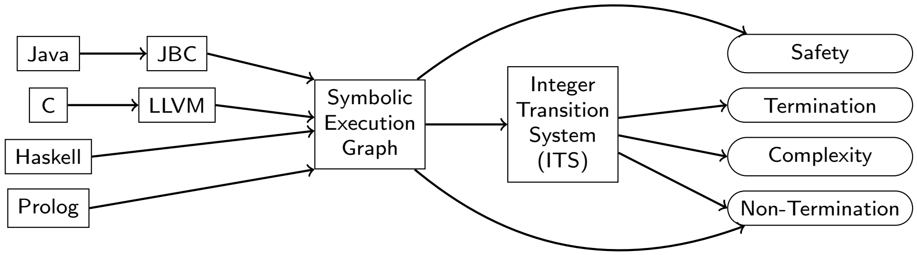
# Proving Termination of Programs with Bitvector Arithmetic by Symbolic Execution

Jürgen Giesl

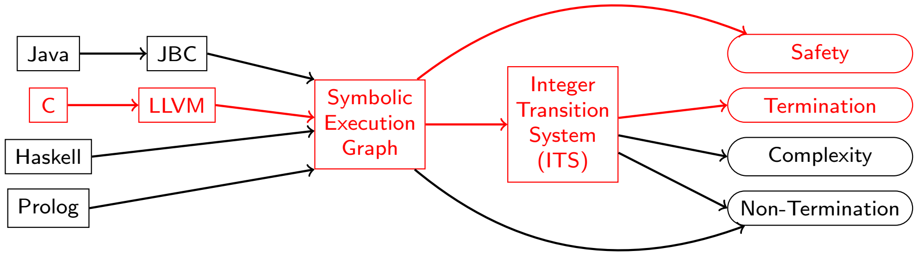
LuFG Informatik 2, RWTH Aachen University, Germany

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

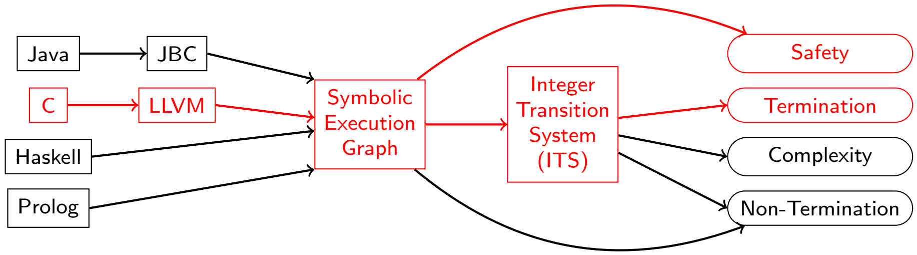
# Termination Analysis in AProVE



# Termination Analysis in AProVE

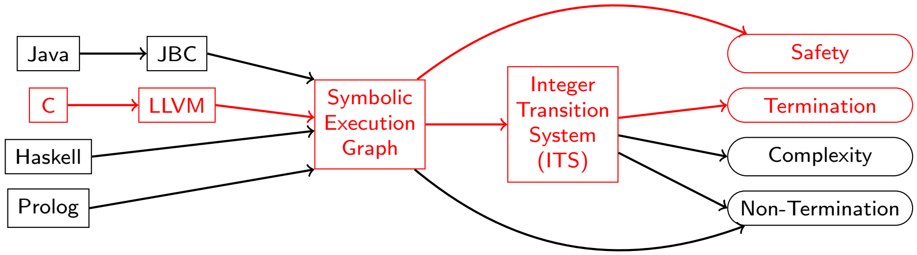


## Termination Analysis in AProVE



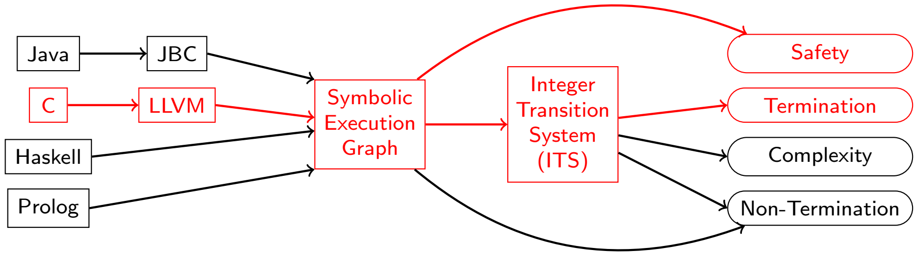
- Termination of C programs with explicit pointer arithmetic

## Termination Analysis in AProVE



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

## 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++;  }
```

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

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

|  |  |
|---|---|
| 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++;  }
```

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

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

| | |
|---|---|
| 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++;  }
```

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

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

|  |  |
|---|---|
| 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 |

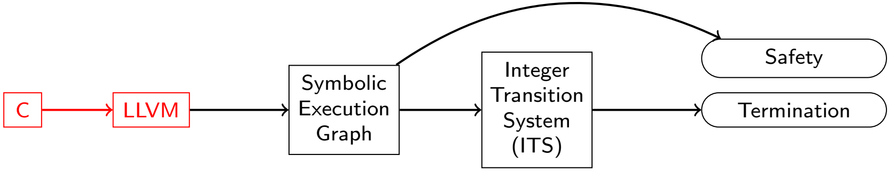
```
void g(unsigned int j)  {
    while (j > 0) j++;   }
```

|  |  |
|---|---|
| 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

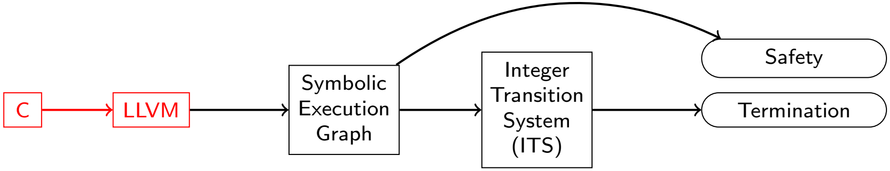
```
void g(unsigned int j)   {
    while (j > 0) j++;   }
```



## From C to LLVM

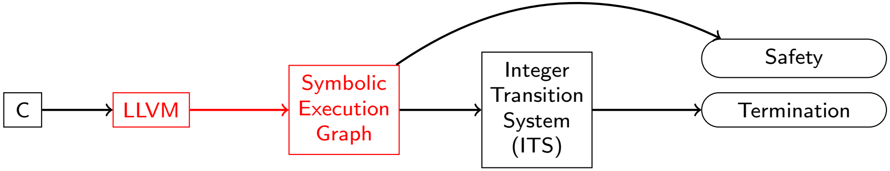
```
define i32 @g(i32 j) {
entry: 0:ad = alloca i32
       1:store i32 j, i32* ad
       2:br label cmp
cmp:   0:j1 = load i32* ad
       1:j1p = icmp ugt i32 j1, 0
       2:br i1 j1p, label body,
                    label done
body:  0:j2 = load i32* ad
       1:inc = add i32 j2, 1
       2:store i32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

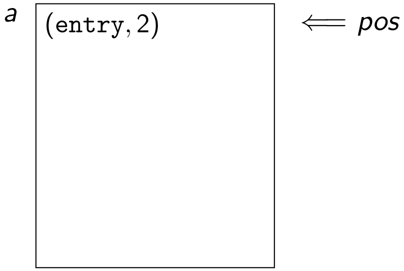
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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

$a$

$$\boxed{\begin{array}{l}(\texttt{entry}, 2) \\ \{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\} \\ \\ \\ \\ \end{array}}$$

$\Longleftarrow$ *pos*

$\Longleftarrow$ *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 }
```

$a$

$$\boxed{\begin{array}{l} (\texttt{entry}, 2) \\[1em] \{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\} \\[1em] \{[\![v_{\texttt{ad}},\ v_{end}]\!]\} \\[4em] \end{array}} \quad \begin{array}{l} \Longleftarrow pos \\[1em] \Longleftarrow PV \\[1em] \Longleftarrow AL \\[4em] \end{array}$$

**Abstract state *a*:**

- *pos*: program position (block, next instruction)
- *PV*: program variables $\to$ 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 }
```

$a$

$$\boxed{\begin{array}{l} (\mathtt{entry}, 2) \\ \{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} \\ \{[\![v_{\mathtt{ad}},\ v_{end}]\!]\} \\ \{v_{end} = v_{\mathtt{ad}} + 3\} \end{array}} \quad \begin{array}{l} \Longleftarrow\ pos \\ \Longleftarrow\ PV \\ \Longleftarrow\ AL \\ \Longleftarrow\ KB \end{array}$$

**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 }
```

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

**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 }
```

$$a \quad \boxed{\begin{array}{l} (\texttt{entry}, 2) \\[1em] \{\texttt{j} = v_{\texttt{j}}, \texttt{ad} = v_{\texttt{ad}}\} \\[1em] \{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\} \\[1em] \{v_{end} = v_{\texttt{ad}} + 3\} \\[1em] \{v_{\texttt{ad}} \hookrightarrow_{\texttt{i32},u} v_{\texttt{j}}\} \end{array}} \quad \begin{array}{l} \Longleftarrow \ pos \\[1em] \Longleftarrow \ PV \\[1em] \Longleftarrow \ AL \\[1em] \Longleftarrow \ KB \\[1em] \Longleftarrow \ PT \end{array}$$

