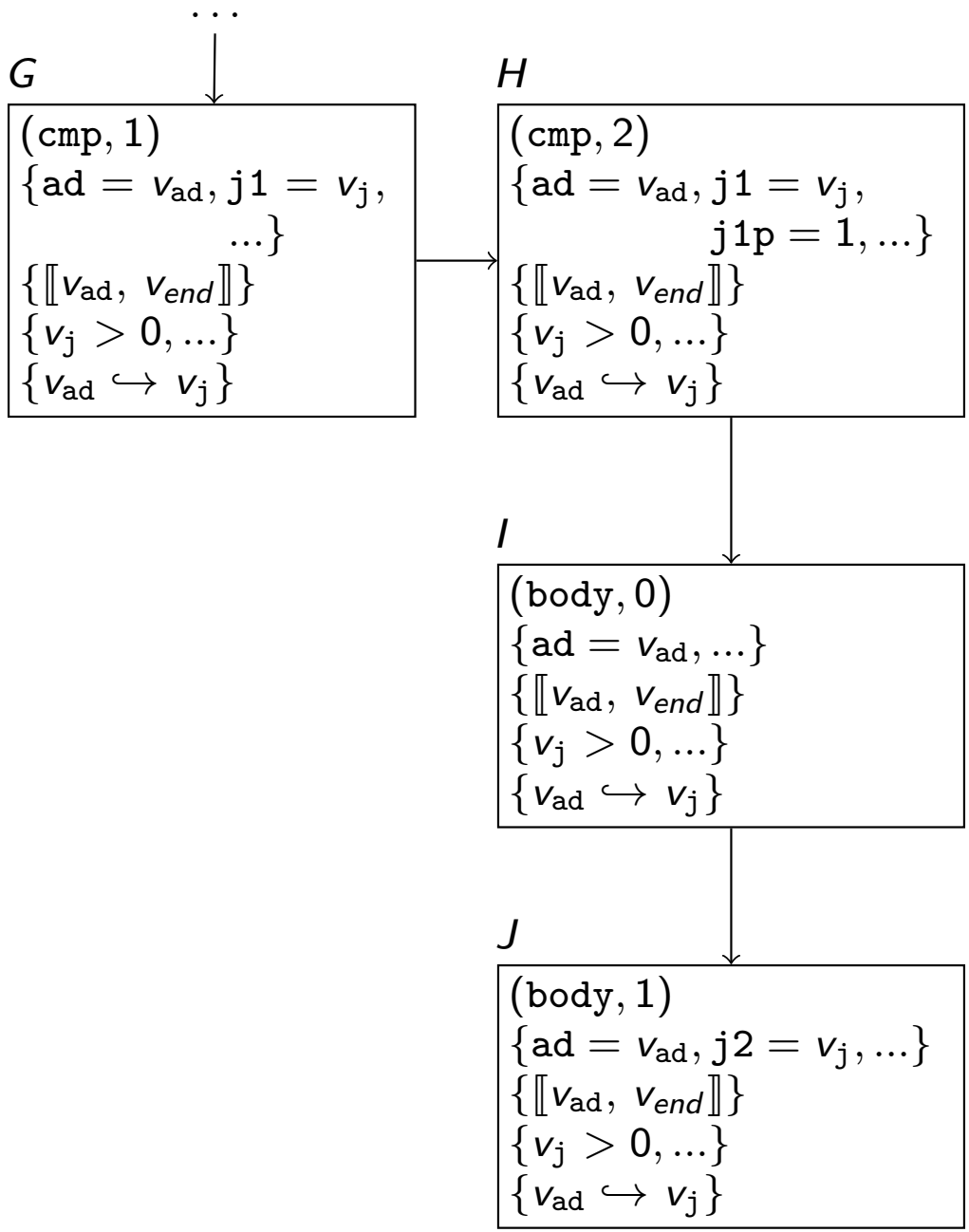
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



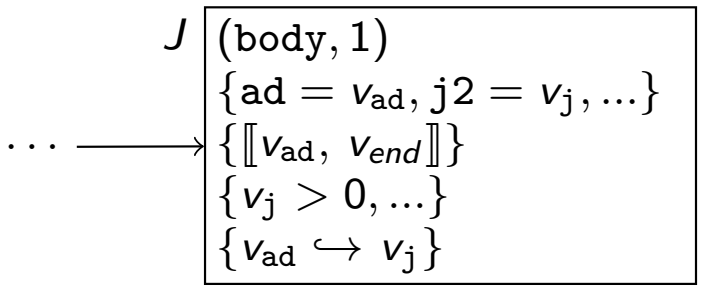
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



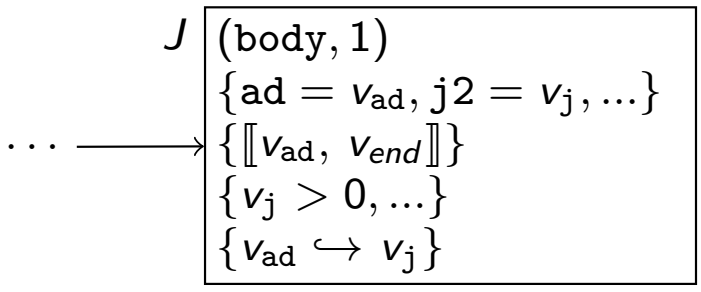
Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Addition

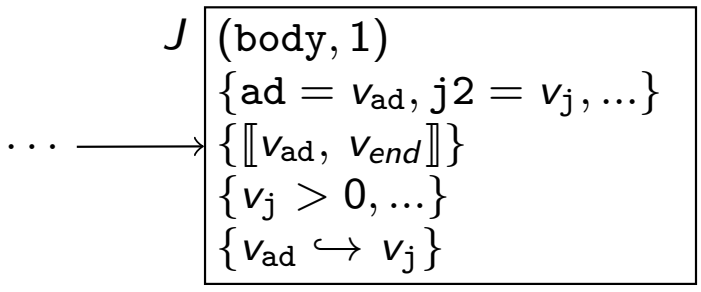
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Symbolic execution rule for `x = add i32` t_1, t_2

Addition

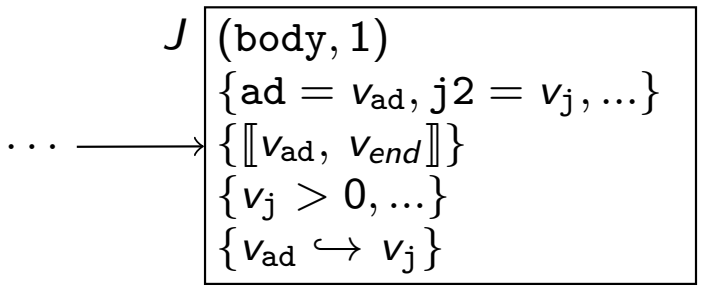
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Symbolic execution rule for `x = add i32` t_1, t_2 where $\mathtt{x} \in \mathcal{U}$

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

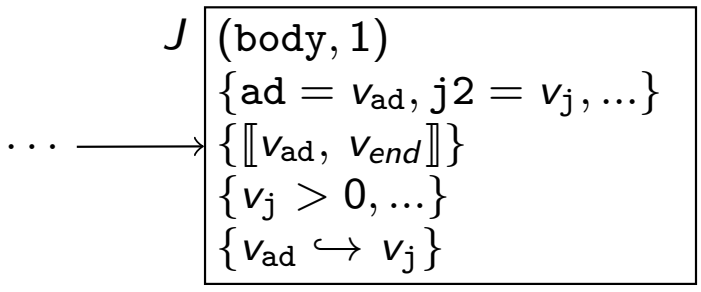


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \mathsf{umax}_{32}$

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

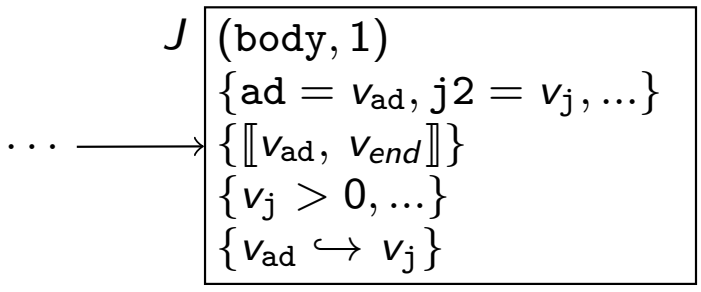


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

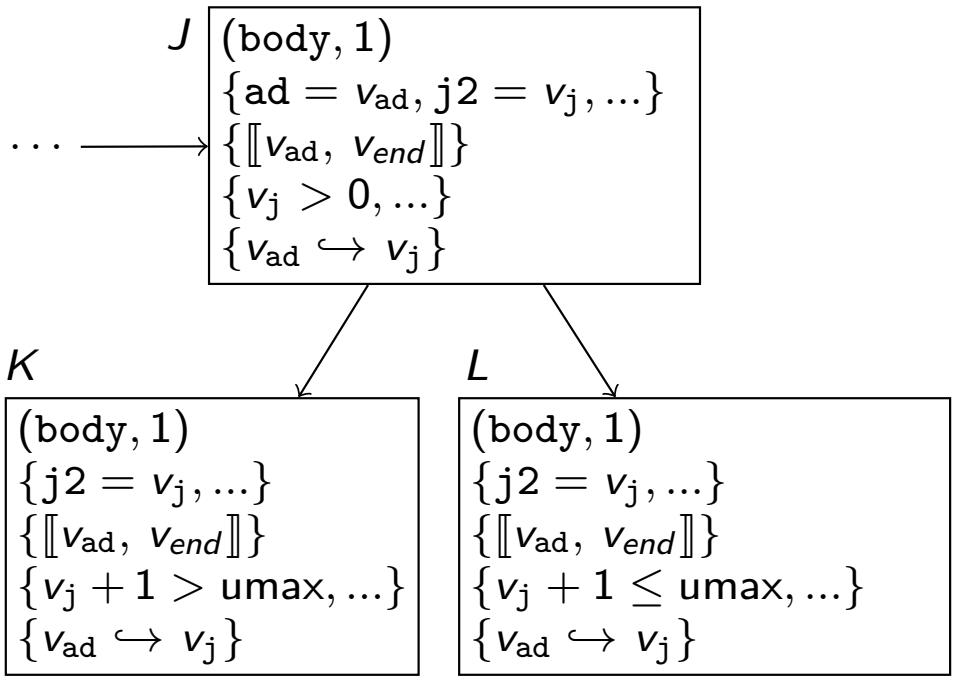


Symbolic execution rule for `x = add i32` t_1, t_2 **where** `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$
- otherwise: case analysis

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

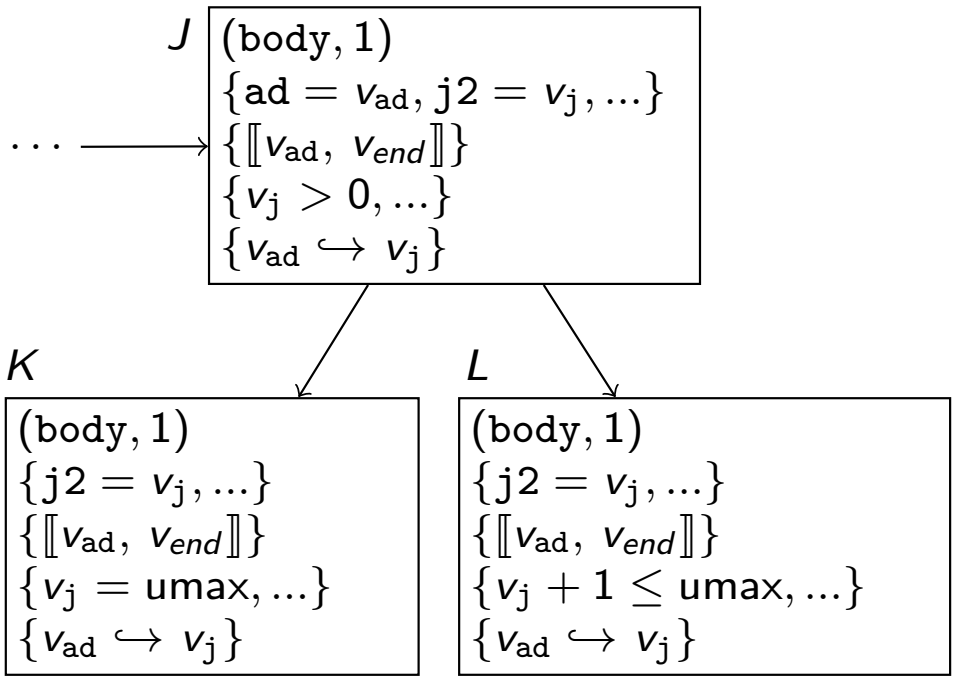


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$
- otherwise: case analysis

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

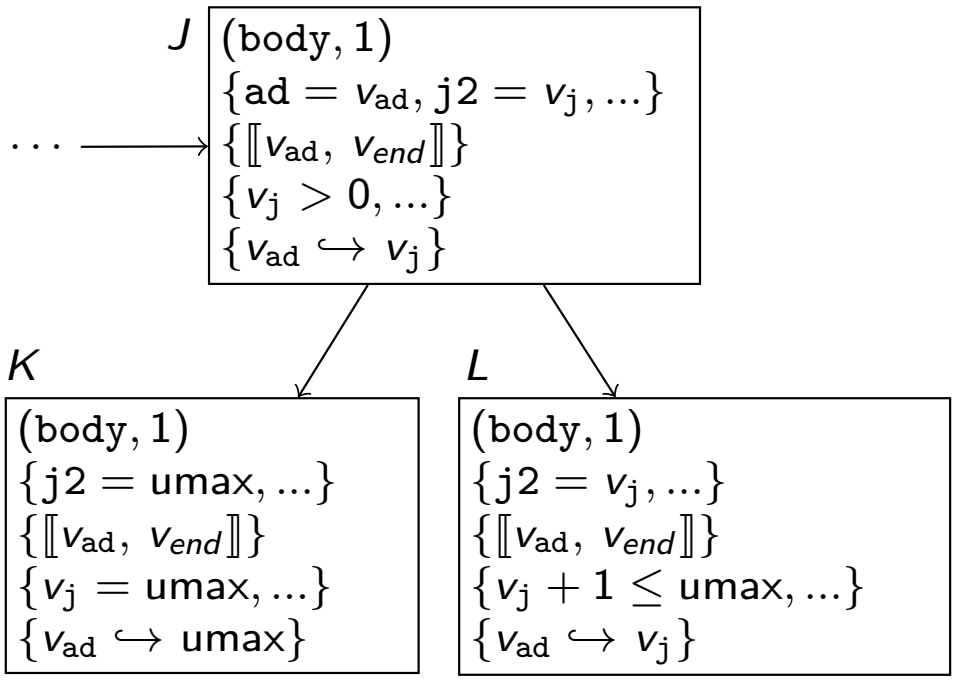


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$
- otherwise: case analysis

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

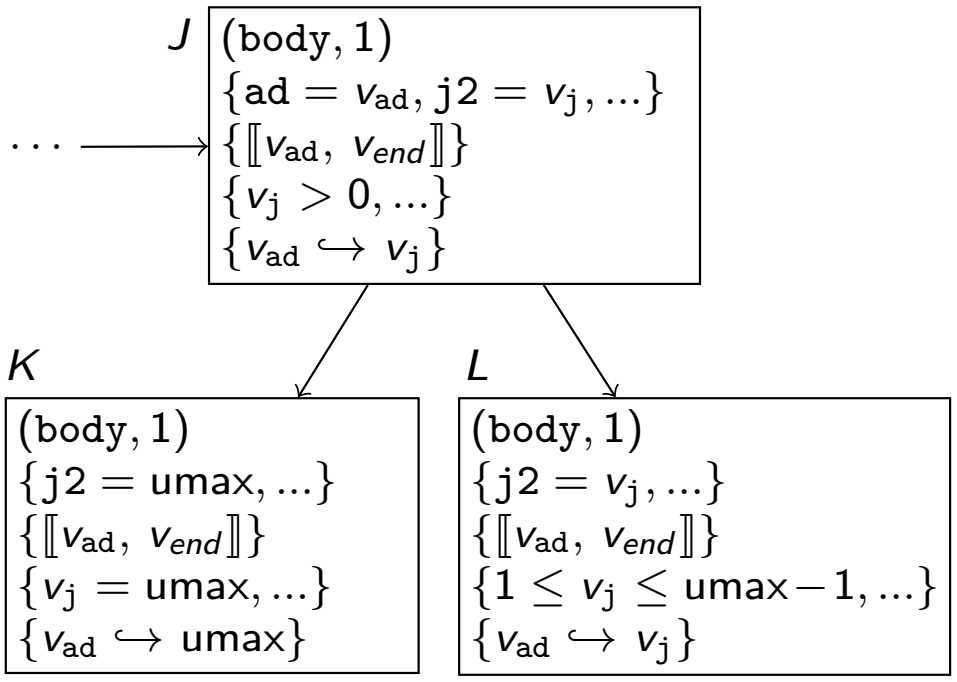


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$
- otherwise: case analysis

Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

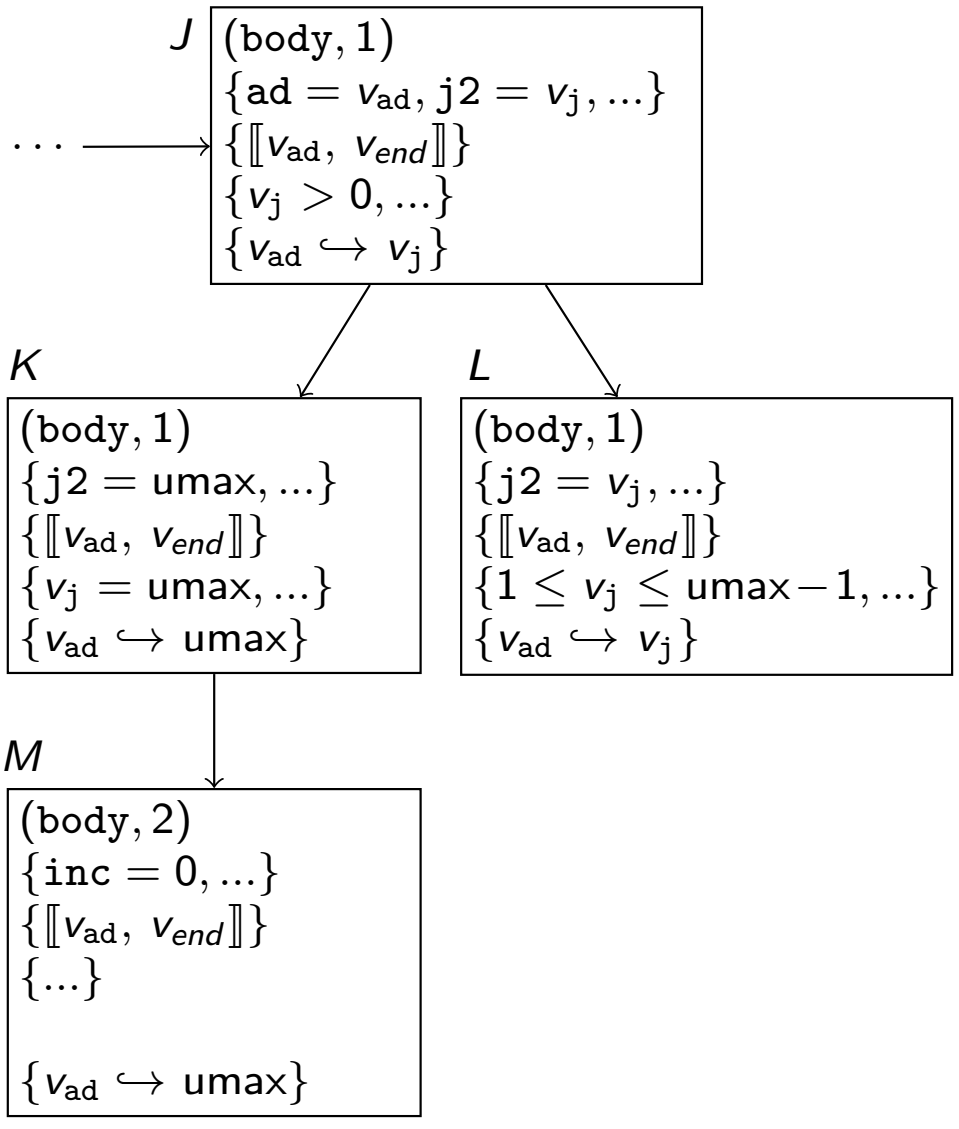


Symbolic execution rule for `x = add i32` t_1, t_2 where `x` $\in \mathcal{U}$

- set `x` to $PV_u(t_1) + PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \text{umax}_{32}$
- set `x` to $PV_u(t_1) + PV_u(t_2) - 2^{32}$ if $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) > \text{umax}_{32}$
- otherwise: case analysis

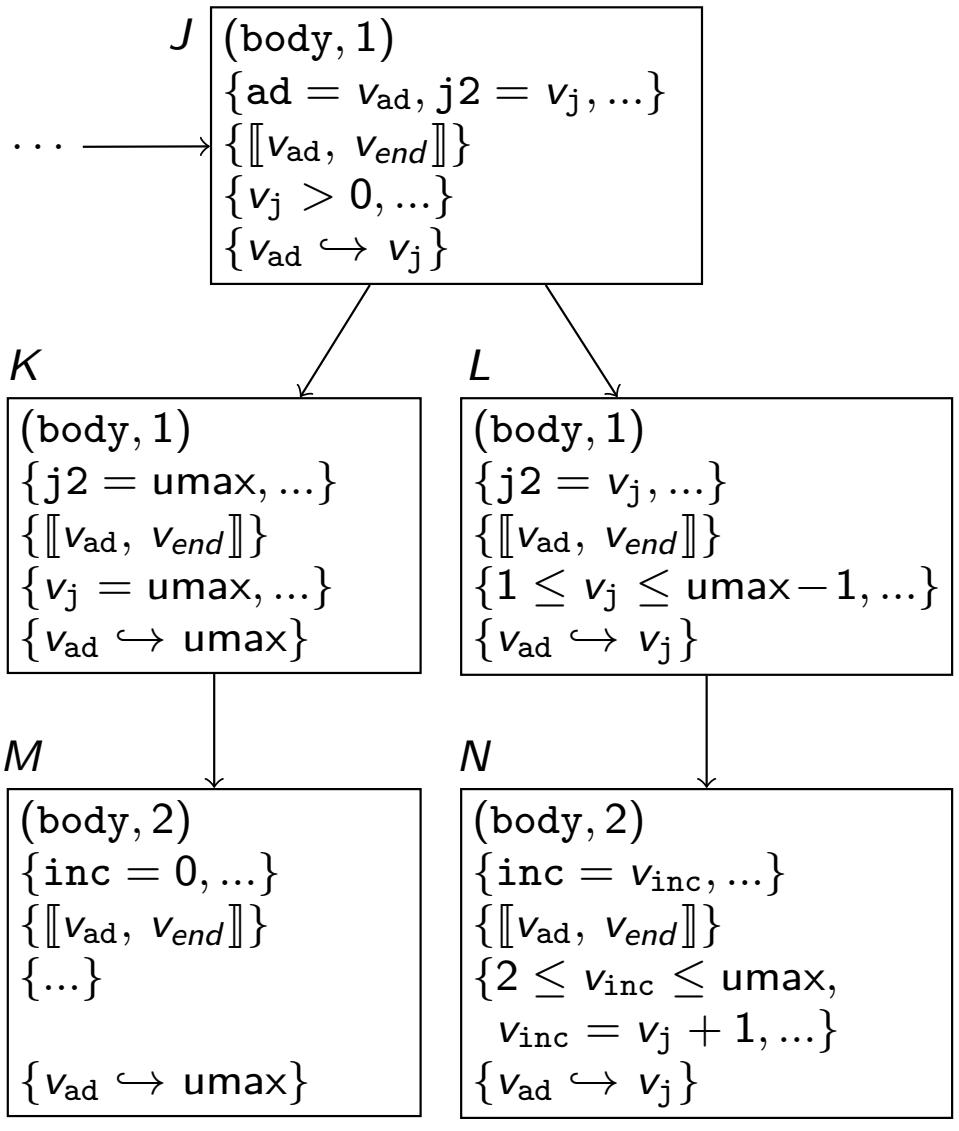
Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



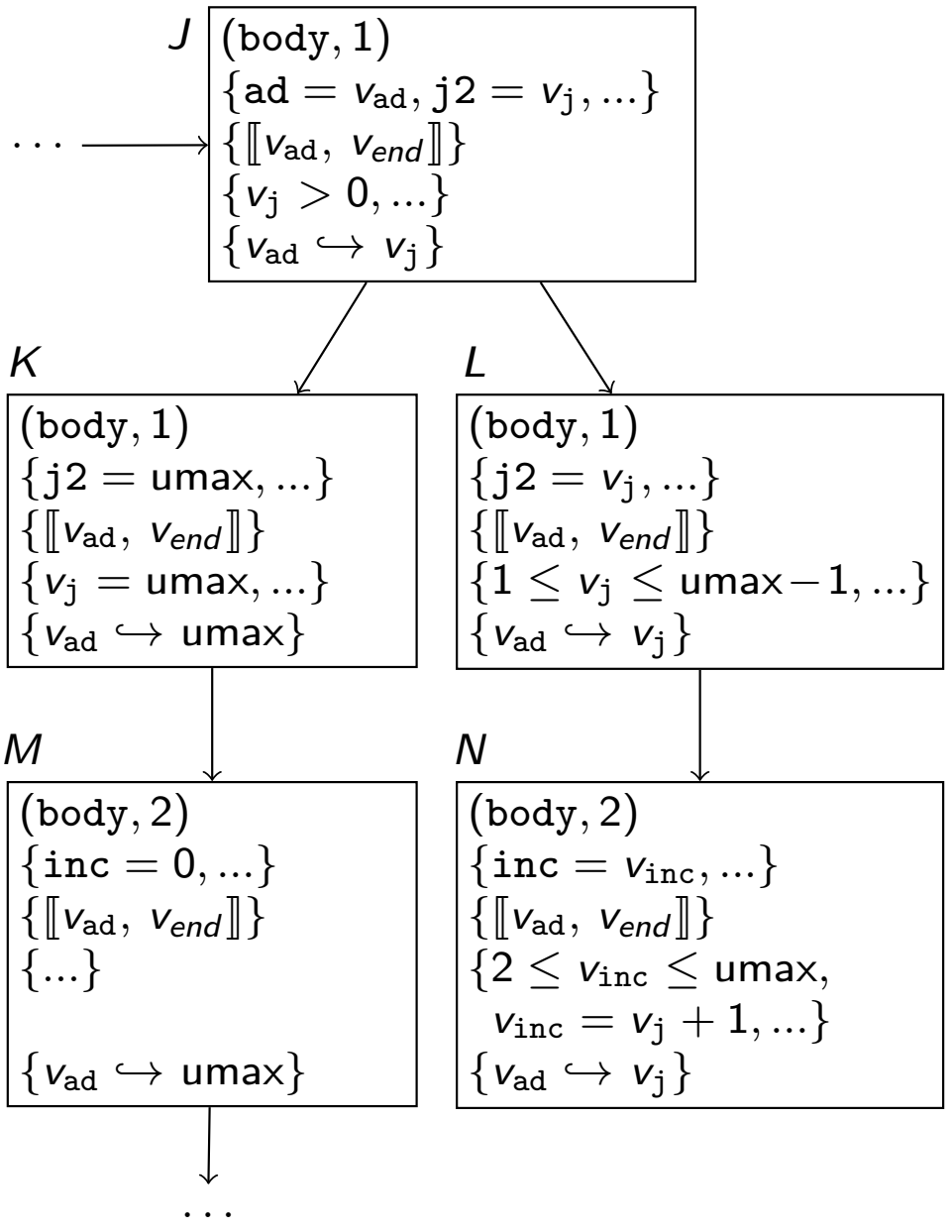
Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



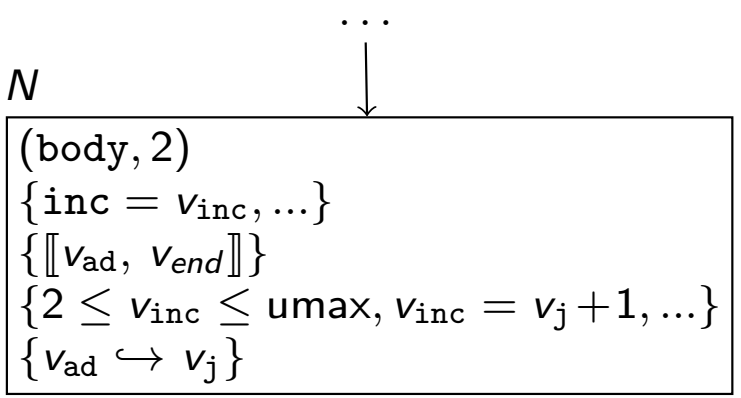
Addition

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



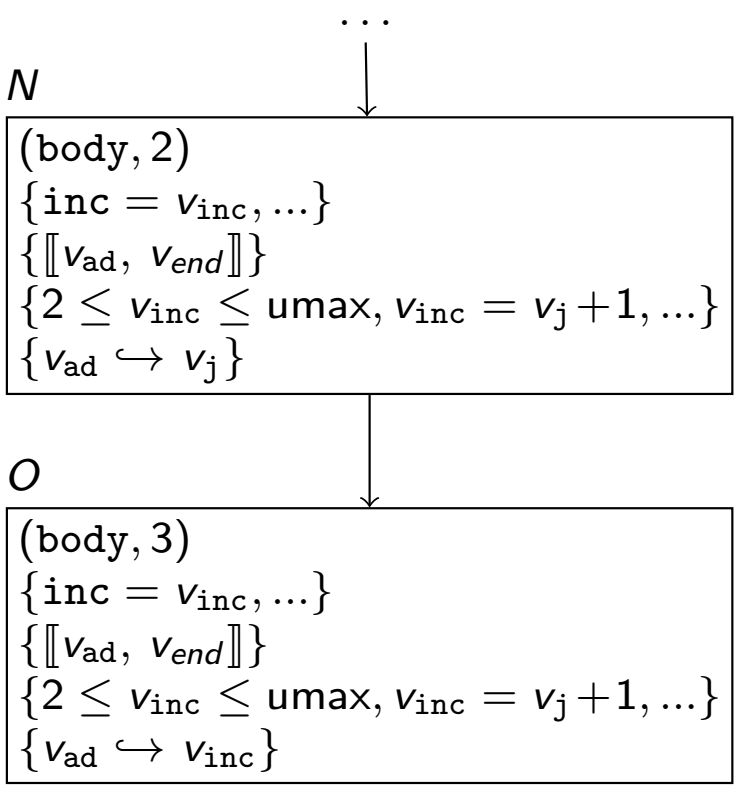
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



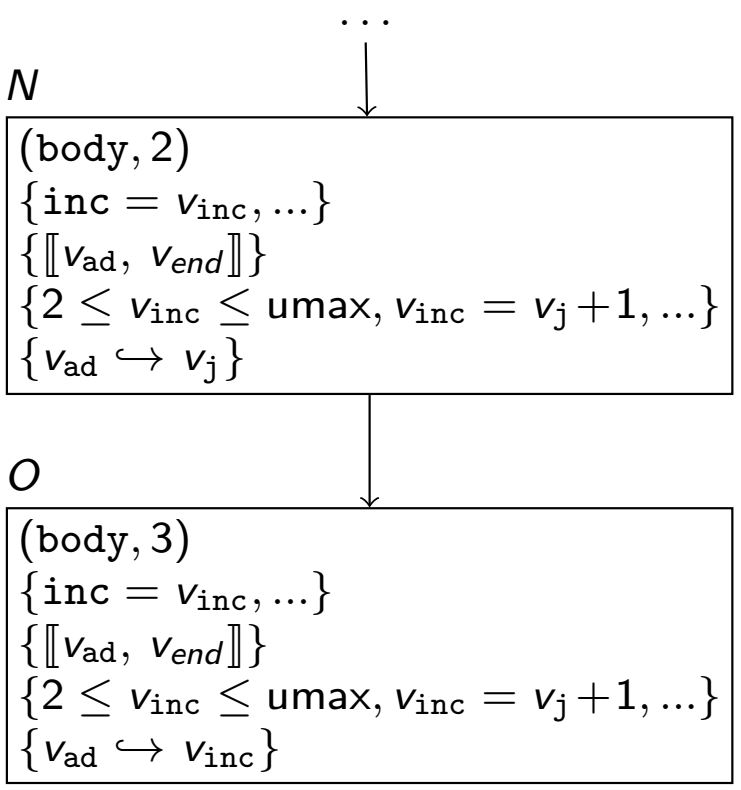
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



