Complexity Analysis for Java with AProVE

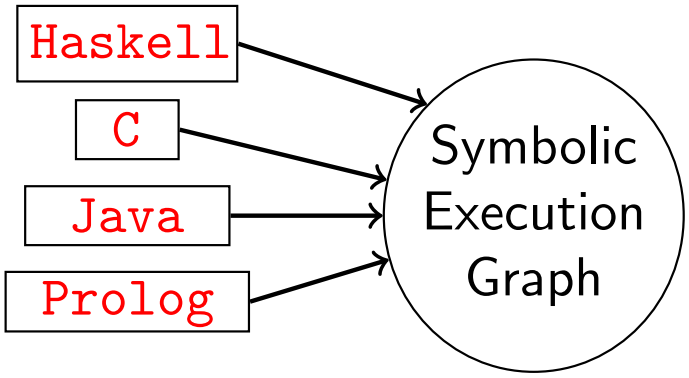
Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

joint work with Florian Frohn

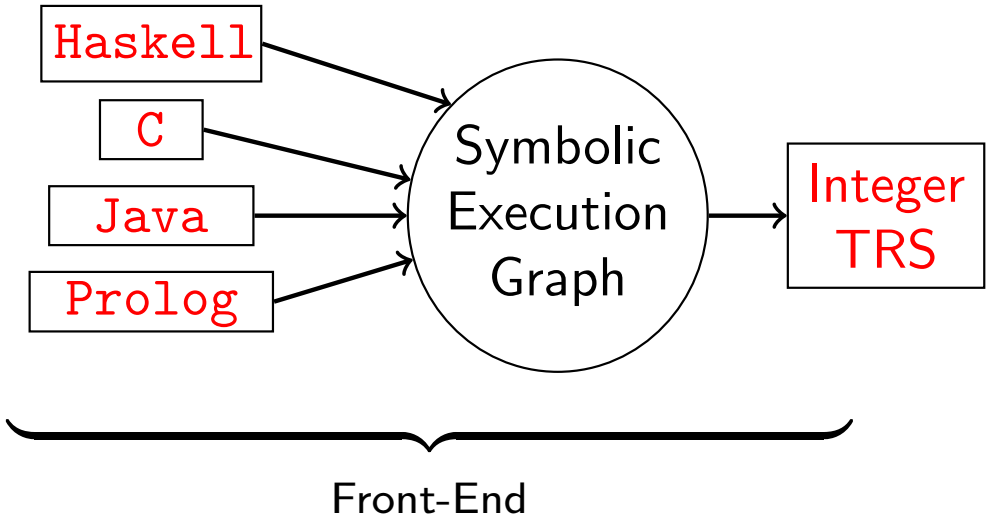
AProVE for Termination Analysis

AProVE for Termination Analysis