**Abstract state *a*:** *ERR* or

- *pos*: program position (block, next instruction)
- *PV*: program variables $\to$ 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 }
```

$$a \quad \boxed{\begin{array}{l} (\mathtt{entry}, 2) \\[1em] \{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} \\[1em] \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\[1em] \{v_{end} = v_{\mathtt{ad}} + 3\} \\[1em] \{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} \end{array}} \quad \begin{array}{l} \Longleftarrow pos \\[1em] \Longleftarrow PV \\[1em] \Longleftarrow AL \\[1em] \Longleftarrow KB \\[1em] \Longleftarrow PT \end{array}$$

- $\langle a \rangle$: FO formula

## 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 }
```

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

- $\langle a \rangle$: 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 }
```

$a$

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

- $\langle a \rangle$: 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 }
```

$a$

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

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

    $\mathtt{j} \in \mathcal{U}$ has type 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 }
```

$a$

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

- $\langle a \rangle$: 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 }
```

$a$

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

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

## 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 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 }
```

$$
a \quad \boxed{\begin{array}{l} (\mathtt{entry}, 2) \\[1ex] \{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}\} \\[1ex] \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\[1ex] \{v_{end} = v_{\mathtt{ad}} + 3\} \\[1ex] \{v_{\mathtt{ad}} \hookrightarrow_{\mathtt{i32},u} v_{\mathtt{j}}\} \end{array}} \quad \begin{array}{l} \Longleftarrow\ pos \\[1ex] \Longleftarrow\ PV \\[1ex] \Longleftarrow\ AL \\[1ex] \Longleftarrow\ KB \\[1ex] \Longleftarrow\ PT \end{array}
$$

- $\langle a \rangle$: 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 }
```

$A$

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

# 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 }
```

$$A \quad \boxed{\begin{array}{l} (\texttt{entry},0) \\ \{\texttt{j} = v_{\texttt{j}},\ldots\} \\ \varnothing \\ \{0 \le v_{\texttt{j}} \le \mathsf{umax},\ldots\} \\ \varnothing \end{array}} \quad \begin{array}{l} \Longleftarrow \mathit{pos} \\ \Longleftarrow \mathit{PV} \\ \Longleftarrow \mathit{AL} \\ \Longleftarrow \mathit{KB} \\ \Longleftarrow \mathit{PT} \end{array}$$

# 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 }
```

$A$

$$\boxed{\begin{array}{l} (\mathtt{entry}, 0) \\ \{\mathtt{j} = v_{\mathtt{j}}, \ldots\} \\ \varnothing \\ \{0 \le v_{\mathtt{j}} \le \mathsf{umax}, \ldots\} \\ \varnothing \end{array}} \begin{array}{l} \Longleftarrow \mathit{pos} \\ \Longleftarrow \mathit{PV} \\ \Longleftarrow \mathit{AL} \\ \Longleftarrow \mathit{KB} \\ \Longleftarrow \mathit{PT} \end{array}$$

## 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 }
```

$$\begin{tikzcd}
A\ \boxed{\begin{array}{l} (\texttt{entry}, 0) \\ \{\mathrm{j} = v_{\mathrm{j}}, \dots\} \\ \varnothing \\ \{0 \le v_{\mathrm{j}} \le \mathrm{umax}, \dots\} \\ \varnothing \end{array}} \arrow[r] & B\ \boxed{\begin{array}{l} (\texttt{entry}, 1) \\ \{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}}, \dots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{v_{end} = v_{\mathrm{ad}} + 3, \dots\} \\ \varnothing \end{array}}
\end{tikzcd}$$

## 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 }
```

$$\begin{tikzcd} A\ \boxed{\begin{array}{l} (\texttt{entry}, 0)\\ \{\mathsf{j} = v_{\mathsf{j}}, \ldots\}\\ \varnothing\\ \{0 \leq v_{\mathsf{j}} \leq \mathsf{umax}, \ldots\}\\ \varnothing \end{array}} \arrow[r] & B\ \boxed{\begin{array}{l} (\texttt{entry}, 1)\\ \{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \ldots\}\\ \{[\![v_{\mathsf{ad}},\ v_{end}]\!]\}\\ \{v_{end} = v_{\mathsf{ad}} + 3, \ldots\}\\ \varnothing \end{array}} \end{tikzcd}$$

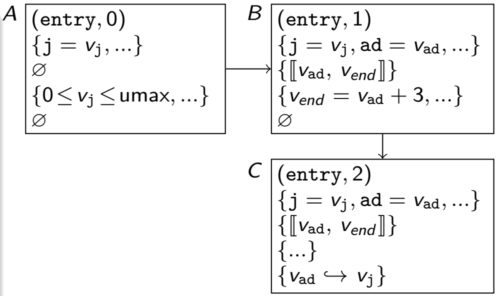
# Symbolic Execution

```
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 }
```

$$\begin{tikzcd}
A\ \boxed{\begin{array}{l} (\mathtt{entry}, 0) \\ \{\mathsf{j} = v_{\mathsf{j}}, \ldots\} \\ \varnothing \\ \{0 \leq v_{\mathsf{j}} \leq \mathsf{umax}, \ldots\} \\ \varnothing \end{array}} \arrow[r] & B\ \boxed{\begin{array}{l} (\mathtt{entry}, 1) \\ \{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \ldots\} \\ \{[\![v_{\mathsf{ad}},\ v_{end}]\!]\} \\ \{v_{end} = v_{\mathsf{ad}} + 3, \ldots\} \\ \varnothing \end{array}} \arrow[d] \\
 & C\ \boxed{\begin{array}{l} (\mathtt{entry}, 2) \\ \{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \ldots\} \\ \{[\![v_{\mathsf{ad}},\ v_{end}]\!]\} \\ \{\ldots\} \\ \{v_{\mathsf{ad}} \hookrightarrow v_{\mathsf{j}}\} \end{array}}
\end{tikzcd}$$

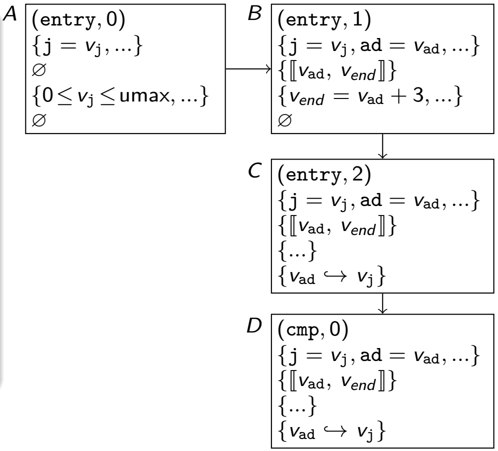
## Symbolic Execution

```
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
(entry, 0)
$\{j = v_j, \ldots\}$
$\varnothing$
$\{0 \leq v_j \leq \text{umax}, \ldots\}$
$\varnothing$
B
(entry, 1)
$\{j = v_j, ad = v_{ad}, \ldots\}$
$\{[\![v_{ad},\ v_{end}]\!]\}$
$\{v_{end} = v_{ad} + 3, \ldots\}$
$\varnothing$
C
(entry, 2)
$\{j = v_j, ad = v_{ad}, \ldots\}$
$\{[\![v_{ad},\ v_{end}]\!]\}$
$\{\ldots\}$
$\{v_{ad} \hookrightarrow v_j\}$

## Symbolic Execution

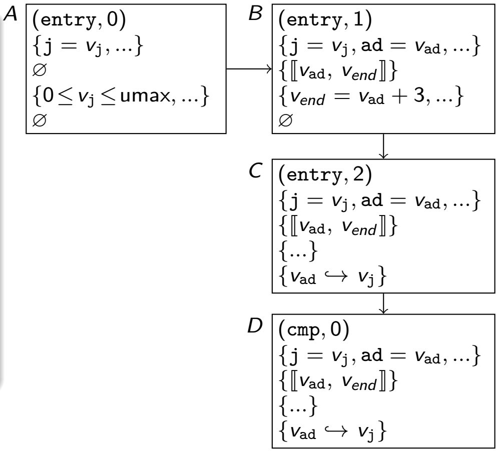
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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



$A$
$(\texttt{entry}, 0)$
$\{\mathsf{j} = v_{\mathsf{j}}, \dots\}$
$\varnothing$
$\{0 \le v_{\mathsf{j}} \le \mathsf{umax}, \dots\}$
$\varnothing$
$B$
$(\texttt{entry}, 1)$
$\{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \dots\}$
$\{[\![v_{\mathsf{ad}},\ v_{end}]\!]\}$
$\{v_{end} = v_{\mathsf{ad}} + 3, \dots\}$
$\varnothing$
$C$
$(\texttt{entry}, 2)$
$\{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \dots\}$
$\{[\![v_{\mathsf{ad}},\ v_{end}]\!]\}$
$\{\dots\}$
$\{v_{\mathsf{ad}} \hookrightarrow v_{\mathsf{j}}\}$
$D$
$(\texttt{cmp}, 0)$
$\{\mathsf{j} = v_{\mathsf{j}}, \mathsf{ad} = v_{\mathsf{ad}}, \dots\}$
$\{[\![v_{\mathsf{ad}},\ v_{end}]\!]\}$
$\{\dots\}$
$\{v_{\mathsf{ad}} \hookrightarrow v_{\mathsf{j}}\}$

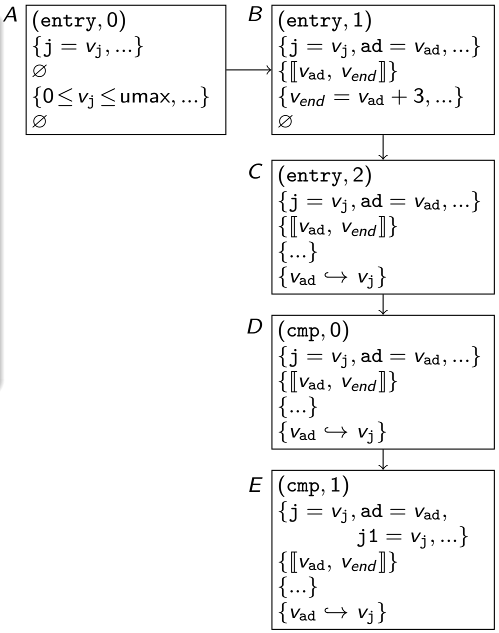
# Symbolic Execution