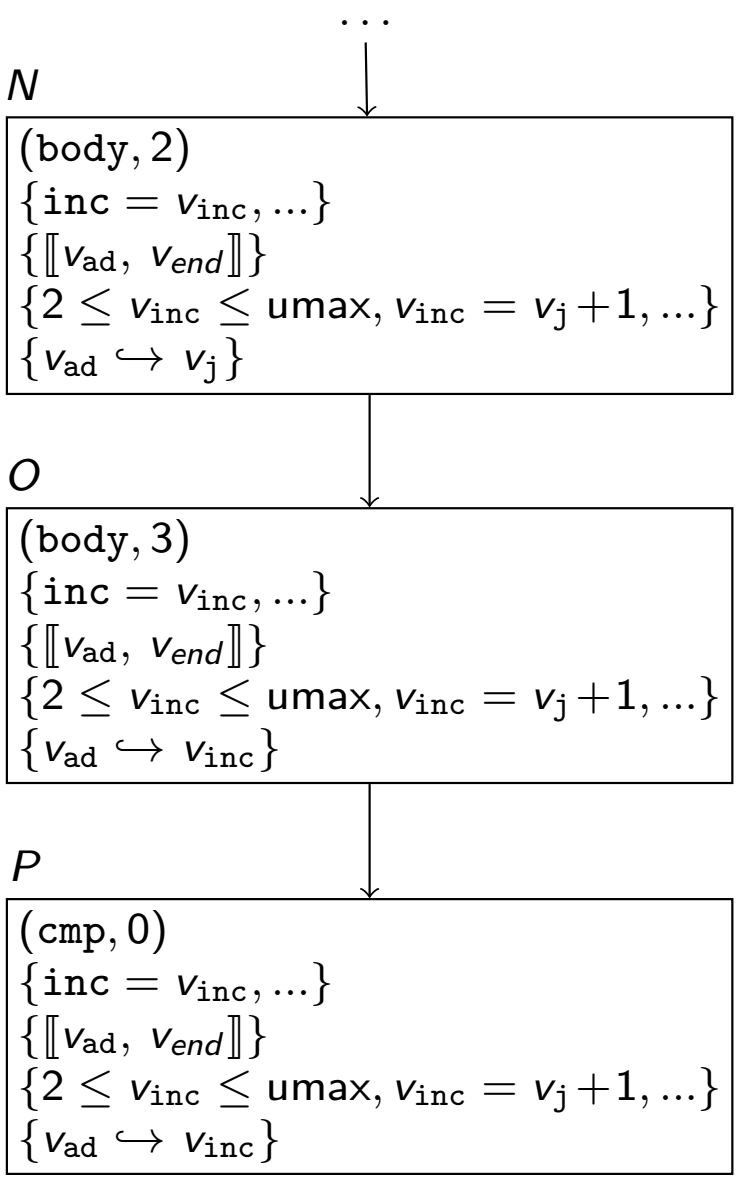
Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Symbolic Execution

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

P

$(\texttt{cmp}, 0)$

$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$

$\cdots \longrightarrow \{[\![v_{\texttt{ad}}, v_{end}]\!]\}$

$\{2 \leq v_{\texttt{inc}} \leq \mathsf{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$

$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

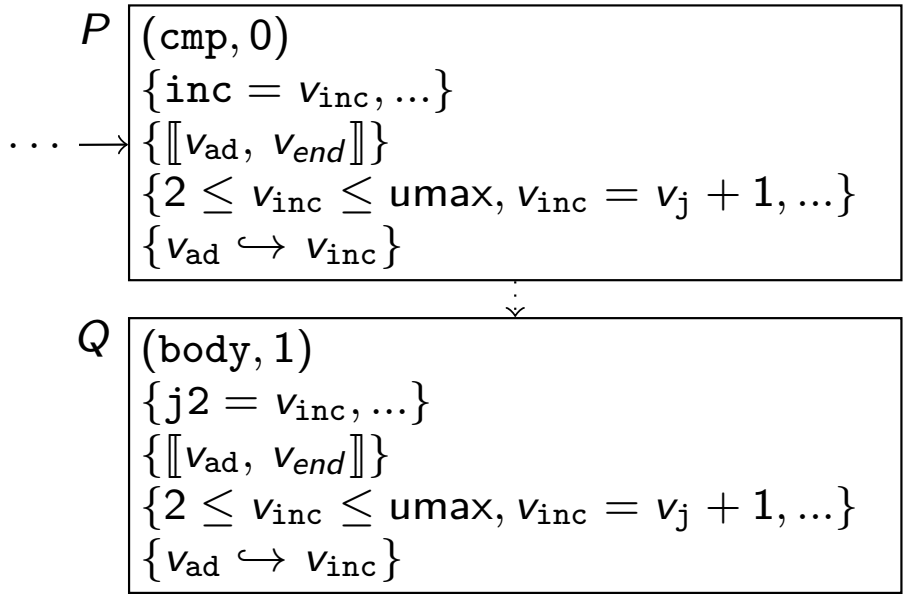
Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

$$\cdots \longrightarrow P \begin{array}{|l} (\mathtt{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![v_{\mathtt{ad}}, v_{end}]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}$$

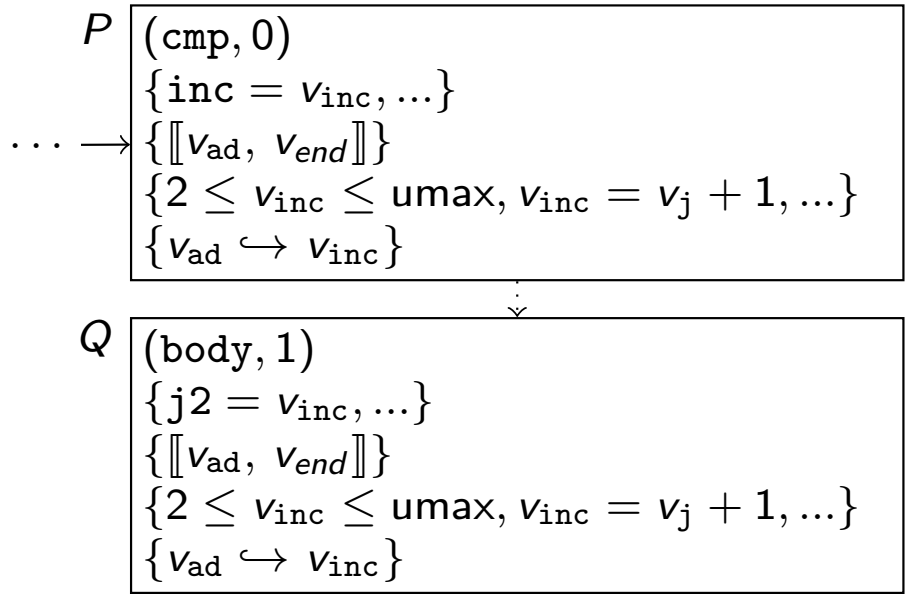
Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



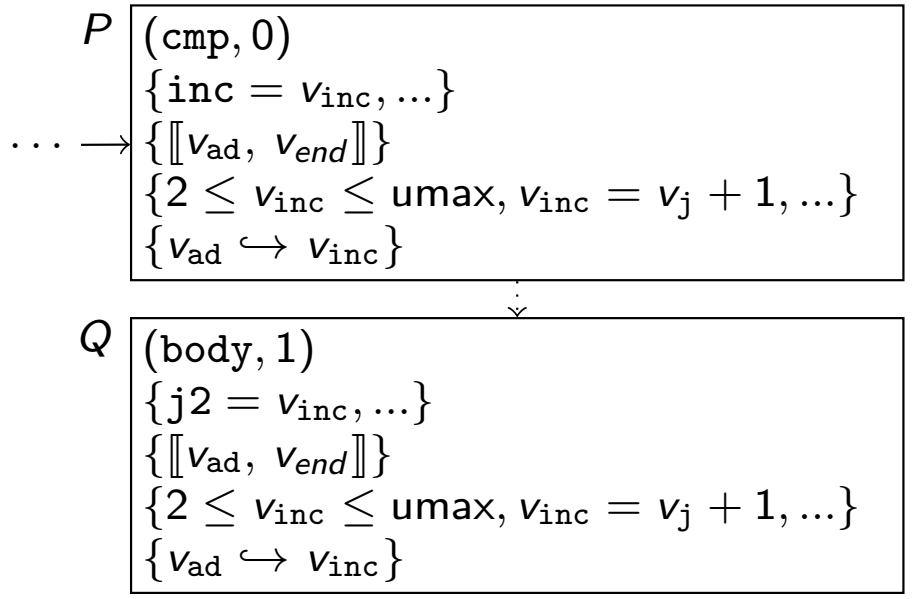
Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



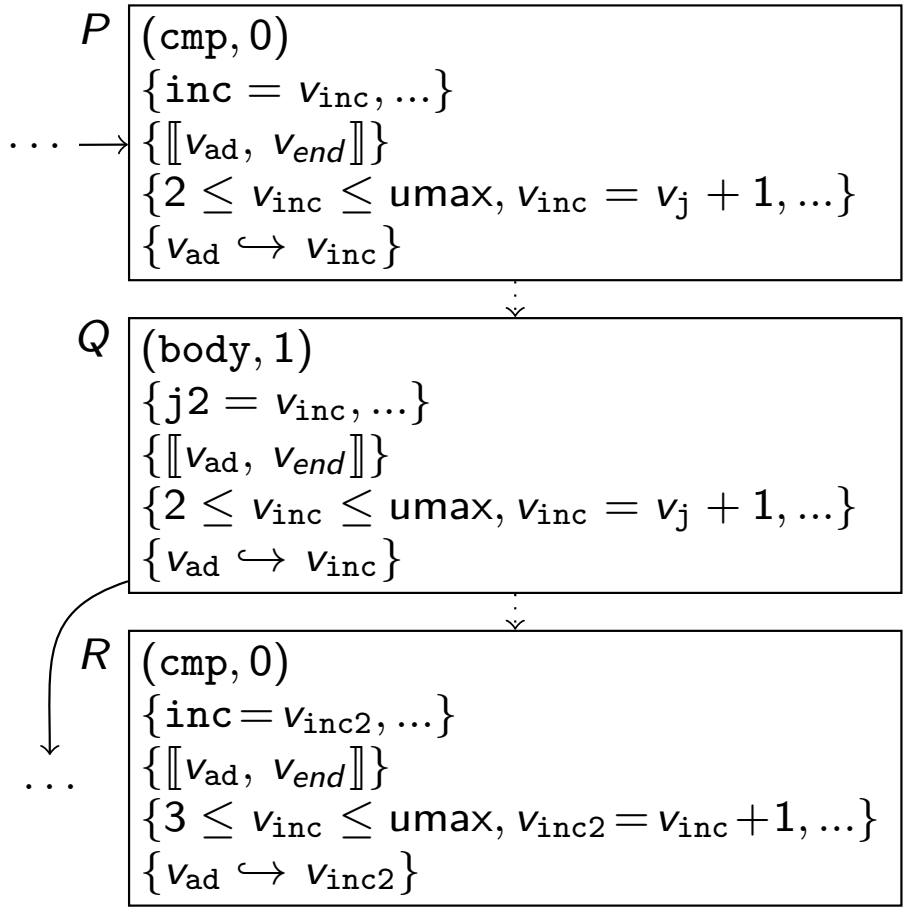
Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



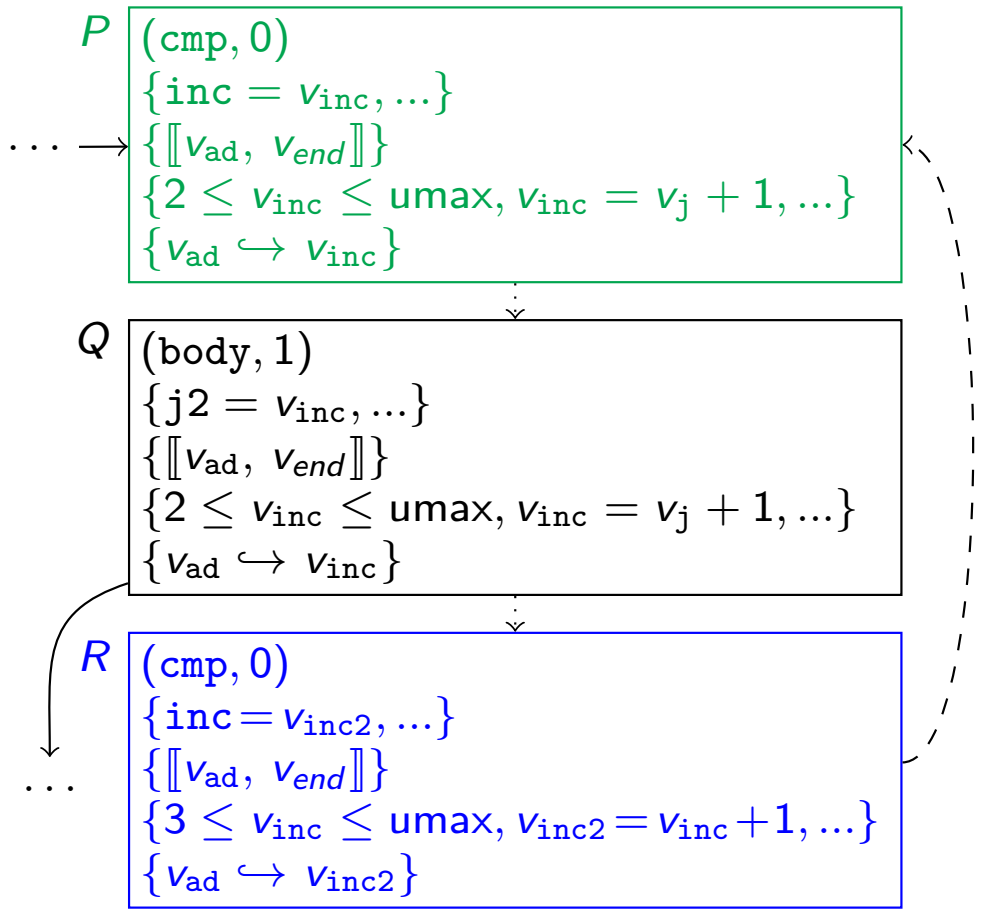
Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



Generalization

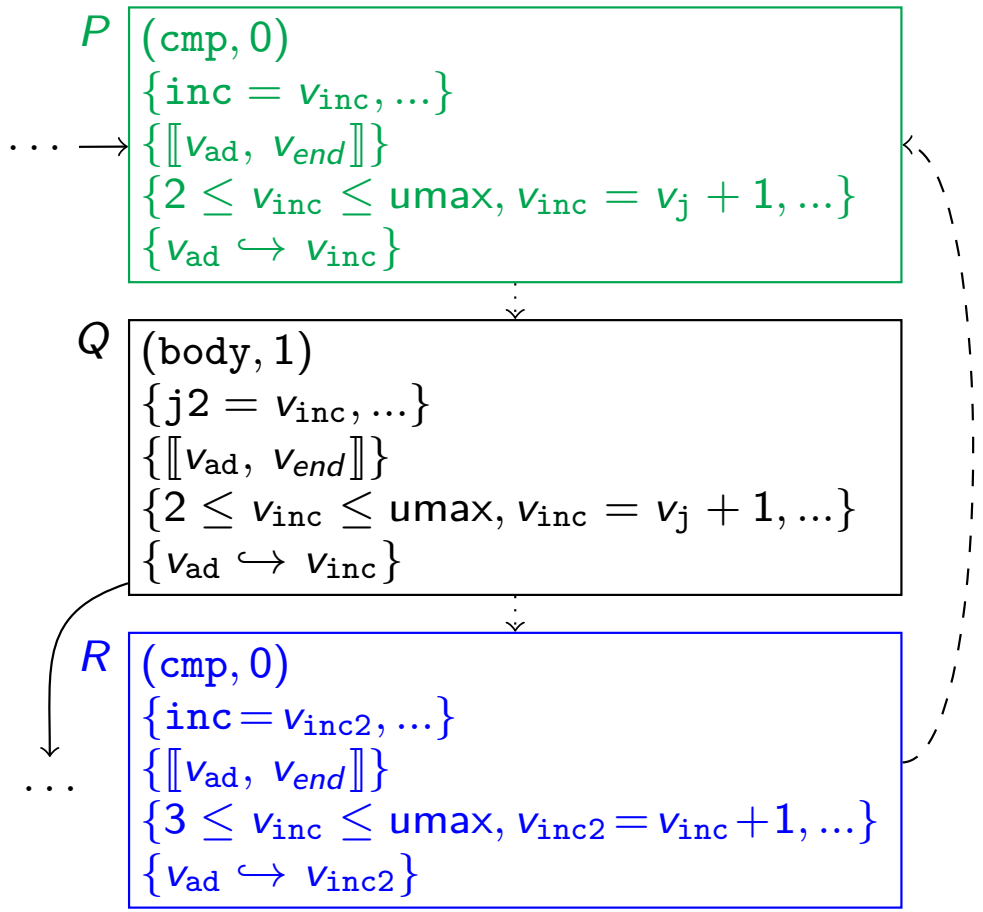
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```