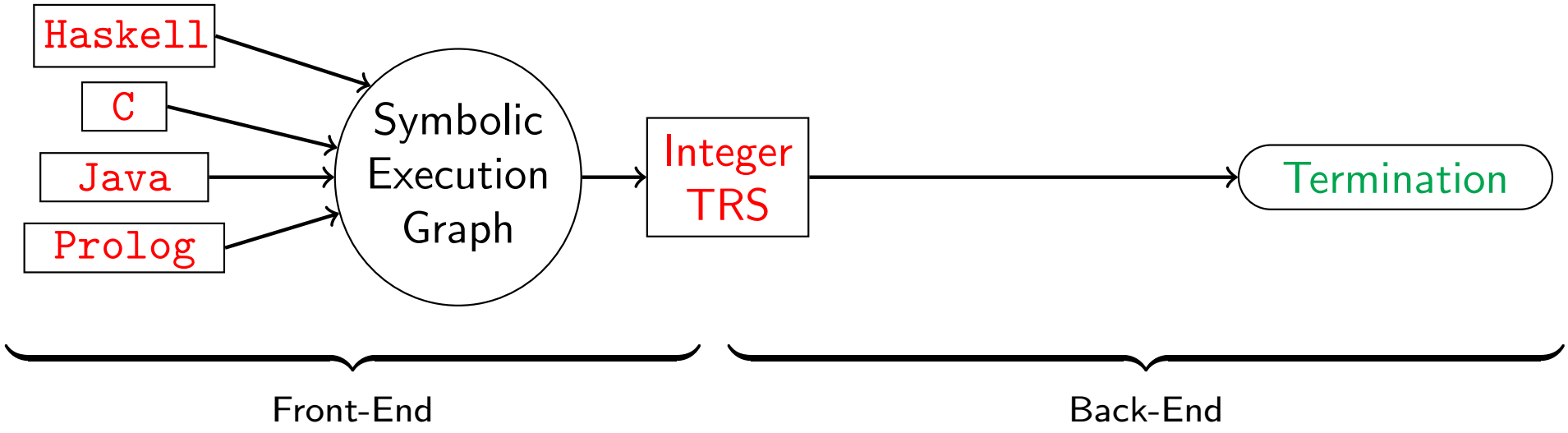
Termination

AProVE for Termination Analysis

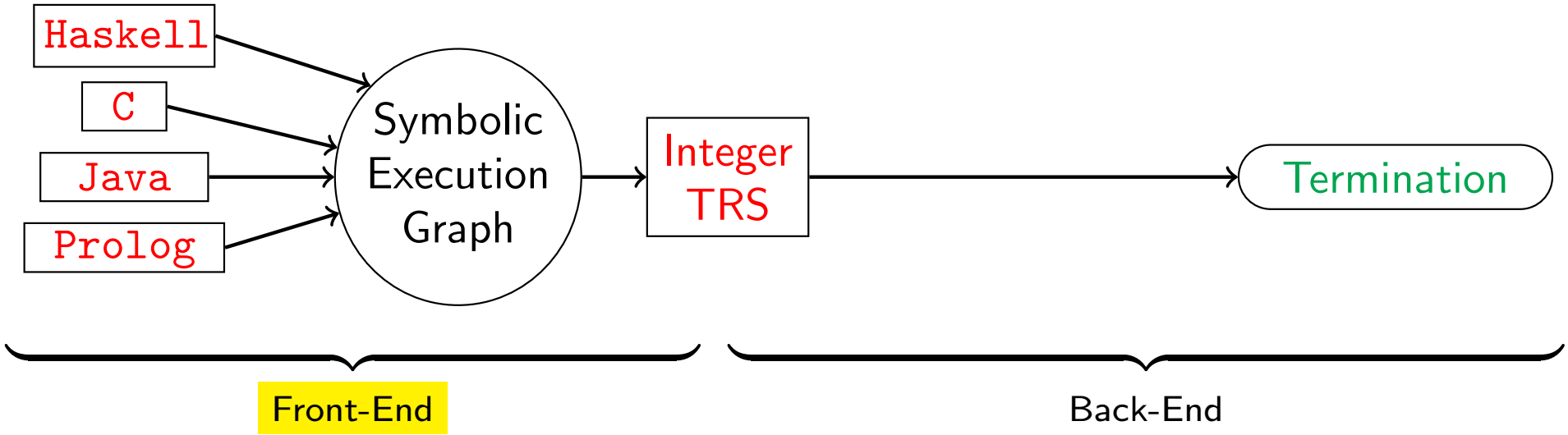


Termination

AProVE for Termination Analysis

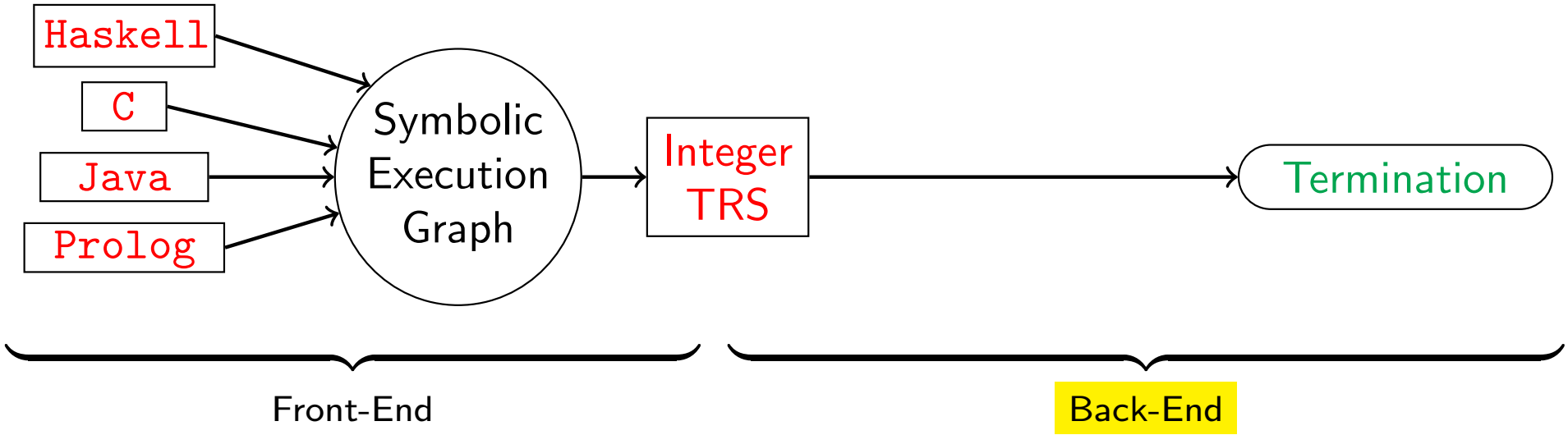


AProVE for Termination Analysis



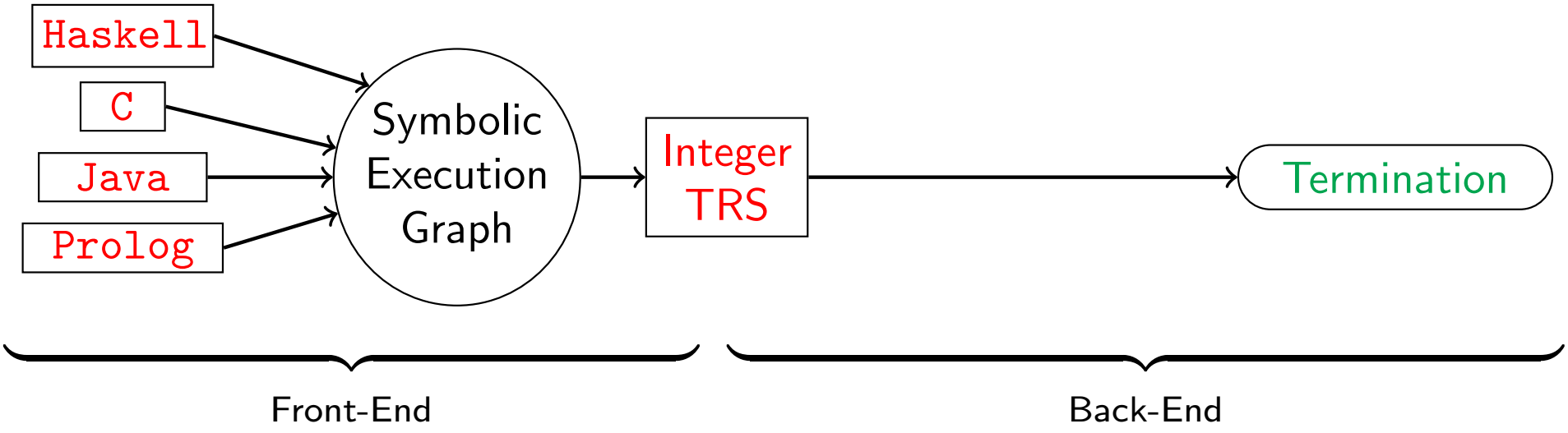