```
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 }
```



*A*
(entry, 0)
$\{\mathrm{j} = v_{\mathrm{j}}, \ldots\}$
$\emptyset$
$\{0 \le v_{\mathrm{j}} \le \mathrm{umax}, \ldots\}$
$\emptyset$

*B*
(entry, 1)
$\{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}}, \ldots\}$
$\{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}$
$\{v_{end} = v_{\mathrm{ad}} + 3, \ldots\}$
$\emptyset$

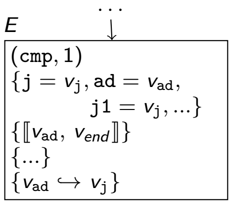
*C*
(entry, 2)
$\{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}}, \ldots\}$
$\{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}$
$\{\ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

*D*
(cmp, 0)
$\{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}}, \ldots\}$
$\{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}$
$\{\ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

*E*
(cmp, 1)
$\{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}},$
$\mathrm{j1} = v_{\mathrm{j}}, \ldots\}$
$\{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}$
$\{\ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

# 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 }
```



$\dots$
$E$
$(\mathrm{cmp}, 1)$
$\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}},$
$\mathtt{j1} = v_{\mathtt{j}}, \dots\}$
$\{[\![ v_{\mathtt{ad}}, v_{end} ]\!]\}$
$\{\dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

## 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 }
```

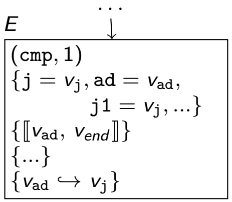
$E$ $\quad\ldots\downarrow$

$$\boxed{\begin{array}{l}(\mathrm{cmp}, 1)\\ \{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}},\\ \qquad\quad \mathtt{j1} = v_{\mathtt{j}}, \ldots\}\\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}\\ \{\ldots\}\\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}\end{array}}$$

**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 }
```



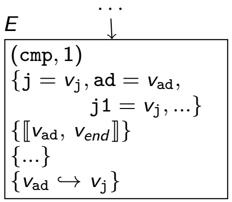
$E$
$\dots$
$(\mathrm{cmp}, 1)$
$\{\mathrm{j} = v_{\mathrm{j}}, \mathrm{ad} = v_{\mathrm{ad}},$
$\mathrm{j1} = v_{\mathrm{j}}, \dots\}$
$\{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}$
$\{\dots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

**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 }
```



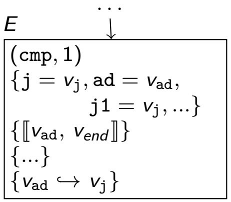
E
. . .
(cmp, 1)
$\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}},$
$\mathtt{j1} = v_{\mathtt{j}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{\ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

**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 }
```



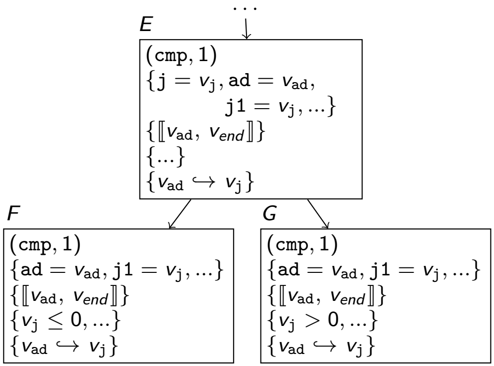
$(\mathrm{cmp}, 1)$
$\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{\dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

**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 }
```

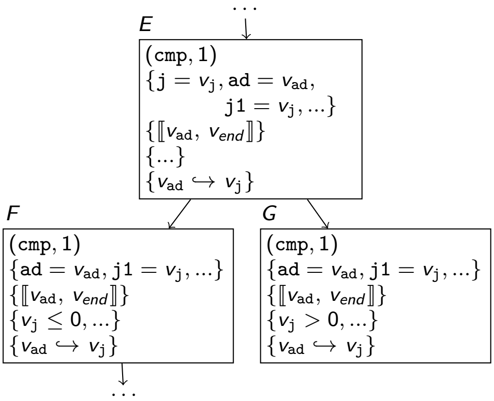


**Symbolic execution rule for `x = icmp ugt i32` $t_1$, $t_2$**

- 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 }
```



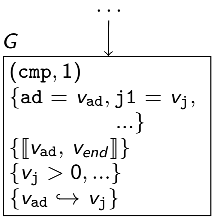
$\dots$
$E$
$(\mathtt{cmp}, 1)$
$\{\mathtt{j} = v_{\mathtt{j}}, \mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{\dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
$F$
$(\mathtt{cmp}, 1)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} \le 0, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
$\dots$
$G$
$(\mathtt{cmp}, 1)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

**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 $\models \langle a \rangle \implies (PV_u(t_1) \le PV_u(t_2))$
- otherwise: case analysis

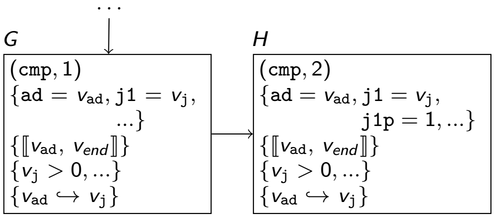
## Symbolic Execution

```
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 }
```


$$\begin{tikzcd} \dots \arrow[d] \\ \boxed{\begin{array}{l} (\mathtt{cmp}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \\ \qquad \dots\} \\ \{[\![v_{\mathtt{ad}}, v_{end}]\!]\} \\ \{v_{\mathtt{j}} > 0, \dots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}} \end{tikzcd}$$

## 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 }
```



$\cdots$
$G$
$(\mathtt{cmp}, 1)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
$H$
$(\mathtt{cmp}, 2)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \mathtt{j1p} = 1, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

## 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 }
```

$$\begin{tikzcd}
G & \cdots \arrow[d] & H \\
& \boxed{\begin{array}{l} (\mathrm{cmp}, 1) \\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j1} = v_{\mathrm{j}}, \\ \qquad \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathrm{j}} > 0, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}} \arrow[r] & \boxed{\begin{array}{l} (\mathrm{cmp}, 2) \\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j1} = v_{\mathrm{j}}, \\ \qquad \mathrm{j1p} = 1, \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathrm{j}} > 0, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}}
\end{tikzcd}$$

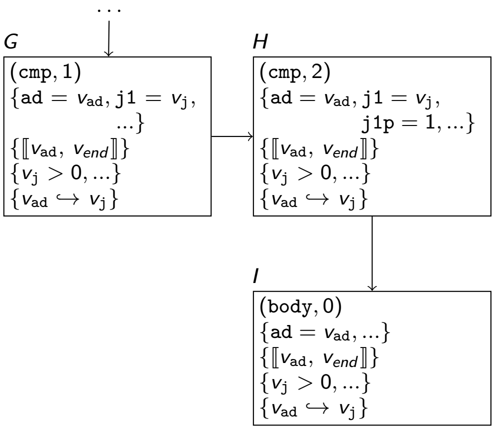
# Symbolic Execution

```
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 }
```

$$\begin{tikzcd}
\cdots \arrow[d] & \\
G\ \boxed{\begin{array}{l} (\mathtt{cmp}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \\ \qquad \ldots\} \\ \{[\![v_{\mathtt{ad}}, v_{end}]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}} \arrow[r] & H\ \boxed{\begin{array}{l} (\mathtt{cmp}, 2) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \\ \qquad \mathtt{j1p} = 1, \ldots\} \\ \{[\![v_{\mathtt{ad}}, v_{end}]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}} \arrow[d] \\
 & I\ \boxed{\begin{array}{l} (\mathtt{body}, 0) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \ldots\} \\ \{[\![v_{\mathtt{ad}}, v_{end}]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}}
\end{tikzcd}$$

## Symbolic Execution

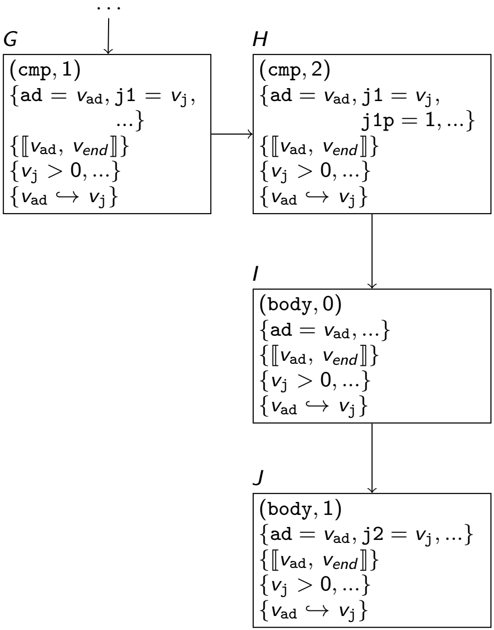
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                      label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
      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



...
G
$(\mathrm{cmp}, 1)$
$\{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j1} = v_{\mathrm{j}}, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} > 0, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$
H
$(\mathrm{cmp}, 2)$
$\{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j1} = v_{\mathrm{j}}, \mathrm{j1p} = 1, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} > 0, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$
I
$(\mathrm{body}, 0)$
$\{\mathrm{ad} = v_{\mathrm{ad}}, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} > 0, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