P is *generalization* of R with μ

Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

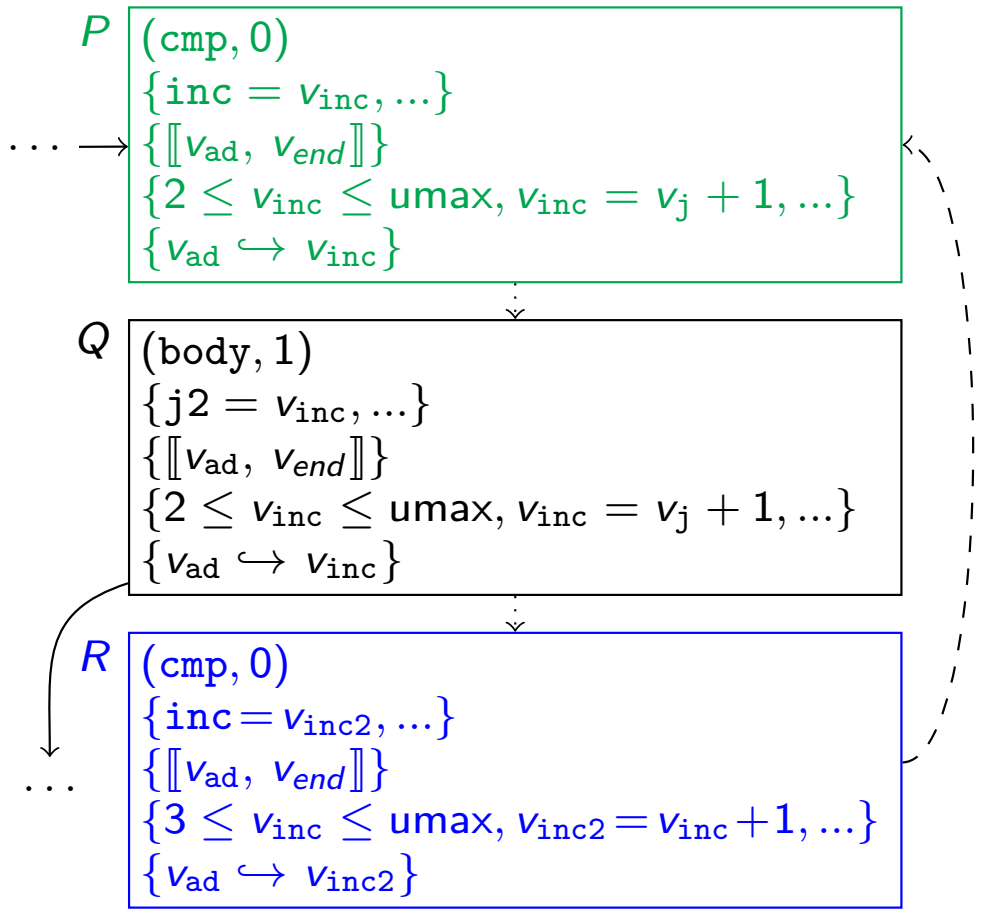


P is *generalization* of R with μ

- $\mu(PV_P(\mathtt{x})) = PV_R(\mathtt{x})$ for all program variables $\mathtt{x}$

Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

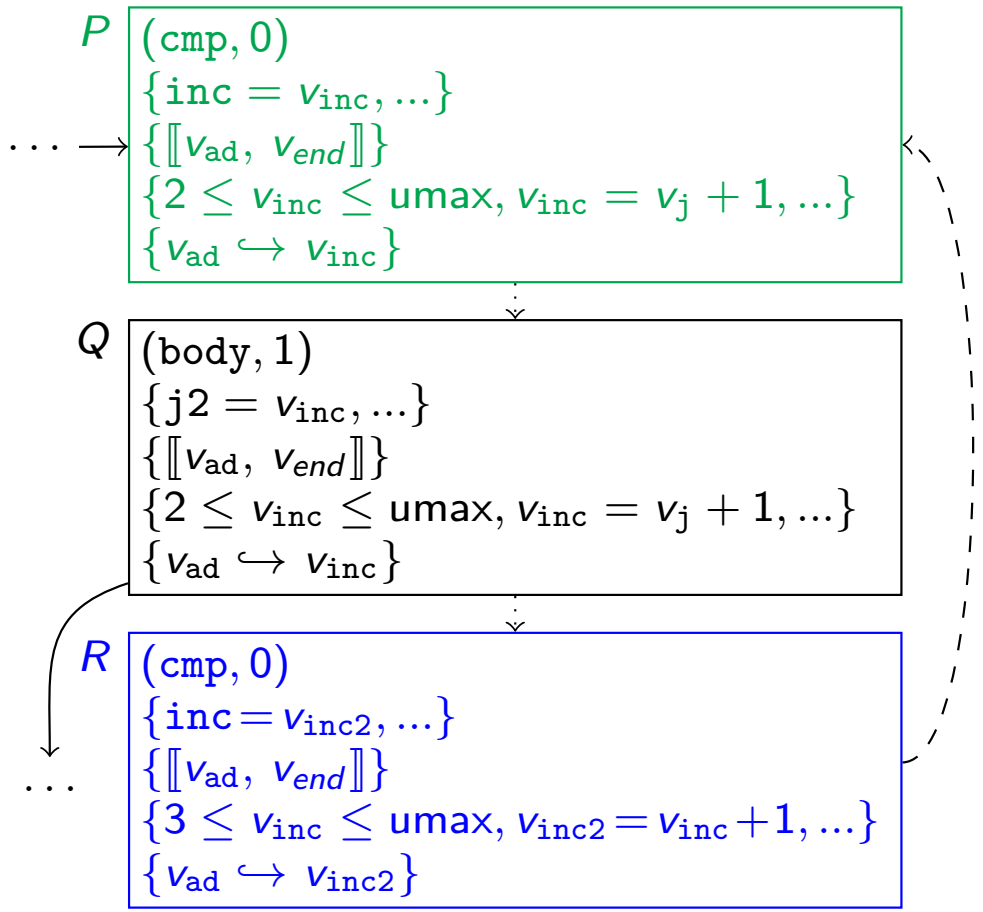


P is *generalization* of R with $\mu(v_j) = v_{inc}$, $\mu(v_{inc}) = v_{inc2}$

- $\mu(PV_P(x)) = PV_R(x)$ for all program variables x

Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

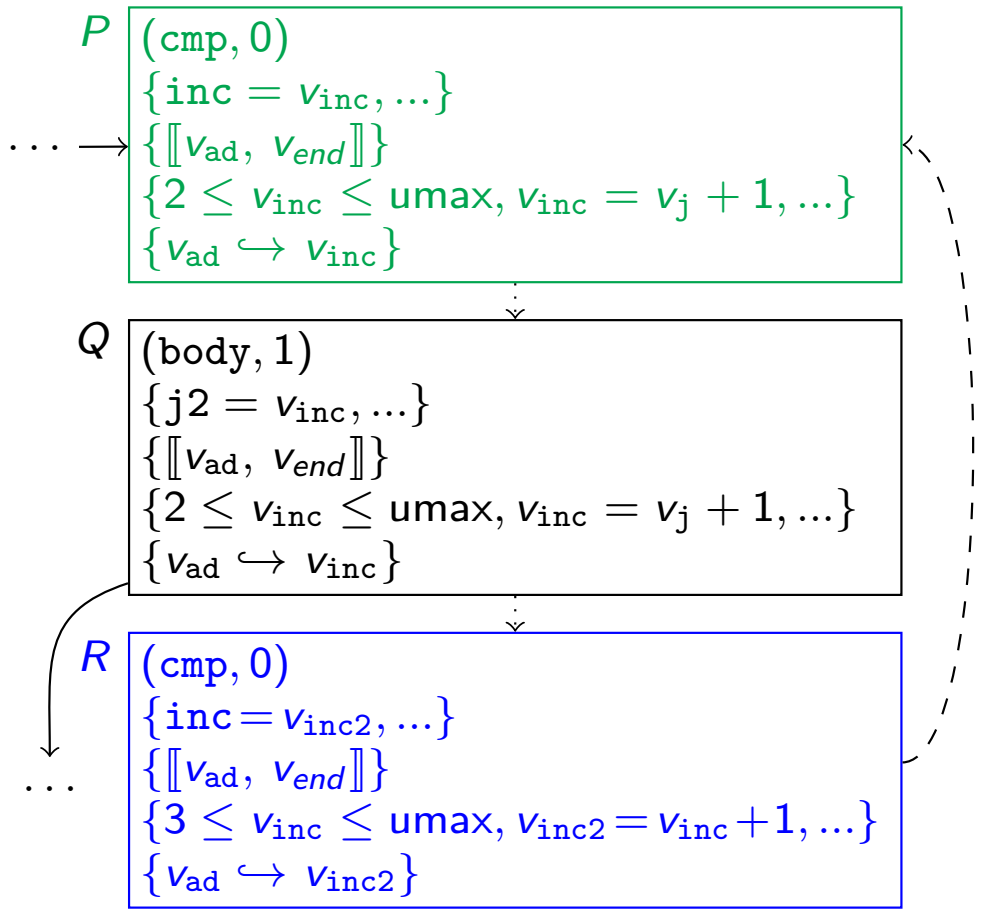


P is *generalization* of R with $\mu(v_j) = v_{inc}$, $\mu(v_{inc}) = v_{inc2}$

- $\mu(PV_P(x)) = PV_R(x)$ for all program variables x
- $[\![v_1, v_2]\!] \in AL_P$ implies $[\![\mu(v_1), \mu(v_2)]\!] \in AL_R$

Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

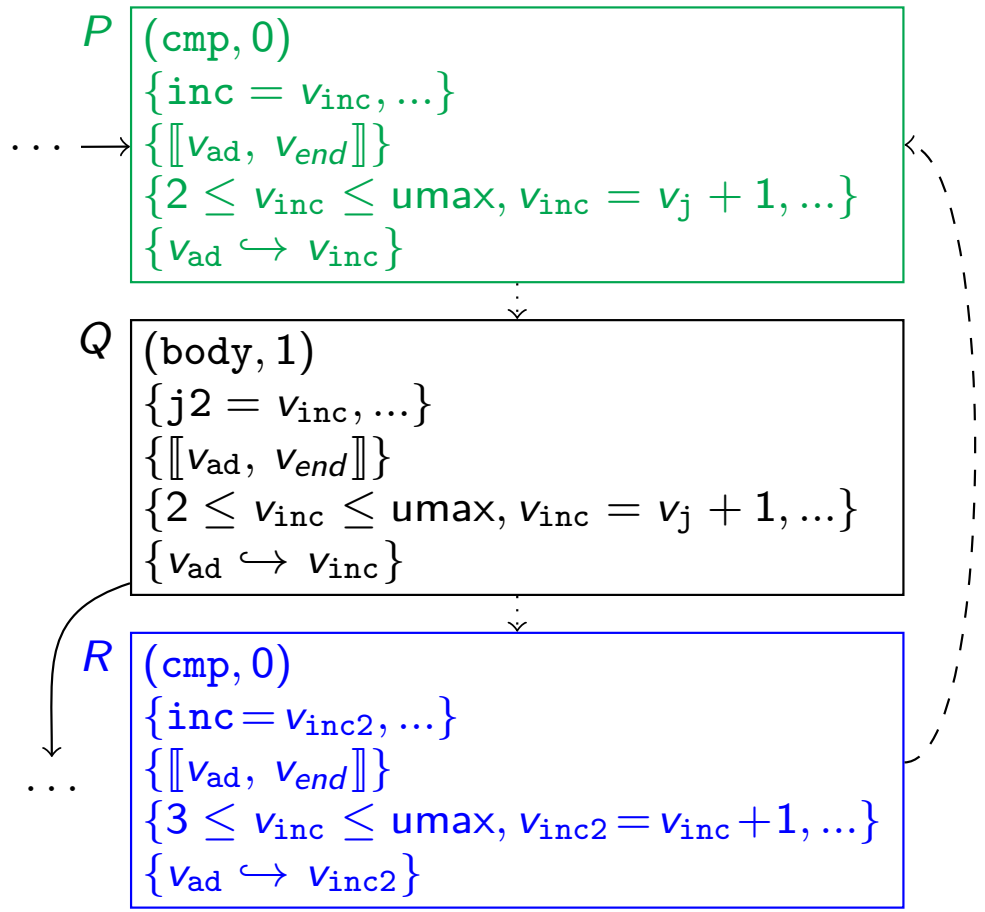


P is *generalization* of R with $\mu(v_j) = v_{inc}$, $\mu(v_{inc}) = v_{inc2}$

- $\mu(PV_P(x)) = PV_R(x)$ for all program variables x
- $[\![v_1, v_2]\!] \in AL_P$ implies $[\![\mu(v_1), \mu(v_2)]\!] \in AL_R$
- $\models \langle R \rangle \implies \mu(KB_P)$

Generalization

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

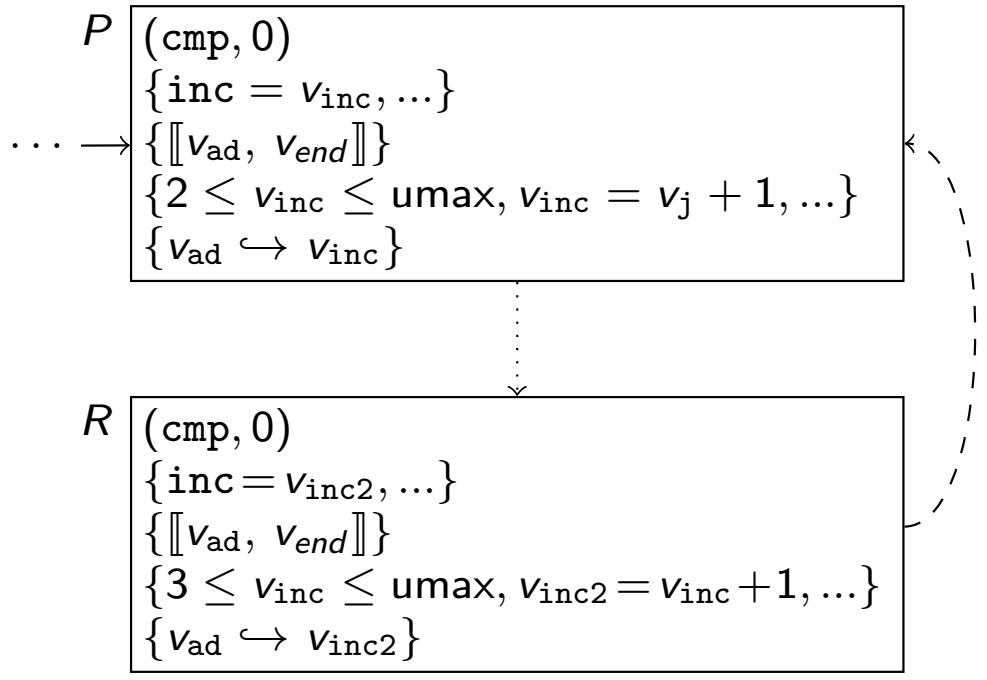


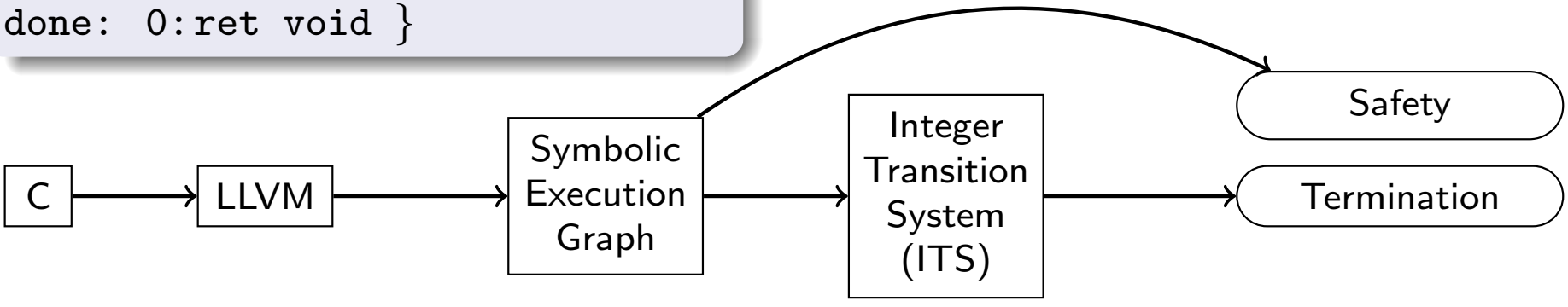
P is *generalization* of R with $\mu(v_j) = v_{inc}$, $\mu(v_{inc}) = v_{inc2}$