## 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 }
```



...
G
$(\mathtt{cmp}, 1)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
H
$(\mathtt{cmp}, 2)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j1} = v_{\mathtt{j}}, \mathtt{j1p} = 1, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
I
$(\mathtt{body}, 0)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$
J
$(\mathtt{body}, 1)$
$\{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j2} = v_{\mathtt{j}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{v_{\mathtt{j}} > 0, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

## 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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & J\ \boxed{\begin{array}{l} (\mathtt{body}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j2} = v_{\mathtt{j}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}}
\end{tikzcd}$$

## 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 }
```

$$\cdots \longrightarrow J\ \boxed{\begin{array}{l} (\mathtt{body}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j2} = v_{\mathtt{j}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}}$$

### 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 }
```

$$\cdots \longrightarrow J\ \boxed{\begin{array}{l} (\mathtt{body}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j2} = v_{\mathtt{j}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}}$$

**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 }
```

$$\cdots \longrightarrow J\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j2} = v_{\mathrm{j}}, \ldots\} \\ \{[\![v_{\mathrm{ad}},\ v_{end}]\!]\} \\ \{v_{\mathrm{j}} > 0, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}}$$

**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 $\models \langle a \rangle \implies PV_u(t_1) + PV_u(t_2) \leq \mathrm{umax}_{32}$

## 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 }
```

$$\cdots \longrightarrow J\ \boxed{\begin{array}{l} (\mathtt{body}, 1) \\ \{\mathtt{ad} = v_{\mathtt{ad}}, \mathtt{j2} = v_{\mathtt{j}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathtt{j}} > 0, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\} \end{array}}$$

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

# 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 }
```

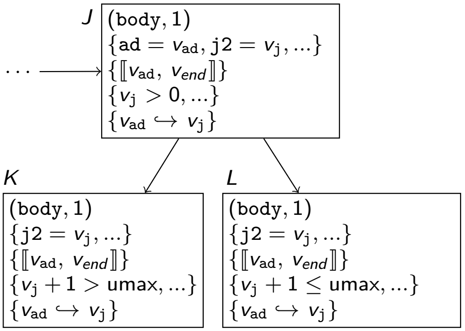
$$\begin{tikzcd} \cdots \arrow[r] & J\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j2} = v_{\mathrm{j}}, \ldots\} \\ \{[\![v_{\mathrm{ad}},\ v_{end}]\!]\} \\ \{v_{\mathrm{j}} > 0, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}} \end{tikzcd}$$

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



$$\begin{tikzcd} \cdots \arrow[r] & J \arrow[dl] \arrow[dr] & \\ K & & L \end{tikzcd}$$

$J$:
$(\mathrm{body}, 1)$
$\{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j2} = v_{\mathrm{j}}, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} > 0, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

$K$:
$(\mathrm{body}, 1)$
$\{\mathrm{j2} = v_{\mathrm{j}}, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} + 1 > \mathrm{umax}, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

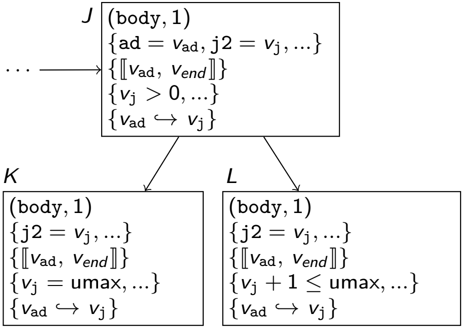
$L$:
$(\mathrm{body}, 1)$
$\{\mathrm{j2} = v_{\mathrm{j}}, \ldots\}$
$\{[\![v_{\mathrm{ad}}, v_{end}]\!]\}$
$\{v_{\mathrm{j}} + 1 \leq \mathrm{umax}, \ldots\}$
$\{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}$

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



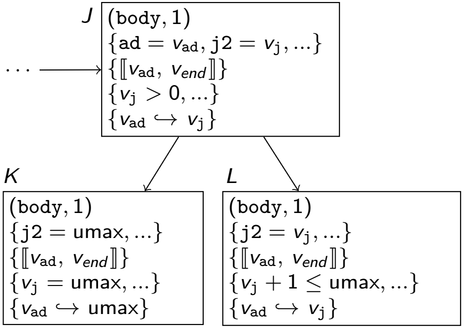
$$\begin{tikzcd}[row sep=large] \cdots \arrow[r] & J: \boxed{\begin{array}{l}(\mathrm{body},1)\\ \{\mathrm{ad}=v_{\mathrm{ad}},\mathrm{j2}=v_{\mathrm{j}},\dots\}\\ \{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}\\ \{v_{\mathrm{j}}>0,\dots\}\\ \{v_{\mathrm{ad}}\hookrightarrow v_{\mathrm{j}}\}\end{array}} \arrow[dl] \arrow[dr] & \\ K: \boxed{\begin{array}{l}(\mathrm{body},1)\\ \{\mathrm{j2}=v_{\mathrm{j}},\dots\}\\ \{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}\\ \{v_{\mathrm{j}}=\mathrm{umax},\dots\}\\ \{v_{\mathrm{ad}}\hookrightarrow v_{\mathrm{j}}\}\end{array}} & & L: \boxed{\begin{array}{l}(\mathrm{body},1)\\ \{\mathrm{j2}=v_{\mathrm{j}},\dots\}\\ \{[\![v_{\mathrm{ad}},\ v_{end}]\!]\}\\ \{v_{\mathrm{j}}+1\le\mathrm{umax},\dots\}\\ \{v_{\mathrm{ad}}\hookrightarrow v_{\mathrm{j}}\}\end{array}} \end{tikzcd}$$

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



$\cdots$
$J$
$(\text{body}, 1)$
$\{\text{ad} = v_{\text{ad}}, \text{j2} = v_{\text{j}}, \ldots\}$
$\{[\![v_{\text{ad}},\ v_{end}]\!]\}$
$\{v_{\text{j}} > 0, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow v_{\text{j}}\}$
$K$
$(\text{body}, 1)$
$\{\text{j2} = \text{umax}, \ldots\}$
$\{[\![v_{\text{ad}},\ v_{end}]\!]\}$
$\{v_{\text{j}} = \text{umax}, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow \text{umax}\}$
$L$
$(\text{body}, 1)$
$\{\text{j2} = v_{\text{j}}, \ldots\}$
$\{[\![v_{\text{ad}},\ v_{end}]\!]\}$
$\{v_{\text{j}} + 1 \leq \text{umax}, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow v_{\text{j}}\}$

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

- 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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & J\ \boxed{\begin{array}{l}(\mathrm{body}, 1)\\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j2} = v_{\mathrm{j}}, \ldots\}\\ \{[\![ v_{\mathrm{ad}}, v_{end} ]\!]\}\\ \{v_{\mathrm{j}} > 0, \ldots\}\\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}\end{array}} \arrow[dl] \arrow[dr] & \\
K\ \boxed{\begin{array}{l}(\mathrm{body}, 1)\\ \{\mathrm{j2} = \mathrm{umax}, \ldots\}\\ \{[\![ v_{\mathrm{ad}}, v_{end} ]\!]\}\\ \{v_{\mathrm{j}} = \mathrm{umax}, \ldots\}\\ \{v_{\mathrm{ad}} \hookrightarrow \mathrm{umax}\}\end{array}} & & L\ \boxed{\begin{array}{l}(\mathrm{body}, 1)\\ \{\mathrm{j2} = v_{\mathrm{j}}, \ldots\}\\ \{[\![ v_{\mathrm{ad}}, v_{end} ]\!]\}\\ \{1 \leq v_{\mathrm{j}} \leq \mathrm{umax} - 1, \ldots\}\\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\}\end{array}}
\end{tikzcd}$$

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

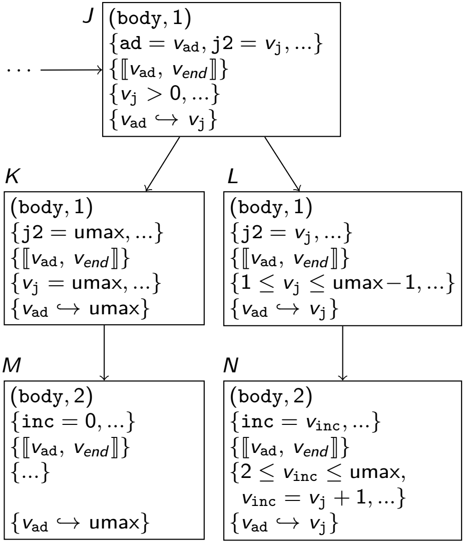
# Addition

```
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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & J\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathrm{ad} = v_{\mathrm{ad}}, \mathrm{j2} = v_{\mathrm{j}}, \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathrm{j}} > 0, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}} \arrow[dl] \arrow[d] \\
K\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathrm{j2} = \mathrm{umax}, \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{v_{\mathrm{j}} = \mathrm{umax}, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow \mathrm{umax}\} \end{array}} \arrow[d] & L\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathrm{j2} = v_{\mathrm{j}}, \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{1 \leq v_{\mathrm{j}} \leq \mathrm{umax} - 1, \ldots\} \\ \{v_{\mathrm{ad}} \hookrightarrow v_{\mathrm{j}}\} \end{array}} \\
M\ \boxed{\begin{array}{l} (\mathrm{body}, 2) \\ \{\mathrm{inc} = 0, \ldots\} \\ \{[\![ v_{\mathrm{ad}},\ v_{end} ]\!]\} \\ \{\ldots\} \\ \\ \{v_{\mathrm{ad}} \hookrightarrow \mathrm{umax}\} \end{array}} &
\end{tikzcd}$$

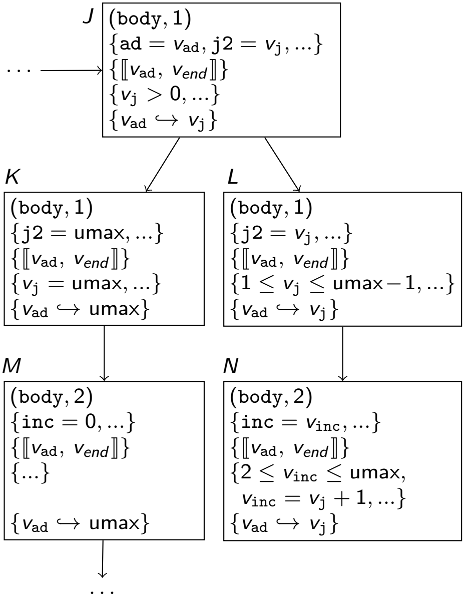
## Addition

```
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

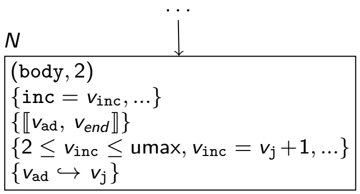
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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



$J$
$(\text{body}, 1)$
$\{\text{ad} = v_{\text{ad}}, \text{j2} = v_{\text{j}}, \ldots\}$
$\{[\![ v_{\text{ad}},\ v_{end} ]\!]\}$
$\{v_{\text{j}} > 0, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow v_{\text{j}}\}$
$\ldots$
$K$
$(\text{body}, 1)$
$\{\text{j2} = \text{umax}, \ldots\}$
$\{[\![ v_{\text{ad}},\ v_{end} ]\!]\}$
$\{v_{\text{j}} = \text{umax}, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow \text{umax}\}$
$L$
$(\text{body}, 1)$
$\{\text{j2} = v_{\text{j}}, \ldots\}$
$\{[\![ v_{\text{ad}},\ v_{end} ]\!]\}$
$\{1 \leq v_{\text{j}} \leq \text{umax} - 1, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow v_{\text{j}}\}$
$M$
$(\text{body}, 2)$
$\{\text{inc} = 0, \ldots\}$
$\{[\![ v_{\text{ad}},\ v_{end} ]\!]\}$
$\{\ldots\}$
$\{v_{\text{ad}} \hookrightarrow \text{umax}\}$
$N$
$(\text{body}, 2)$
$\{\text{inc} = v_{\text{inc}}, \ldots\}$
$\{[\![ v_{\text{ad}},\ v_{end} ]\!]\}$
$\{2 \leq v_{\text{inc}} \leq \text{umax},$
$v_{\text{inc}} = v_{\text{j}} + 1, \ldots\}$
$\{v_{\text{ad}} \hookrightarrow v_{\text{j}}\}$
$\ldots$

# 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 }
```



$\dots$
$N$
$(\mathrm{body}, 2)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{j}}\}$

# Symbolic Execution

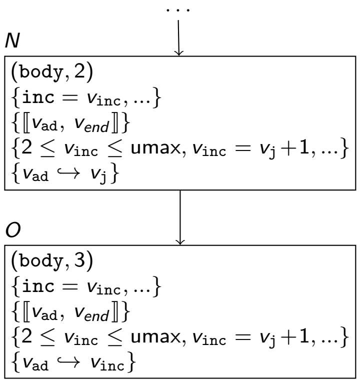
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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

$$\begin{tikzcd}
\dots \arrow[d] \\
\boxed{\begin{array}{l} (\text{body}, 2) \\ \{\texttt{inc} = v_{\texttt{inc}}, \dots\} \\ \{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \dots\} \\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{j}}\} \end{array}} \arrow[d] \\
\boxed{\begin{array}{l} (\text{body}, 3) \\ \{\texttt{inc} = v_{\texttt{inc}}, \dots\} \\ \{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \dots\} \\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\} \end{array}}
\end{tikzcd}$$

$N$ labels the first box; $O$ labels the second box.

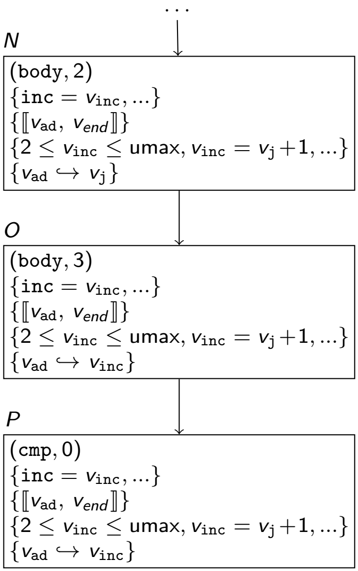
## Symbolic Execution

```
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 }
```


$$\begin{tikzcd}
\dots \arrow[d] \\
N: \boxed{\begin{array}{l}(\texttt{body}, 2)\\ \{\texttt{inc} = v_{\texttt{inc}}, \dots\}\\ \{[\![v_{\texttt{ad}},\ v_{end}]\!]\}\\ \{2 \le v_{\texttt{inc}} \le \mathsf{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \dots\}\\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{j}}\}\end{array}} \arrow[d] \\
O: \boxed{\begin{array}{l}(\texttt{body}, 3)\\ \{\texttt{inc} = v_{\texttt{inc}}, \dots\}\\ \{[\![v_{\texttt{ad}},\ v_{end}]\!]\}\\ \{2 \le v_{\texttt{inc}} \le \mathsf{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \dots\}\\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}\end{array}}
\end{tikzcd}$$

## 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 }
```



. . .
$N$
$(\text{body}, 2)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{j}}\}$
$O$
$(\text{body}, 3)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$P$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

## 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 }
```

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

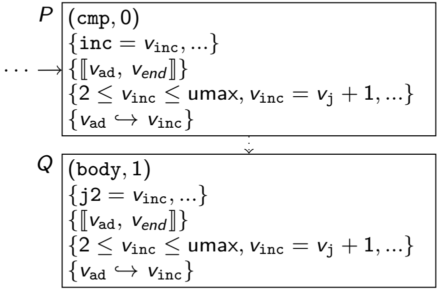
## Generalization

```
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 }
```

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

## 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 }
```



$$\begin{tikzcd} \cdots \arrow[r] & P \arrow[d, dotted] \\ & Q \end{tikzcd}$$

$P$

$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

$Q$

$(\mathtt{body}, 1)$
$\{\mathtt{j2} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

## 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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & P \; \boxed{\begin{array}{l} (\mathrm{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}} \arrow[d, dashed] \\
 & Q \; \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathtt{j2} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}}
\end{tikzcd}$$

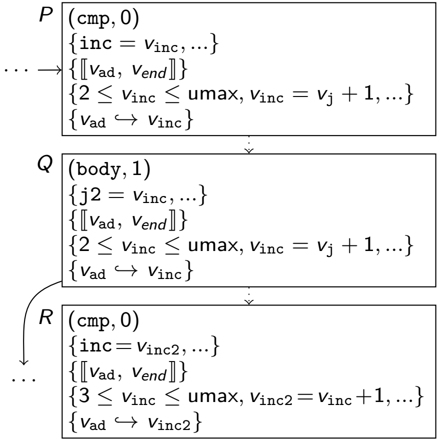
## Generalization

```
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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & P\ \boxed{\begin{array}{l} (\mathtt{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}} \arrow[d, dotted] \\
& Q\ \boxed{\begin{array}{l} (\mathtt{body}, 1) \\ \{\mathtt{j2} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}}
\end{tikzcd}$$

## 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 }
```



$P$

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

$Q$

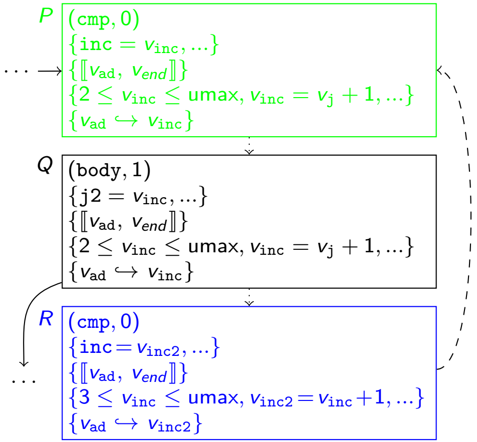
$$\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathtt{j2} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}$$

$R$

$$\begin{array}{l} (\mathrm{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{3 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\} \end{array}$$

## 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 }
```

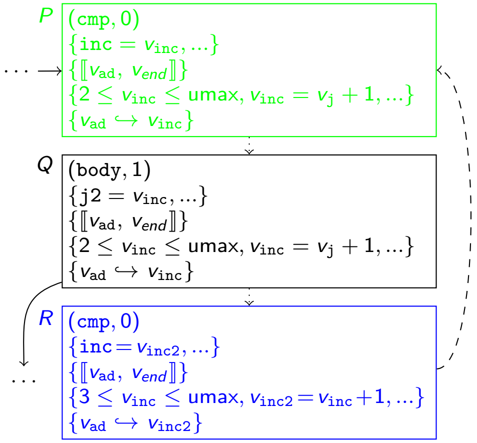


$P$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$Q$
$(\text{body}, 1)$
$\{\texttt{j2} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$R$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$

**$P$ is *generalization* of $R$ with $\mu$**

# 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 }
```



$P$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

$Q$
$(\text{body}, 1)$
$\{\texttt{j2} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