- $\mu(PV_P(x)) = PV_R(x)$ for all program variables x
- $[\![v_1, v_2]\!] \in AL_P$ implies $[\![\mu(v_1), \mu(v_2)]\!] \in AL_R$
- $\models \langle R \rangle \implies \mu(KB_P)$
- $v_1 \hookrightarrow v_2 \in PT_P$ implies $\mu(v_1) \hookrightarrow \mu(v_2) \in PT_R$

Safety

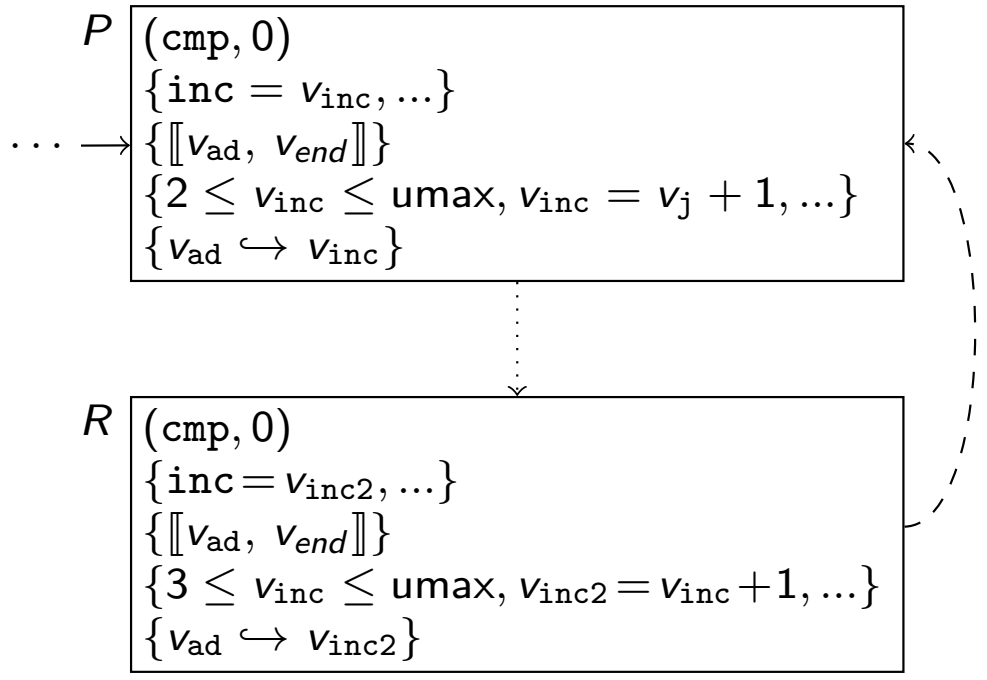
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

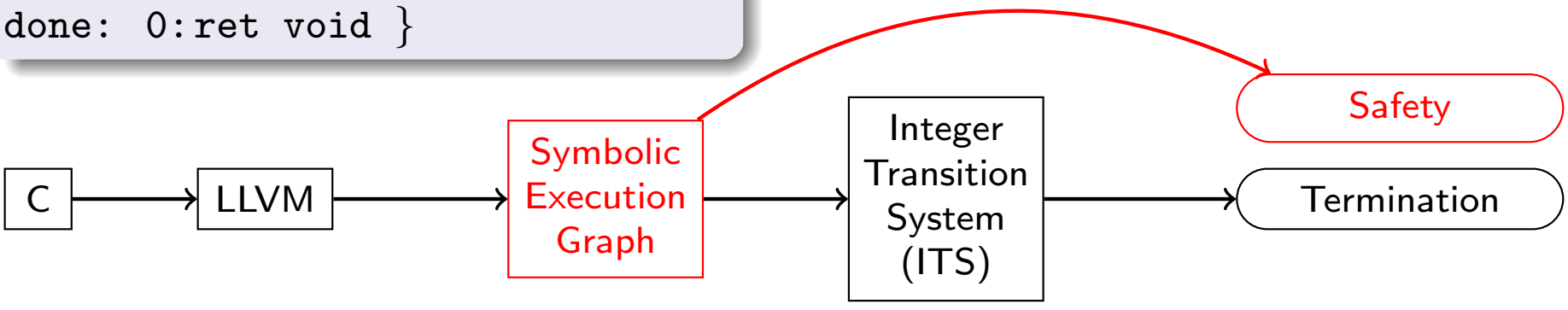




Safety

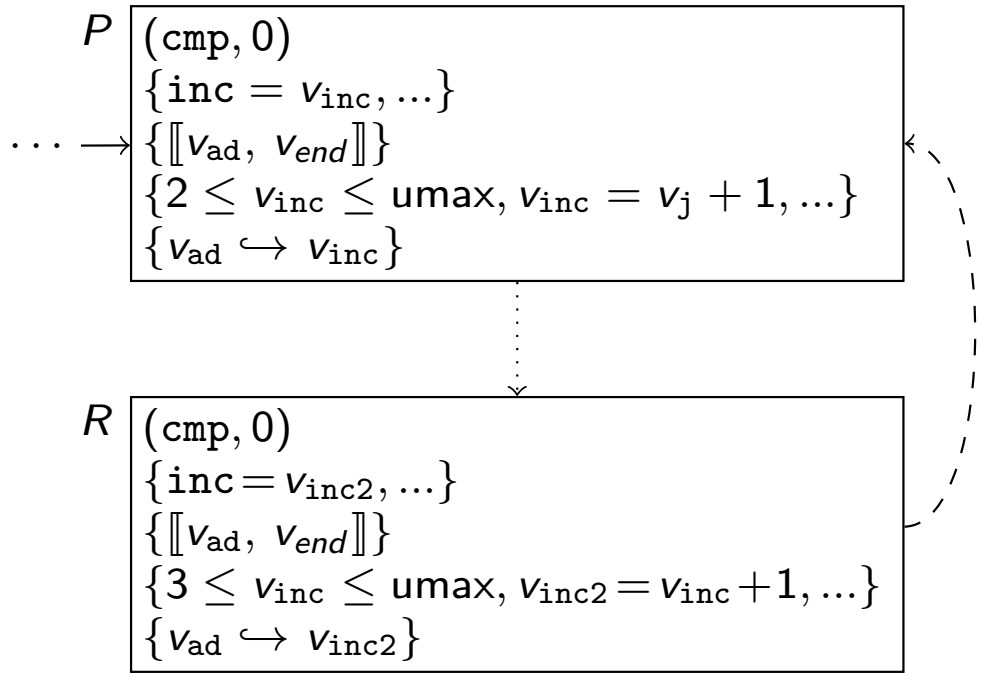
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

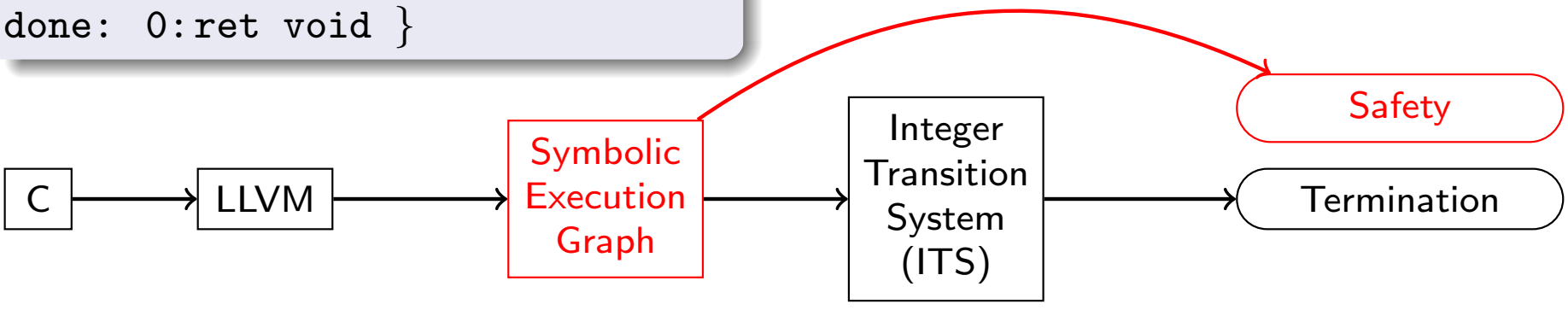




Safety

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

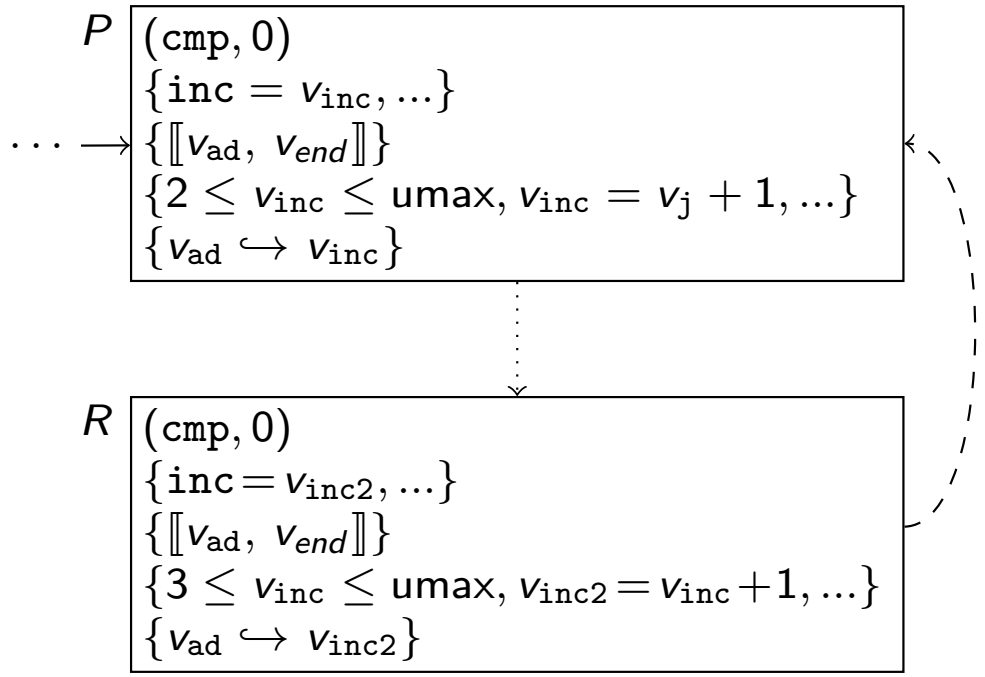


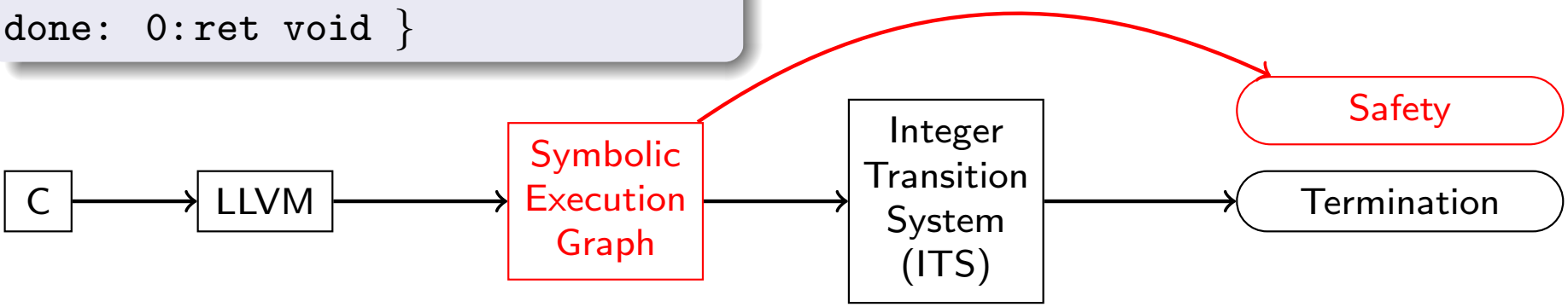


- Symbolic execution graph complete if leaves correspond to `return`

Safety

```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

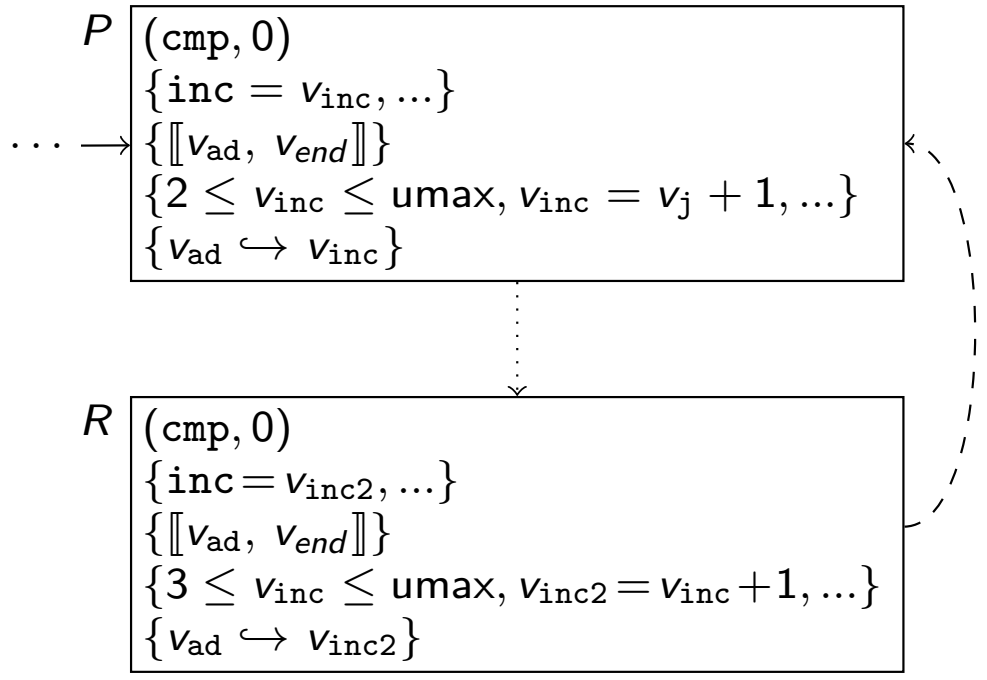


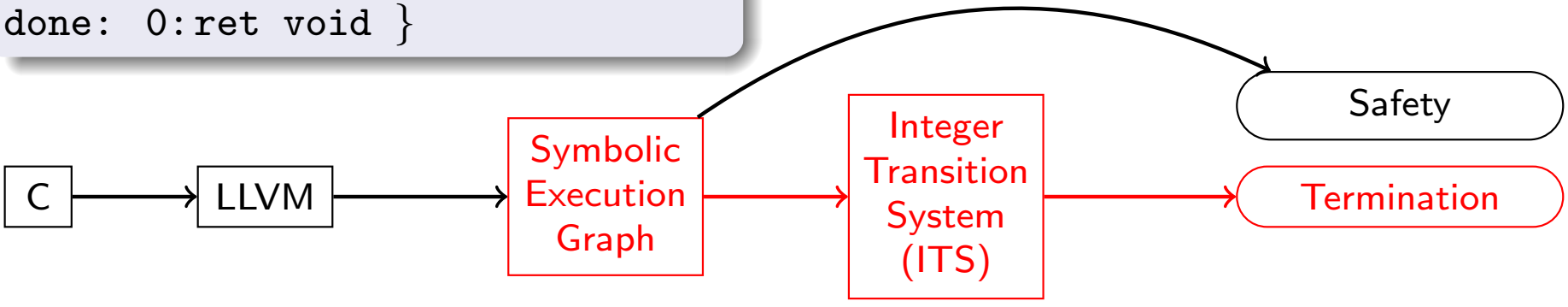


- Symbolic execution graph complete if leaves correspond to `return`
- Complete symbolic execution graph without *ERR* $\Longrightarrow$ Safety

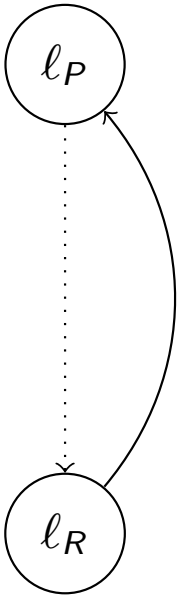
Termination

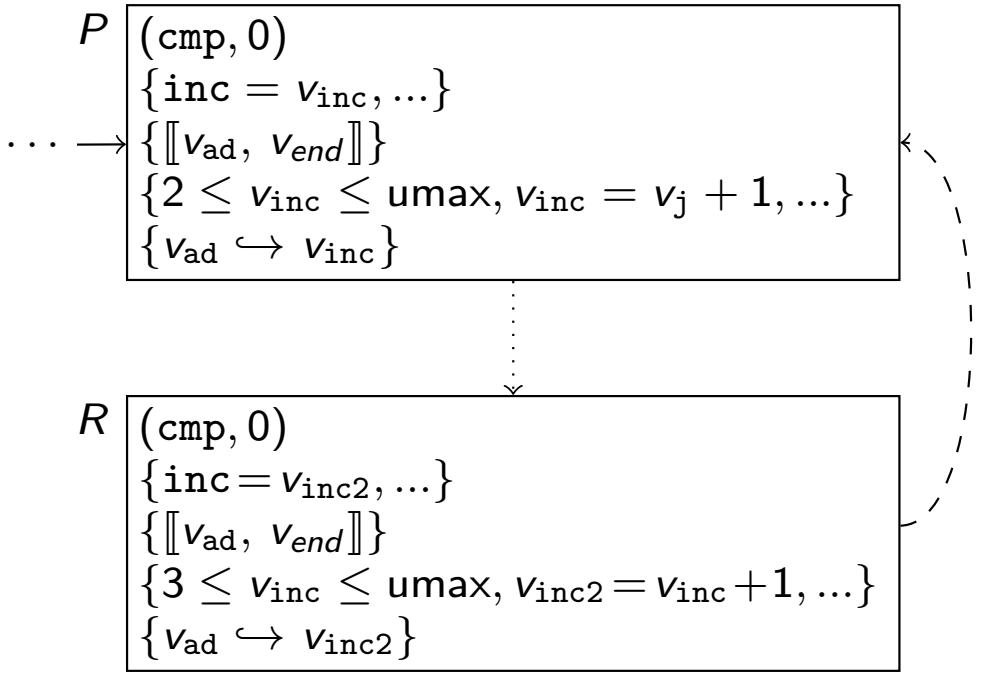
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32 inc, i32* ad
       3:br label cmp
done:  0:ret void }
```