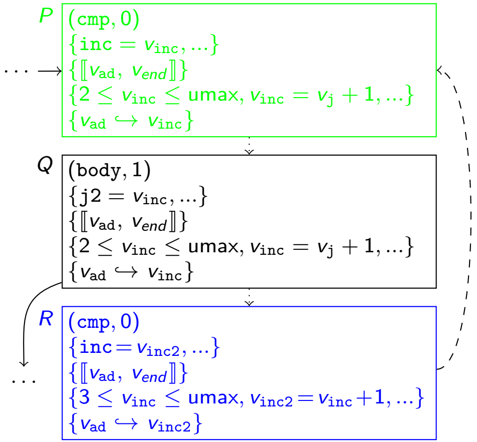
$R$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$

**$P$ is *generalization* of $R$ with $\mu$**

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

## 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 }
```



$P$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\}$
$\{2 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

$Q$
$(\text{body}, 1)$
$\{\texttt{j2} = v_{\texttt{inc}}, \ldots\}$
$\{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\}$
$\{2 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

$R$
$(\text{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![ v_{\texttt{ad}},\ v_{end} ]\!]\}$
$\{3 \le v_{\texttt{inc}} \le \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$

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

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

# 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 }
```

$$\begin{tikzcd}
\cdots \arrow[r] & P\ \boxed{\begin{array}{l} (\mathrm{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}} \arrow[d, dotted] \\
 & Q\ \boxed{\begin{array}{l} (\mathrm{body}, 1) \\ \{\mathtt{j2} = v_{\mathtt{inc}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{2 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\} \end{array}} \arrow[d, dotted] \arrow[dl] \\
\cdots & R\ \boxed{\begin{array}{l} (\mathrm{cmp}, 0) \\ \{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\} \\ \{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\} \\ \{3 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\} \\ \{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\} \end{array}} \arrow[uu, dashed, bend right]
\end{tikzcd}$$

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

- $\mu(PV_P(\mathtt{x})) = PV_R(\mathtt{x})$ for all program variables $\mathtt{x}$
- $[\![ v_1,\ v_2 ]\!] \in AL_P$ 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 }
```

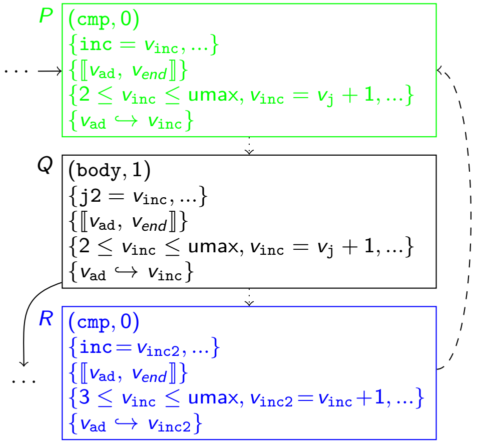
$$\begin{tikzcd}
\cdots \arrow[r] & P\ \boxed{\begin{array}{l} (\text{cmp}, 0)\\ \{\texttt{inc} = v_{\texttt{inc}}, \ldots\}\\ \{[\![v_{\texttt{ad}},\ v_{end}]\!]\}\\ \{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}\\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}\end{array}} \arrow[d, dotted] \\
 & Q\ \boxed{\begin{array}{l} (\text{body}, 1)\\ \{\texttt{j2} = v_{\texttt{inc}}, \ldots\}\\ \{[\![v_{\texttt{ad}},\ v_{end}]\!]\}\\ \{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}\\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}\end{array}} \arrow[d, dotted] \arrow[dl, bend right] \\
\cdots & R\ \boxed{\begin{array}{l} (\text{cmp}, 0)\\ \{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}\\ \{[\![v_{\texttt{ad}},\ v_{end}]\!]\}\\ \{3 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}\\ \{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}\end{array}} \arrow[uu, dashed, bend right]
\end{tikzcd}$$

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

- $\mu(PV_P(\texttt{x})) = PV_R(\texttt{x})$ for all program variables $\texttt{x}$
- $[\![v_1,\ v_2]\!] \in AL_P$ 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 }
```



$P$
$(\mathrm{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

$Q$
$(\mathrm{body}, 1)$
$\{\mathtt{j2} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

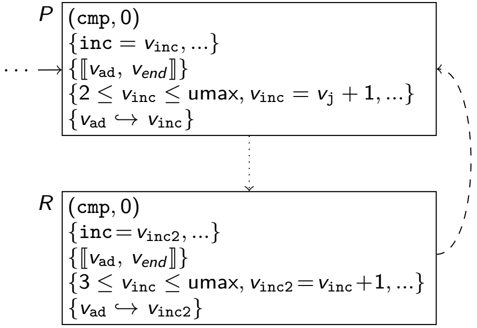
$R$
$(\mathrm{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathrm{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$

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

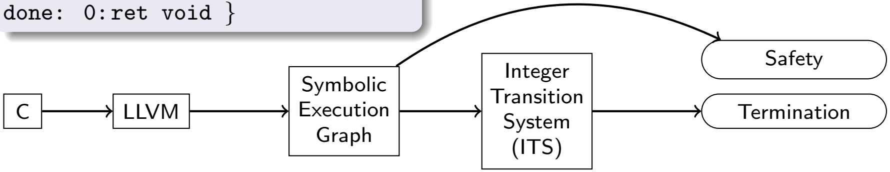
- $\mu(PV_P(\mathtt{x})) = PV_R(\mathtt{x})$ for all program variables $\mathtt{x}$
- $[\![v_1,\ v_2]\!] \in AL_P$ implies $[\![\mu(v_1),\ \mu(v_2)]\!] \in AL_R$
- $\models \langle R \rangle \implies \mu(KB_P)$
- $v_1 \hookrightarrow v_2 \in PT_P$ implies $\mu(v_1) \hookrightarrow \mu(v_2) \in PT_R$

# 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 }
```

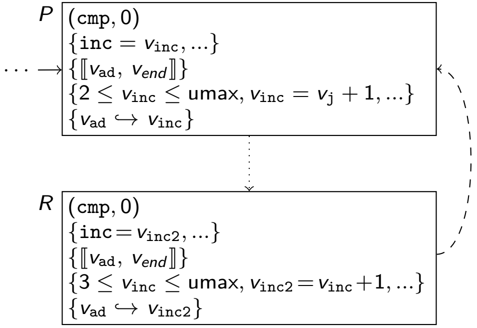


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \dots\}$
$\{[\![v_{\mathtt{ad}}, v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



## Safety

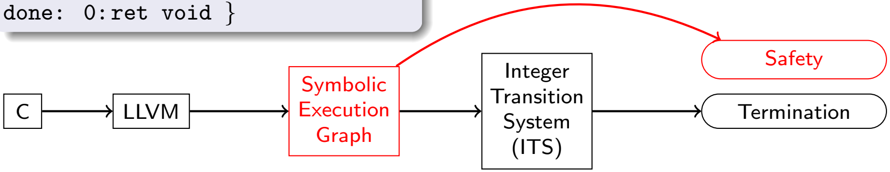
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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


$P$

$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

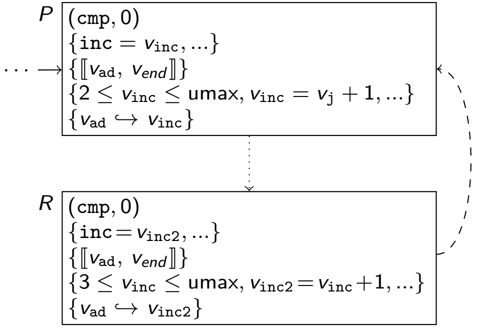
$R$

$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



# 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 }
```

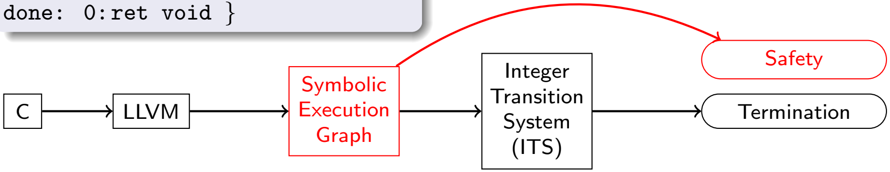


$P$

$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

$R$

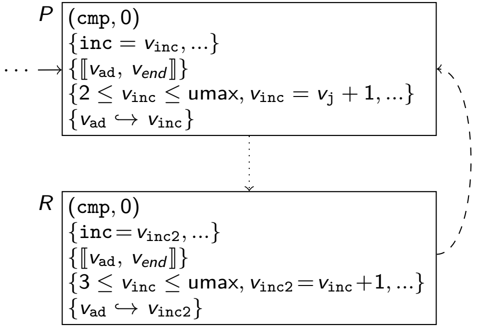
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



- 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 }
```



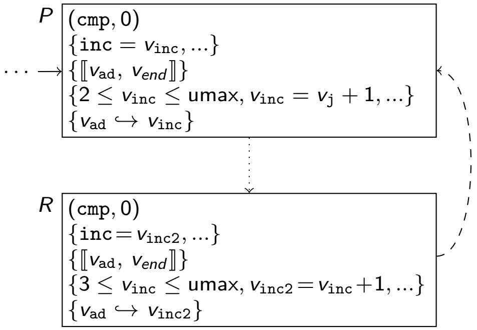
$P$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$R$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$

$$\begin{tikzcd}
 & & & & \boxed{\text{Safety}} \\
\boxed{\text{C}} \arrow[r] & \boxed{\text{LLVM}} \arrow[r] & \boxed{\text{Symbolic Execution Graph}} \arrow[r] \arrow[urr, bend left] & \boxed{\text{Integer Transition System (ITS)}} \arrow[r] & \boxed{\text{Termination}}
\end{tikzcd}$$

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

## Termination

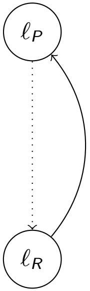
```
define i32 @g(i32 j) {
entry: 0: ad = alloca i32
       1: store i32 j, i32* ad
       2: br label cmp
cmp:   0: j1 = load i32* ad
       1: j1p = icmp ugt i32 j1, 0
       2: br i1 j1p, label body,
                     label done
body:  0: j2 = load i32* ad
       1: inc = add i32 j2, 1
       2: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