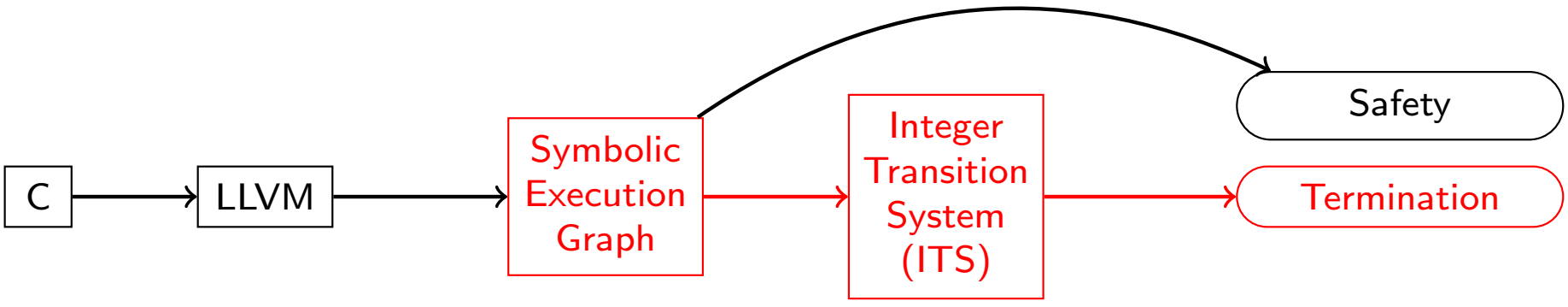




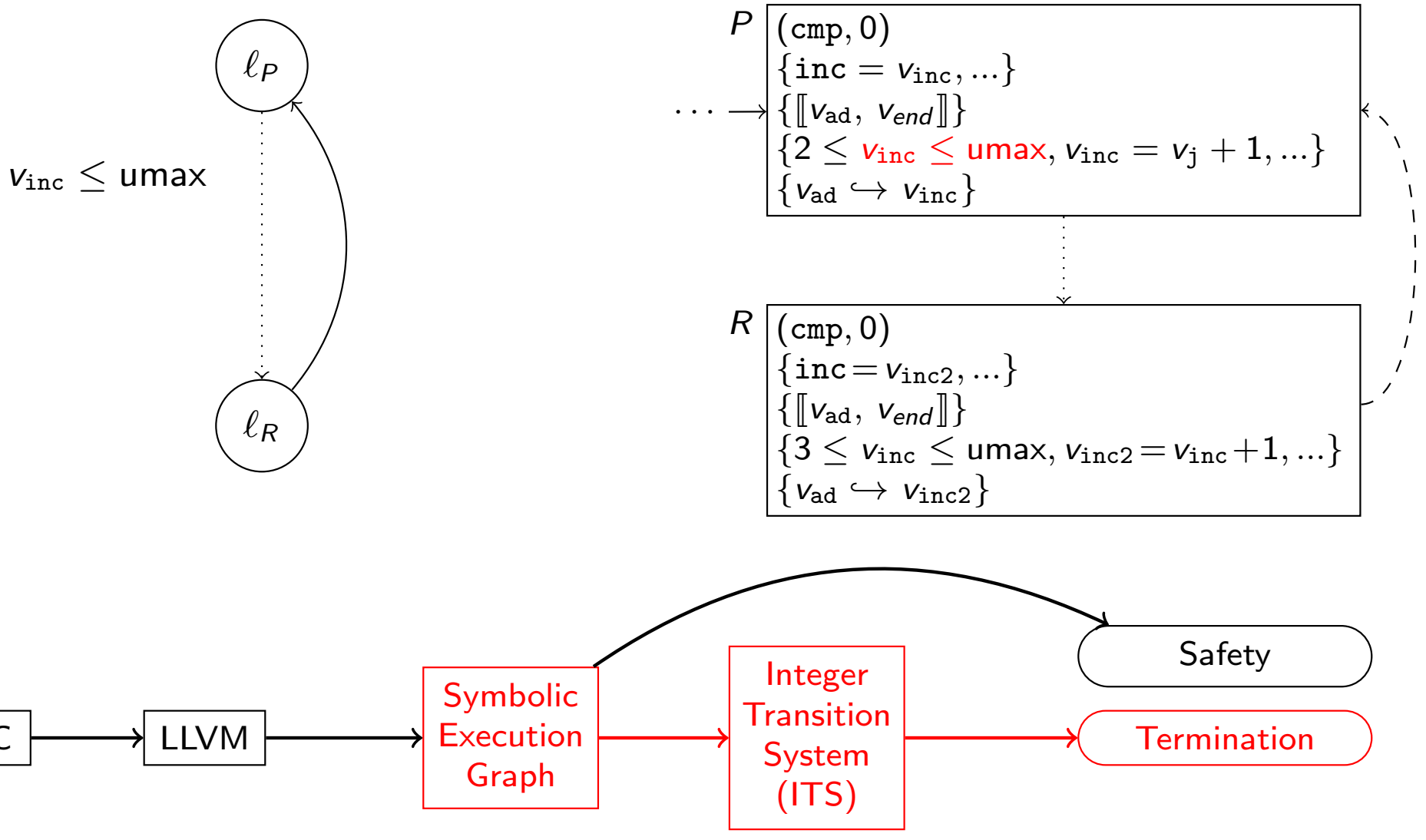
Termination



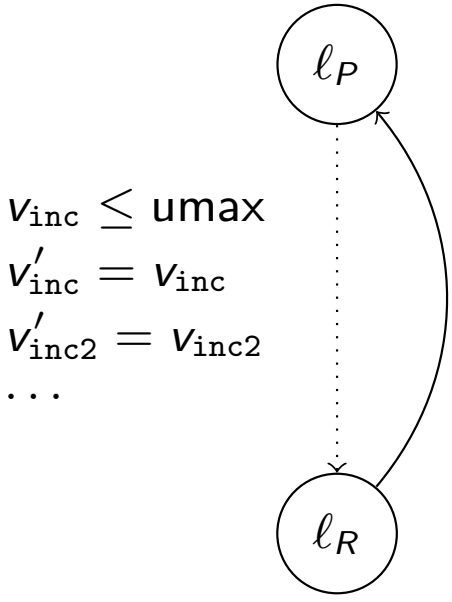


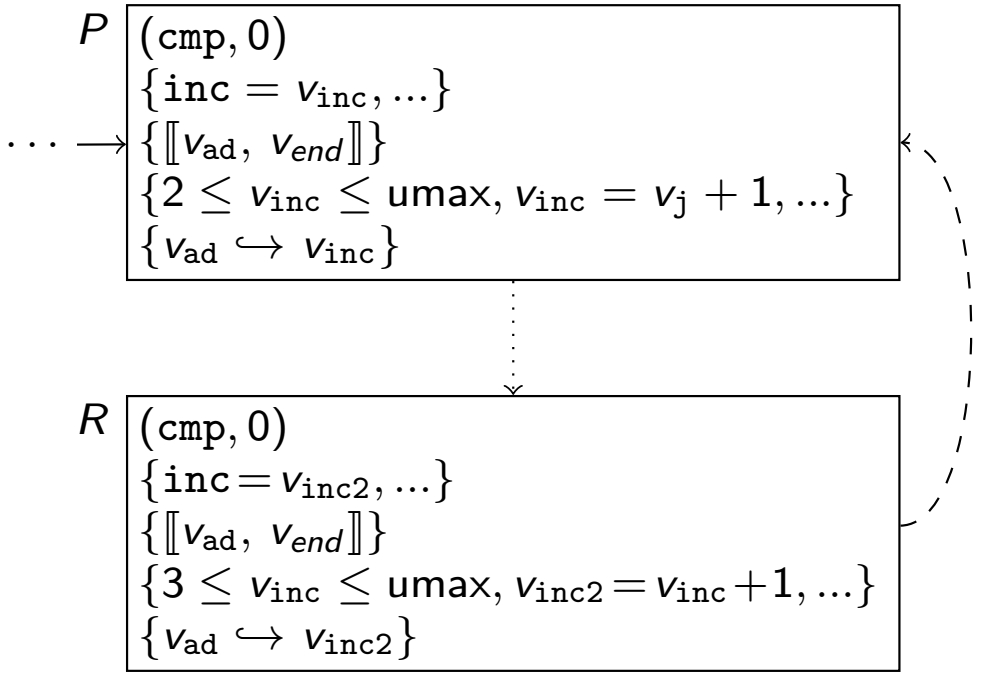


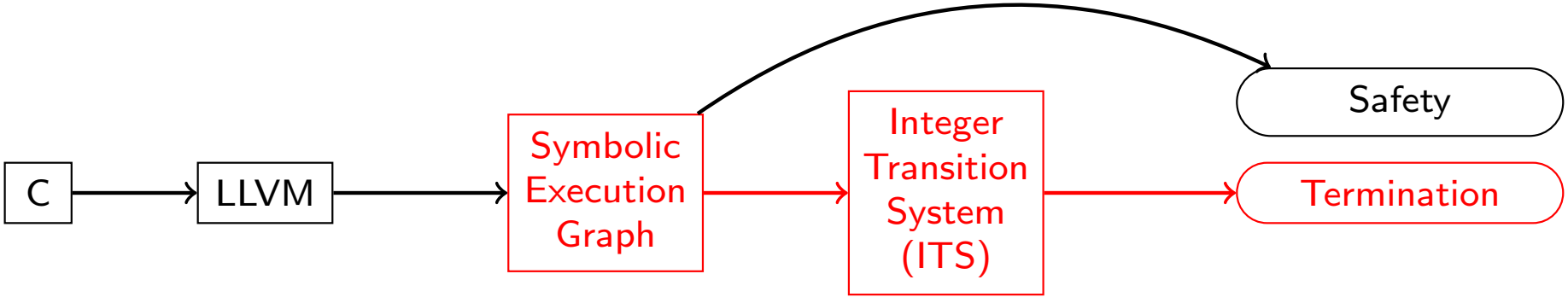
Termination



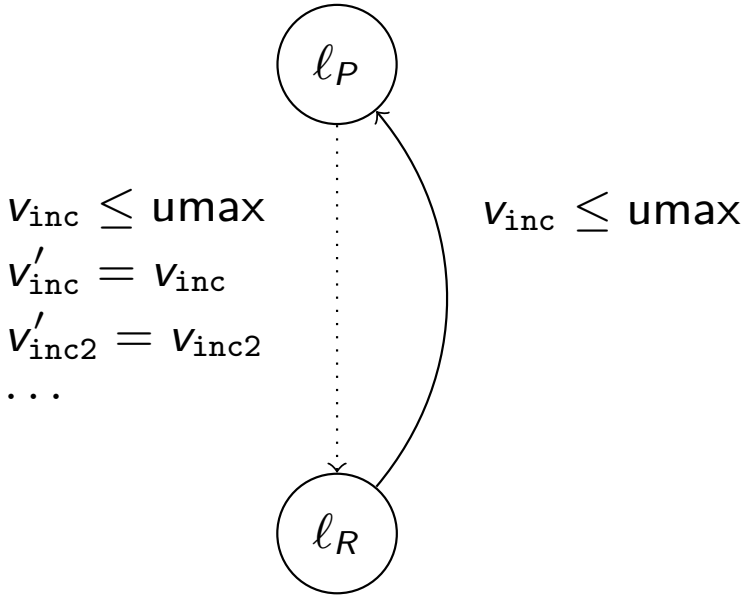
Termination

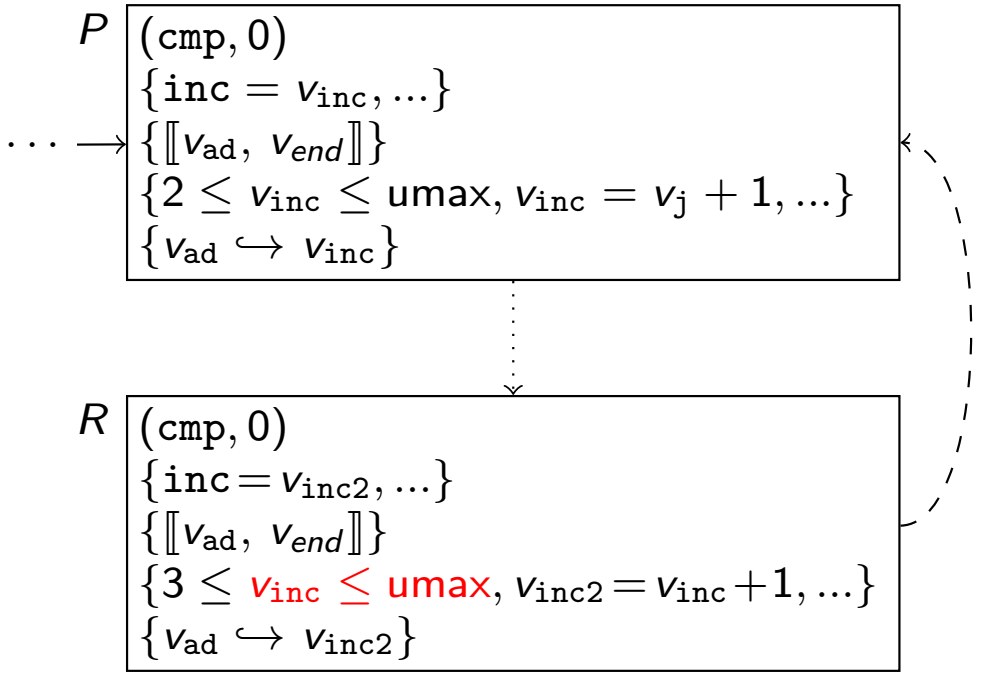


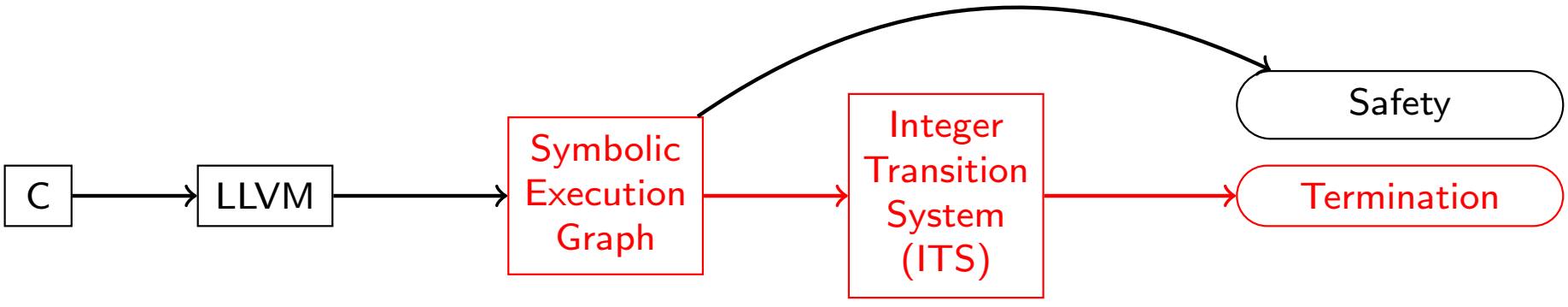




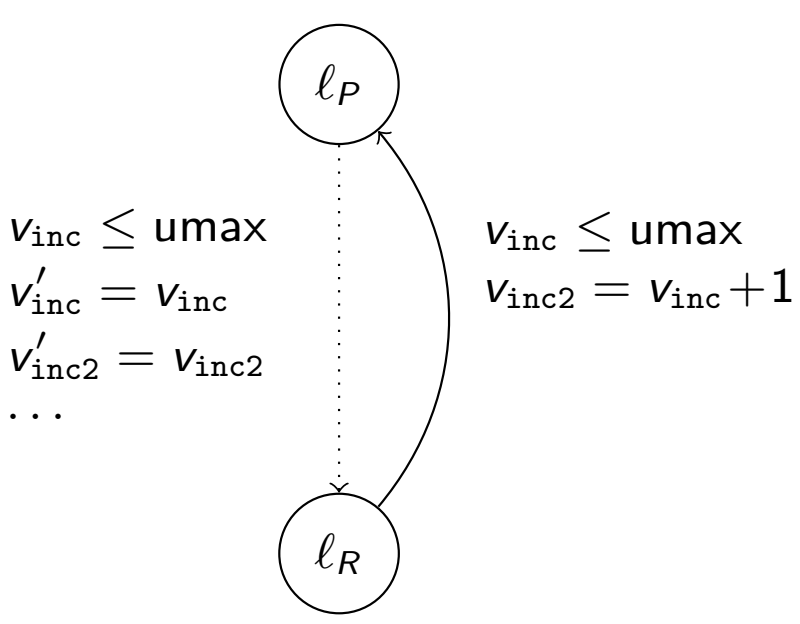
Termination

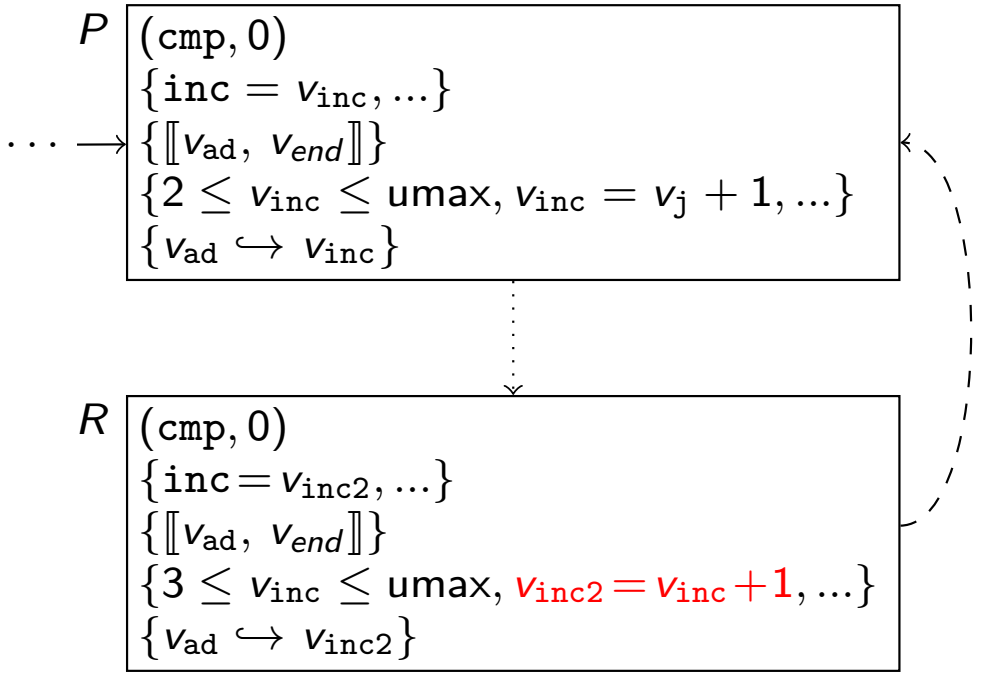


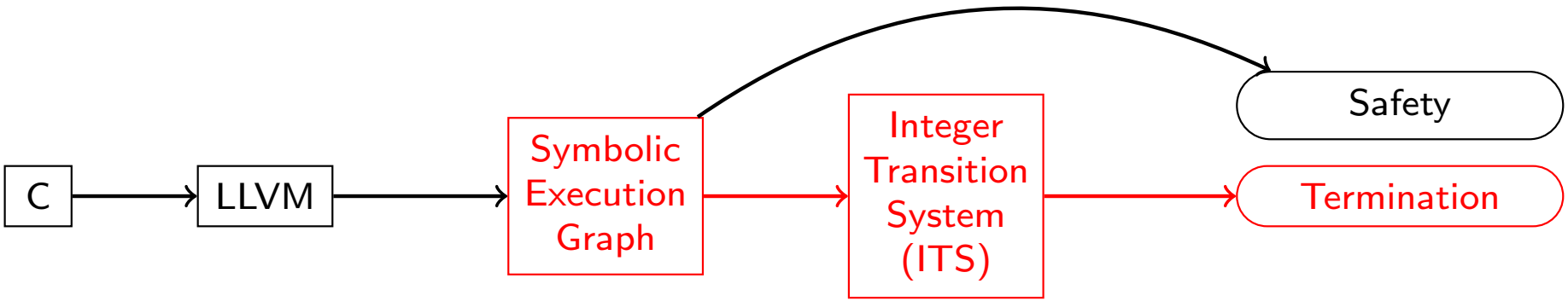




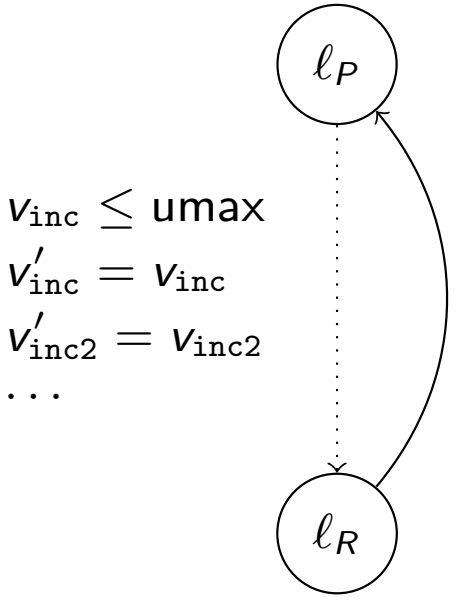
Termination

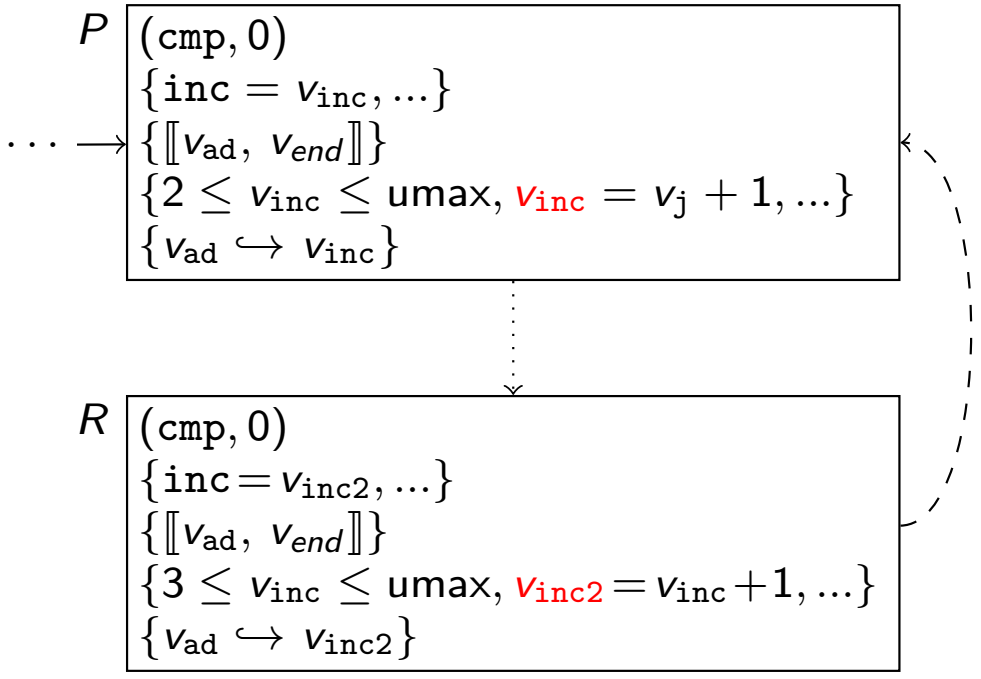


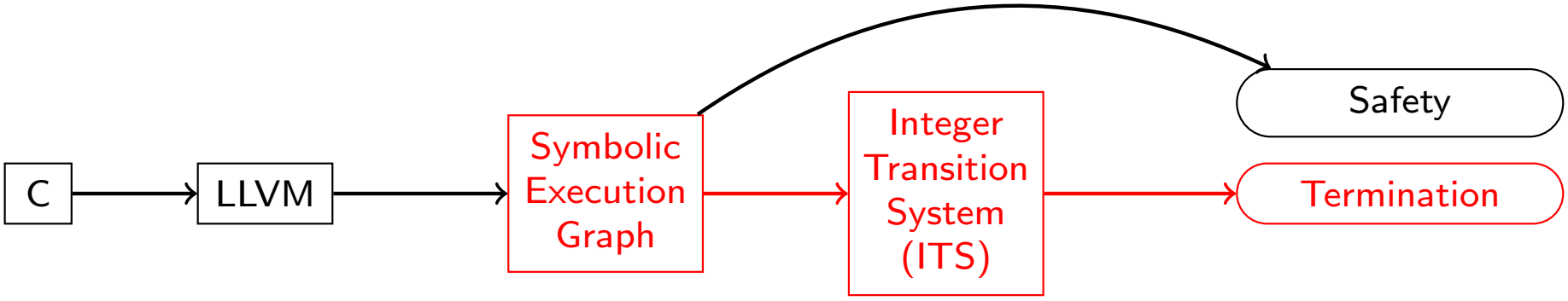




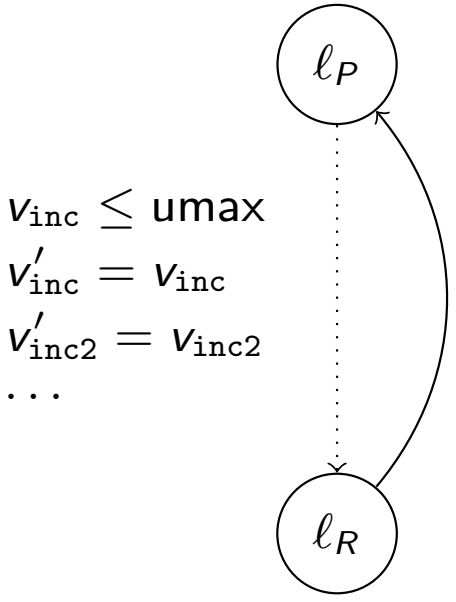
Termination

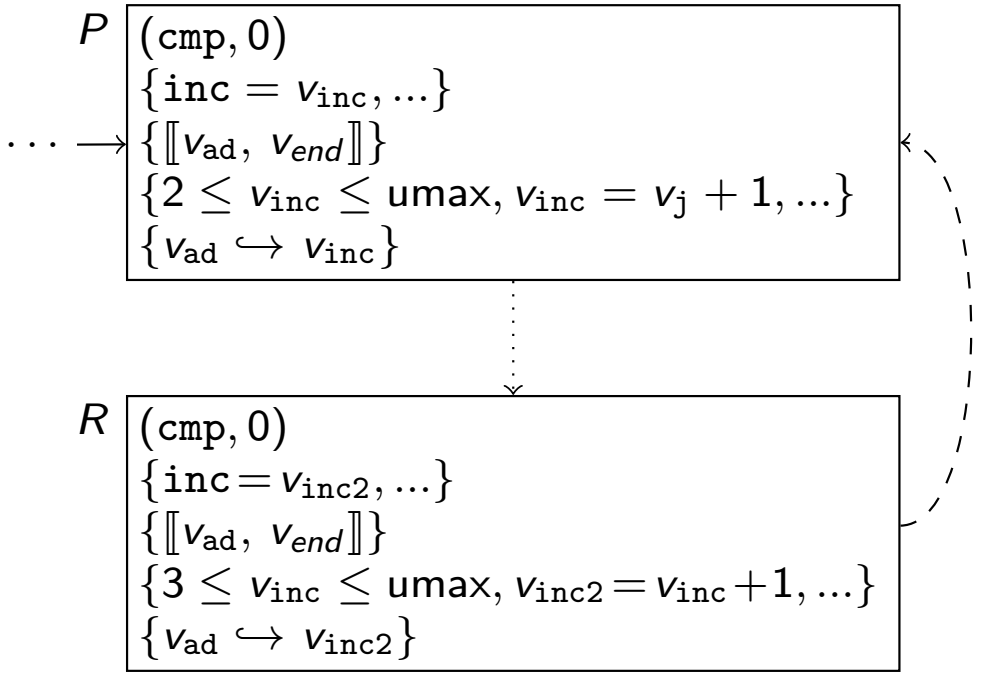


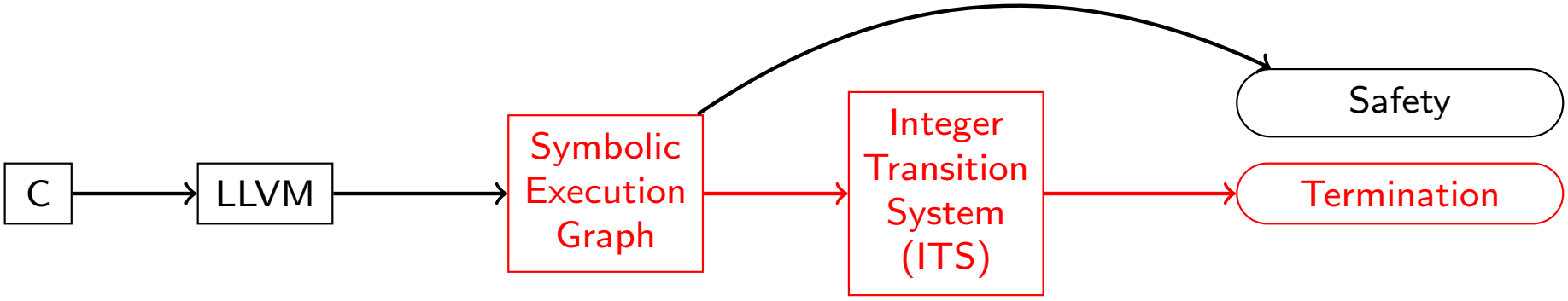




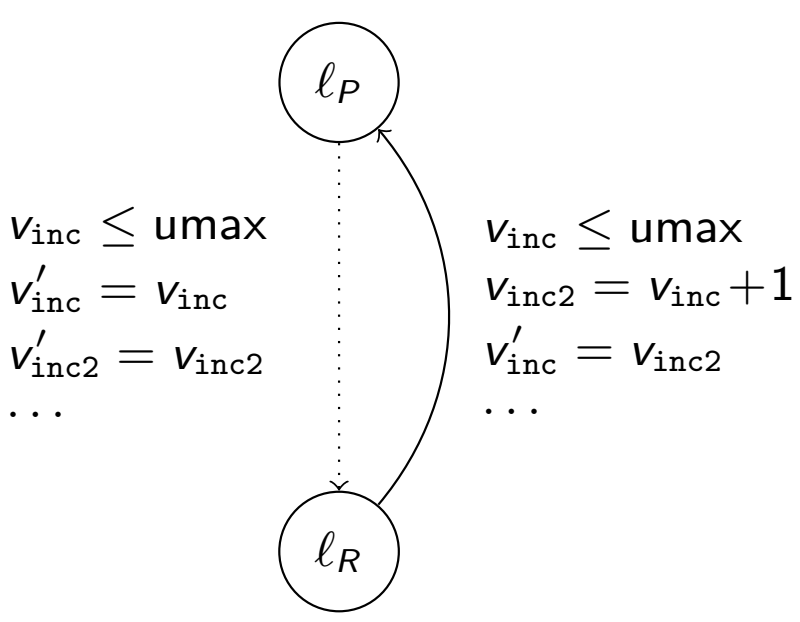
Termination

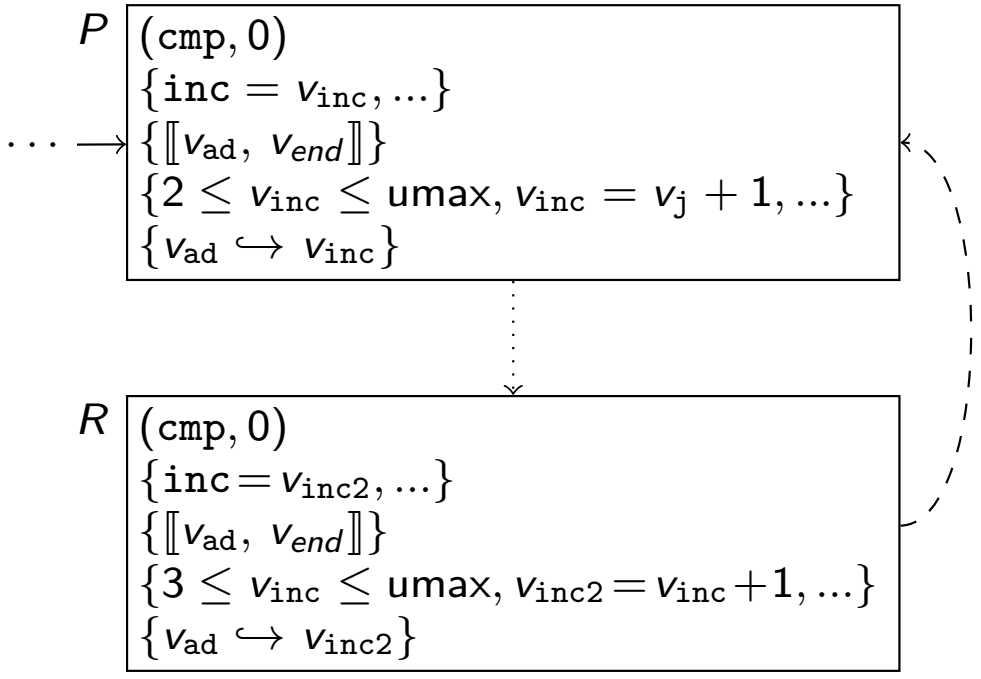


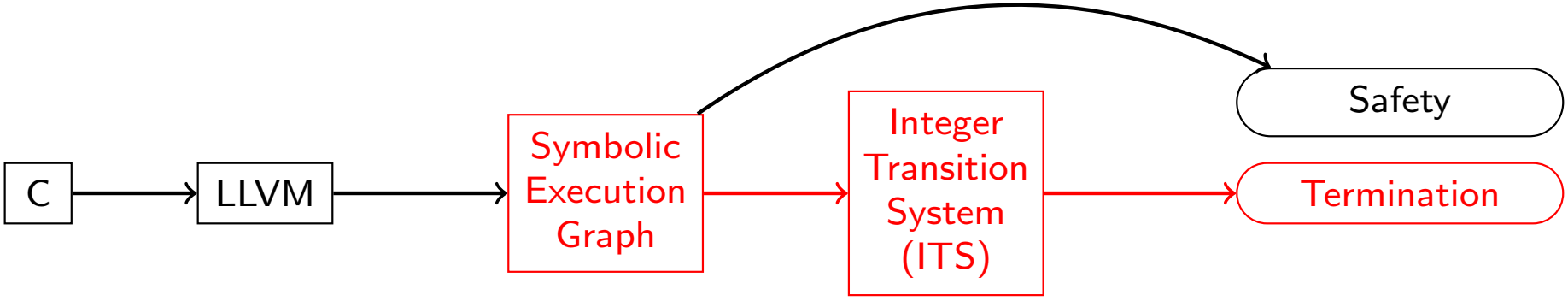




Termination

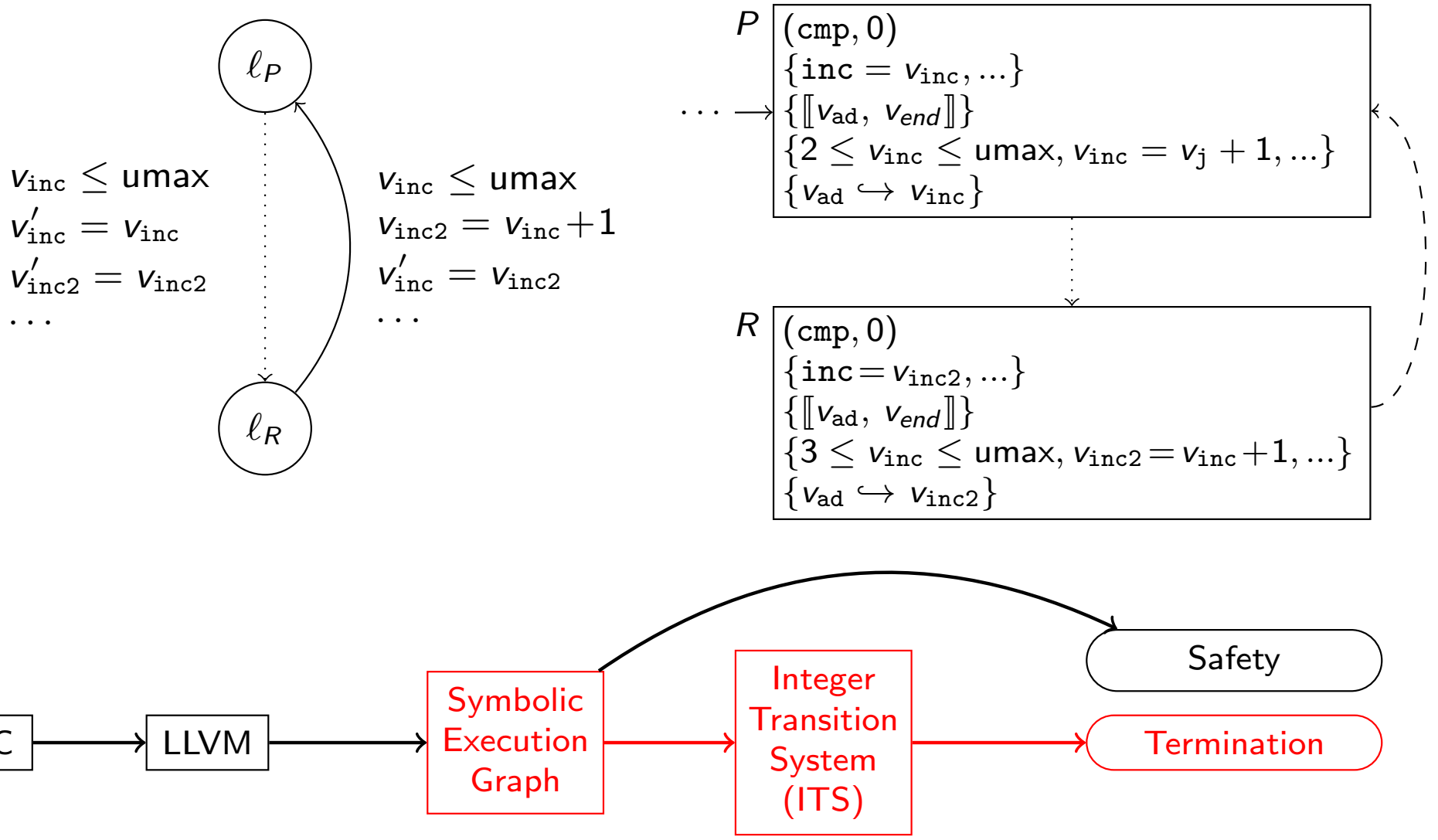






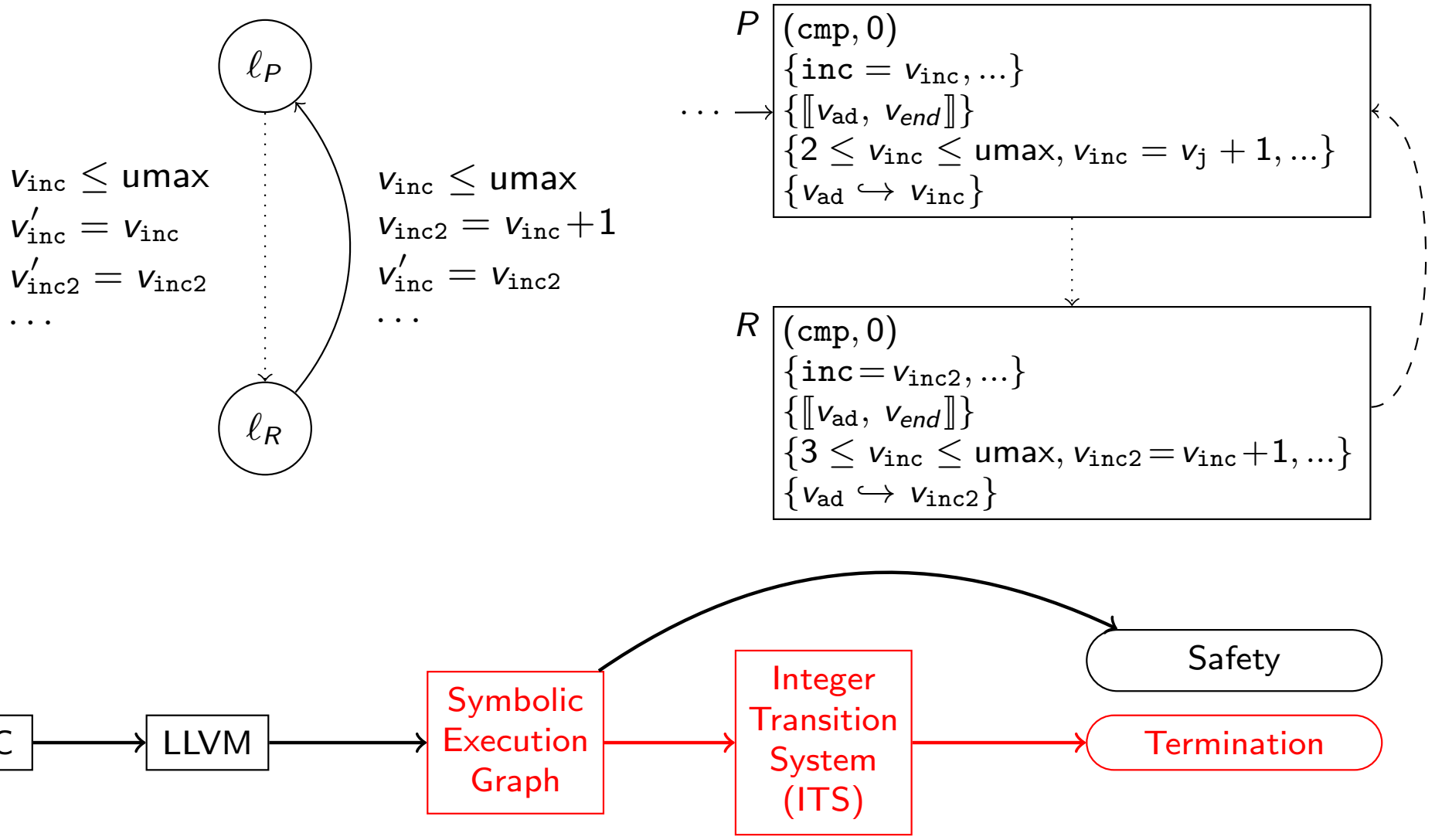
- ITS from cycles of symbolic execution graph

Termination



- ITS from cycles of symbolic execution graph
- ITS termination by existing tools

Termination



- ITS from cycles of symbolic execution graph
- ITS termination by existing tools $\Longrightarrow$ LLVM program terminates

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathrm{y} + \mathrm{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathrm{y} + \mathrm{z} - 2^n \leq \mathsf{umax}_n$

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathrm{y} + \mathrm{z} > \mathrm{umax}_n \quad \Longrightarrow \quad \mathrm{y} + \mathrm{z} - 2^n \leq \mathrm{umax}_n$

- **Multiplication:** case analysis not practical

  $\mathrm{y} * \mathrm{z} > \mathrm{umax}_n \quad \not\Longrightarrow \quad \mathrm{y} * \mathrm{z} - 2^n \leq \mathrm{umax}_n$

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathrm{y} + \mathrm{z} > \mathrm{umax}_n \quad \Longrightarrow \quad \mathrm{y} + \mathrm{z} - 2^n \leq \mathrm{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathrm{y} * \mathrm{z} > \mathrm{umax}_n \quad \not\Longrightarrow \quad \mathrm{y} * \mathrm{z} - 2^n \leq \mathrm{umax}_n$

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

Symbolic execution rule for `x = mul i32` t_1, t_2

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathsf{umax}_{32}$

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathrm{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathrm{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathrm{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathrm{umax}_n$

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathrm{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathrm{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathrm{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathrm{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathrm{umax}_n$

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathrm{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise
- extend KB by additional information on intervals of the result

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- Corresponding rules for **bitwise binary operations** (`and`, `zext`, `trunc`, ...)

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathsf{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise
- extend KB by additional information on intervals of the result

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- Corresponding rules for **bitwise binary operations** (`and`, `zext`, `trunc`, ...)
  - case analysis

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathsf{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise
- extend KB by additional information on intervals of the result

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathsf{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathsf{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathsf{umax}_n$

- Corresponding rules for **bitwise binary operations** (`and`, `zext`, `trunc`, ...)
  - case analysis
  - *modulo*

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathsf{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise
- extend KB by additional information on intervals of the result

Multiplication

- **Addition:** handle overflows by case analysis

  $\mathtt{y} + \mathtt{z} > \mathrm{umax}_n \quad \Longrightarrow \quad \mathtt{y} + \mathtt{z} - 2^n \leq \mathrm{umax}_n$

- **Multiplication:** handle overflows by *modulo*

  $\mathtt{y} * \mathtt{z} > \mathrm{umax}_n \quad \not\Longrightarrow \quad \mathtt{y} * \mathtt{z} - 2^n \leq \mathrm{umax}_n$

- Corresponding rules for **bitwise binary operations** (`and`, `zext`, `trunc`, ...)