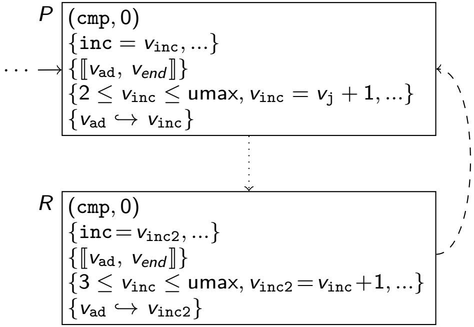


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$

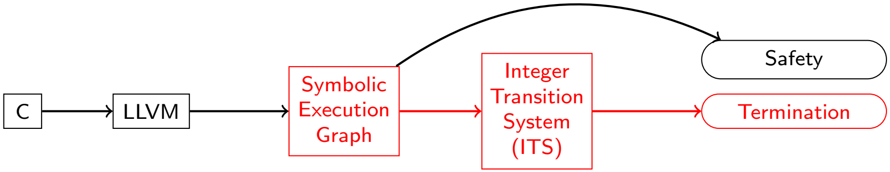
$$\begin{tikzcd}
 & & & & \text{Safety} \\
\text{C} \arrow[r] & \text{LLVM} \arrow[r] & \text{Symbolic Execution Graph} \arrow[r] \arrow[urr, bend left] & \text{Integer Transition System (ITS)} \arrow[r] & \text{Termination}
\end{tikzcd}$$

## Termination

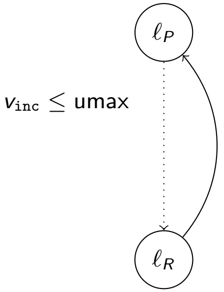


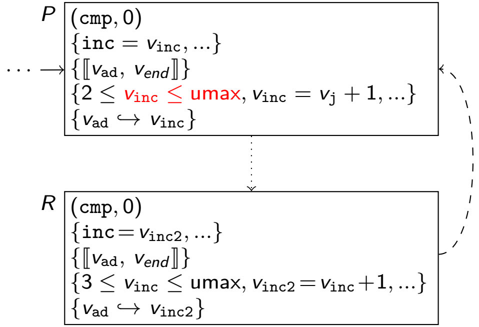


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![ v_{\mathtt{ad}},\ v_{end} ]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



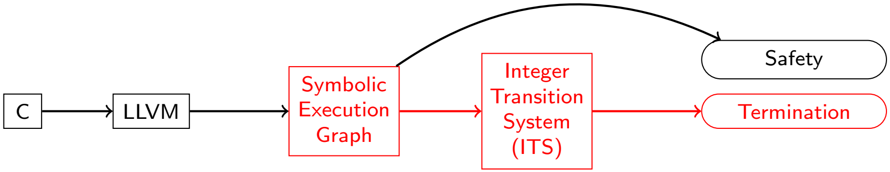
# Termination



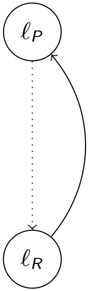


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \mathsf{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$

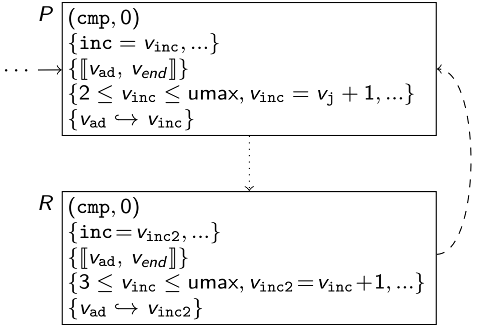
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \mathsf{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$



# Termination



$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\ldots$

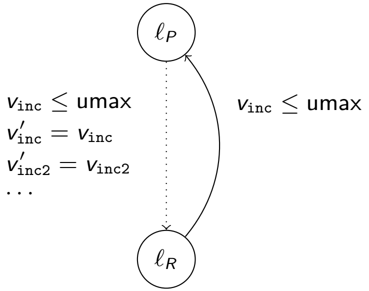


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

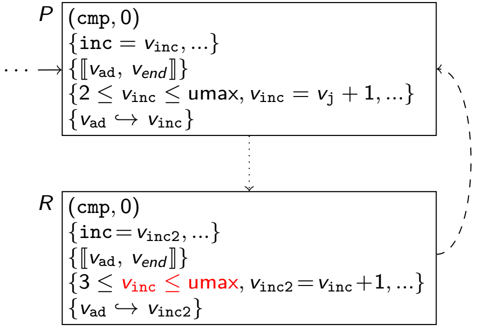
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$

$$\begin{tikzcd}
\text{C} \arrow[r] & \text{LLVM} \arrow[r] & \text{Symbolic Execution Graph} \arrow[r] \arrow[rr, bend left] & \text{Integer Transition System (ITS)} \arrow[r] & \text{Termination} \\
& & & & \text{Safety}
\end{tikzcd}$$

# Termination

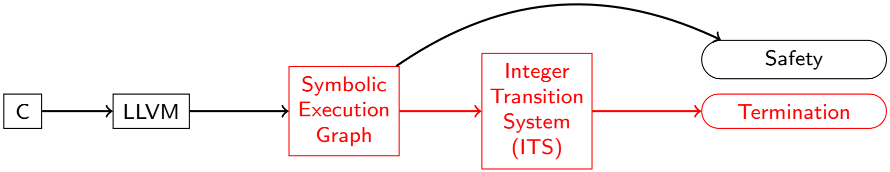


$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\ldots$

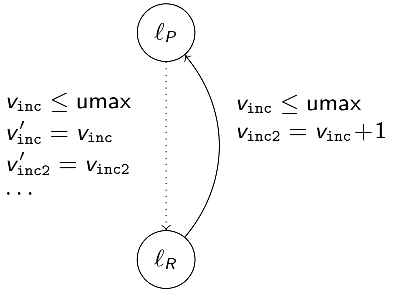


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$

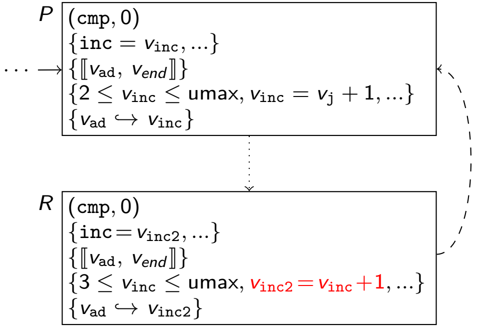
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



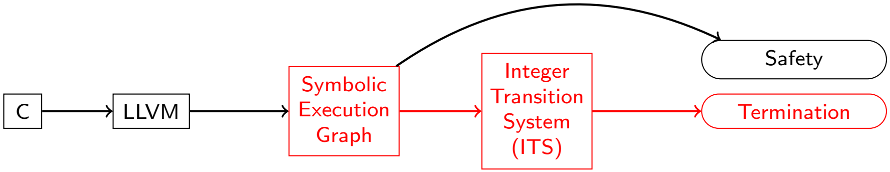
# Termination



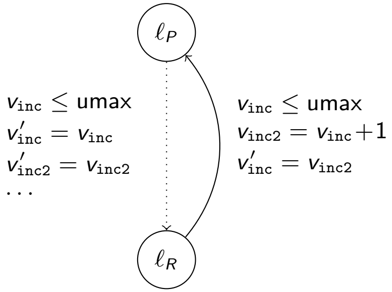
$\ell_P$
$\ell_R$
$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\ldots$
$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v_{\mathtt{inc2}} = v_{\mathtt{inc}}+1$

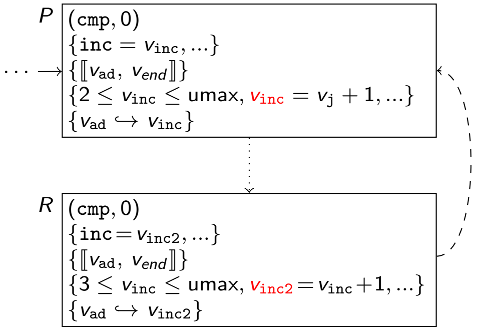


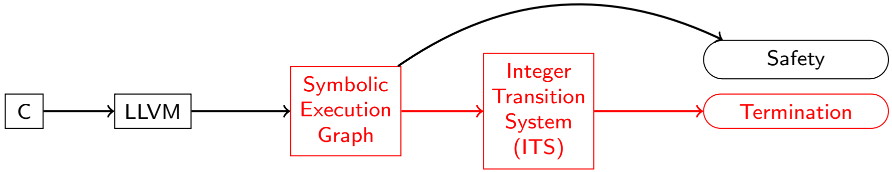
$\ldots$
$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}}+1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