  - case analysis
  - *modulo*
  - extend *KB* by additional information on intervals of result

Symbolic execution rule for `x = mul i32` t_1, t_2 **where** $\mathtt{x} \in \mathcal{U}$

- set `x` to $PV_u(t_1) * PV_u(t_2)$ if $\models \langle a \rangle \implies PV_u(t_1) * PV_u(t_2) \leq \mathrm{umax}_{32}$
- set `x` to $(PV_u(t_1) * PV_u(t_2)) \mod 2^{32}$ otherwise
- extend *KB* by additional information on intervals of the result

Termination of C with Bitvector Arithmetic

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$
- Heuristic to decide whether to represent information on unsigned or signed value of variables in abstract states

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$
- Heuristic to decide whether to represent information on unsigned or signed value of variables in abstract states
- Hybrid approach to handle overflows by case analysis or by *modulo*

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$
- Heuristic to decide whether to represent information on unsigned or signed value of variables in abstract states
- Hybrid approach to handle overflows by case analysis or by *modulo*
- Implementation in AProVE

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$
- Heuristic to decide whether to represent information on unsigned or signed value of variables in abstract states
- Hybrid approach to handle overflows by case analysis or by *modulo*
- Implementation in AProVE
  118 C programs from evaluations of other termination tools

Termination of C with Bitvector Arithmetic

- Symbolic execution combines handling of bitvectors with precise representation of low-level memory operations
- Representation of bitvectors by relations on $\mathbb{Z}$
  $\Longrightarrow$ standard SMT solving and termination analysis over $\mathbb{Z}$
- Heuristic to decide whether to represent information on unsigned or signed value of variables in abstract states
- Hybrid approach to handle overflows by case analysis or by *modulo*
- Implementation in AProVE
  118 C programs from evaluations of other termination tools

	T	**F**	**TO**	**RT**	**T**	**F**	**TO**	**RT**	**%**
AProVE	34	9	9	10.23	61	3	2	5.55	80.5
2LS	23	29	0	0.37	45	21	0	0.33	57.6
KITTeL	27	4	21	1.81	33	3	30	14.17	50.8
Juggernaut	10	19	23	34.12	22	26	18	6.22	27.1
Ultimate	–	–	–	–	11	54	1	12.77	16.7