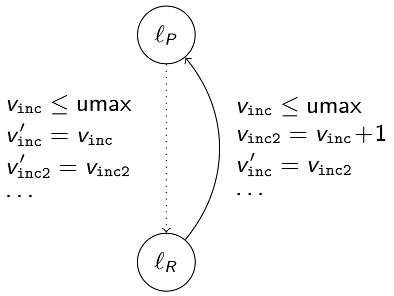
# Termination

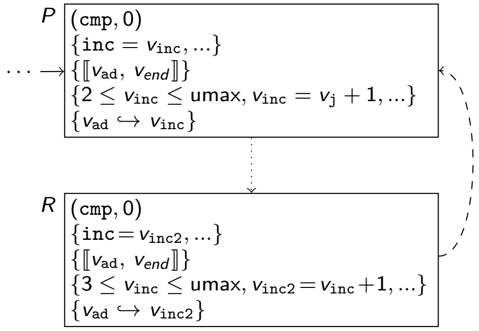

$\ell_P$
$\ell_R$
$v_{\texttt{inc}} \leq \text{umax}$
$v'_{\texttt{inc}} = v_{\texttt{inc}}$
$v'_{\texttt{inc2}} = v_{\texttt{inc2}}$
$\ldots$
$v_{\texttt{inc}} \leq \text{umax}$
$v_{\texttt{inc2}} = v_{\texttt{inc}} + 1$
$v'_{\texttt{inc}} = v_{\texttt{inc2}}$

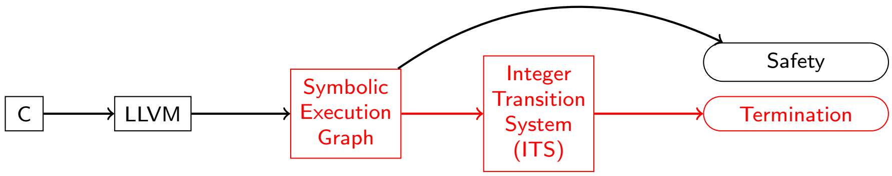

$P$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$R$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \ldots\}$
$\{[\![v_{\texttt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \text{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \ldots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$



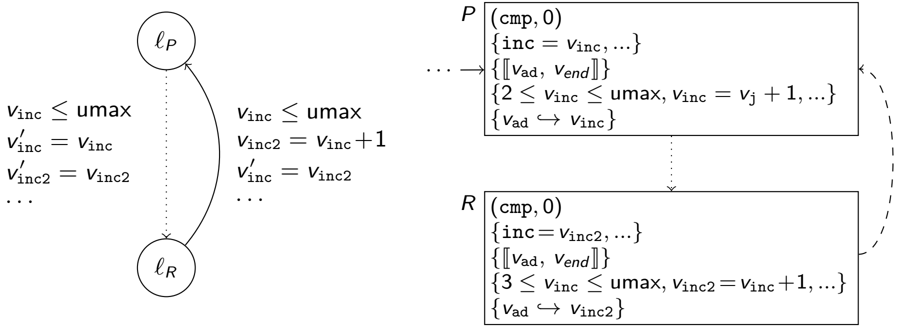
# Termination


$\ell_P$
$\ell_R$
$v_{\mathtt{inc}} \le \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\ldots$
$v_{\mathtt{inc}} \le \mathsf{umax}$
$v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1$
$v'_{\mathtt{inc}} = v_{\mathtt{inc2}}$
$\ldots$

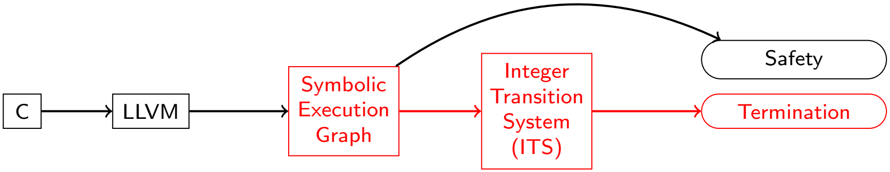

$\ldots$
$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \ldots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \ldots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$



# Termination

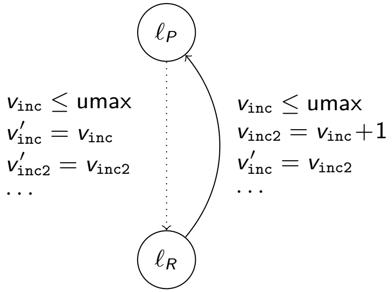


$\ell_P$
$\ell_R$
$v_{\mathtt{inc}} \le \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\dots$
$v_{\mathtt{inc}} \le \mathsf{umax}$
$v_{\mathtt{inc2}} = v_{\mathtt{inc}}+1$
$v'_{\mathtt{inc}} = v_{\mathtt{inc2}}$
$\dots$
$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \le v_{\mathtt{inc}} \le \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}}+1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$

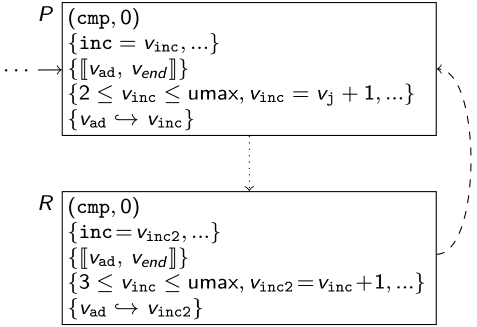


- ITS from cycles of symbolic execution graph

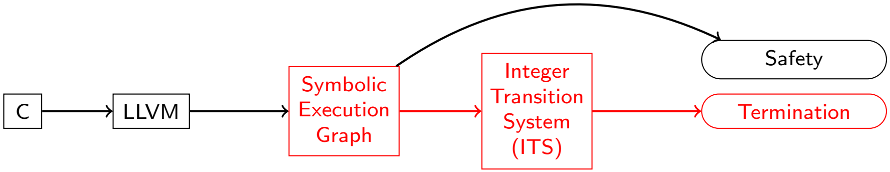
# Termination



$\ell_P$
$\ell_R$
$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v'_{\mathtt{inc}} = v_{\mathtt{inc}}$
$v'_{\mathtt{inc2}} = v_{\mathtt{inc2}}$
$\dots$
$v_{\mathtt{inc}} \leq \mathsf{umax}$
$v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1$
$v'_{\mathtt{inc}} = v_{\mathtt{inc2}}$
$\dots$

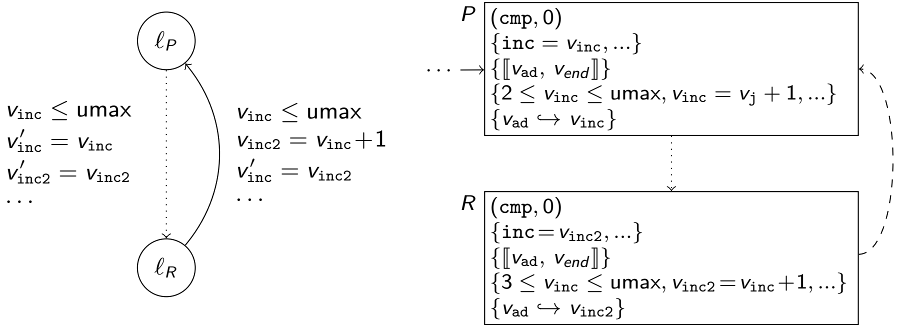


$P$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{2 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc}} = v_{\mathtt{j}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc}}\}$
$\dots$
$R$
$(\mathtt{cmp}, 0)$
$\{\mathtt{inc} = v_{\mathtt{inc2}}, \dots\}$
$\{[\![v_{\mathtt{ad}},\ v_{end}]\!]\}$
$\{3 \leq v_{\mathtt{inc}} \leq \mathsf{umax}, v_{\mathtt{inc2}} = v_{\mathtt{inc}} + 1, \dots\}$
$\{v_{\mathtt{ad}} \hookrightarrow v_{\mathtt{inc2}}\}$

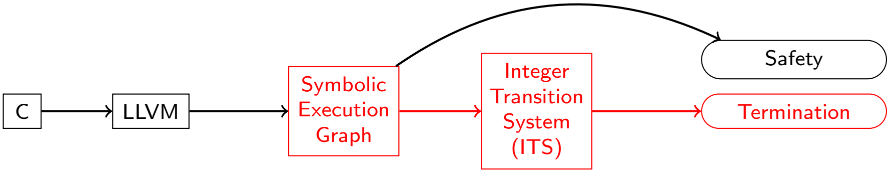


- ITS from cycles of symbolic execution graph
- ITS termination by existing tools

## Termination



$\ell_P$
$\ell_R$
$v_{\texttt{inc}} \leq \mathsf{umax}$
$v'_{\texttt{inc}} = v_{\texttt{inc}}$
$v'_{\texttt{inc2}} = v_{\texttt{inc2}}$
$\dots$
$v_{\texttt{inc}} \leq \mathsf{umax}$
$v_{\texttt{inc2}} = v_{\texttt{inc}} + 1$
$v'_{\texttt{inc}} = v_{\texttt{inc2}}$
$\dots$
$P$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc}}, \dots\}$
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$
$\{2 \leq v_{\texttt{inc}} \leq \mathsf{umax}, v_{\texttt{inc}} = v_{\texttt{j}} + 1, \dots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc}}\}$
$R$
$(\texttt{cmp}, 0)$
$\{\texttt{inc} = v_{\texttt{inc2}}, \dots\}$
$\{[\![v_{\texttt{ad}}, v_{end}]\!]\}$
$\{3 \leq v_{\texttt{inc}} \leq \mathsf{umax}, v_{\texttt{inc2}} = v_{\texttt{inc}} + 1, \dots\}$
$\{v_{\texttt{ad}} \hookrightarrow v_{\texttt{inc2}}\}$



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

## 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

- **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:** case analysis not practical

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

## 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*

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

## 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 `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

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

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

## Multiplication

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

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

- **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 $\models \langle a \rangle \Longrightarrow PV_u(t_1) * PV_u(t_2) \leq \mathsf{umax}_{32}$
- 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

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

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

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

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

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

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

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

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

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

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

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

- **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*
  - extend *KB* by additional information on intervals of result

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

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