- language-specific features when generating symbolic execution graph

AProVE for Termination Analysis



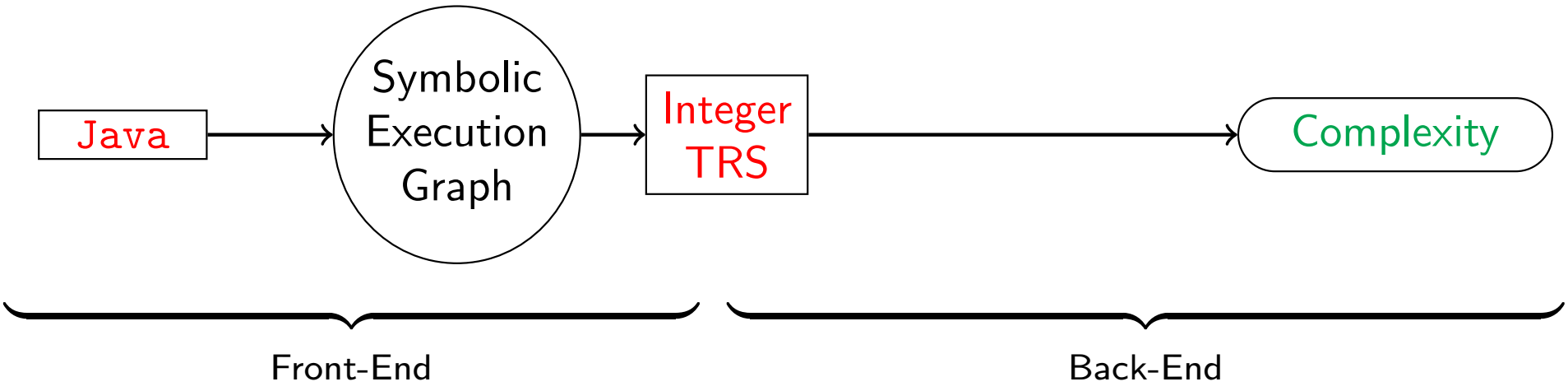
- language-specific features when generating symbolic execution graph
- back-end analyzes Term Rewrite Systems with built-in Integers

AProVE for Termination Analysis

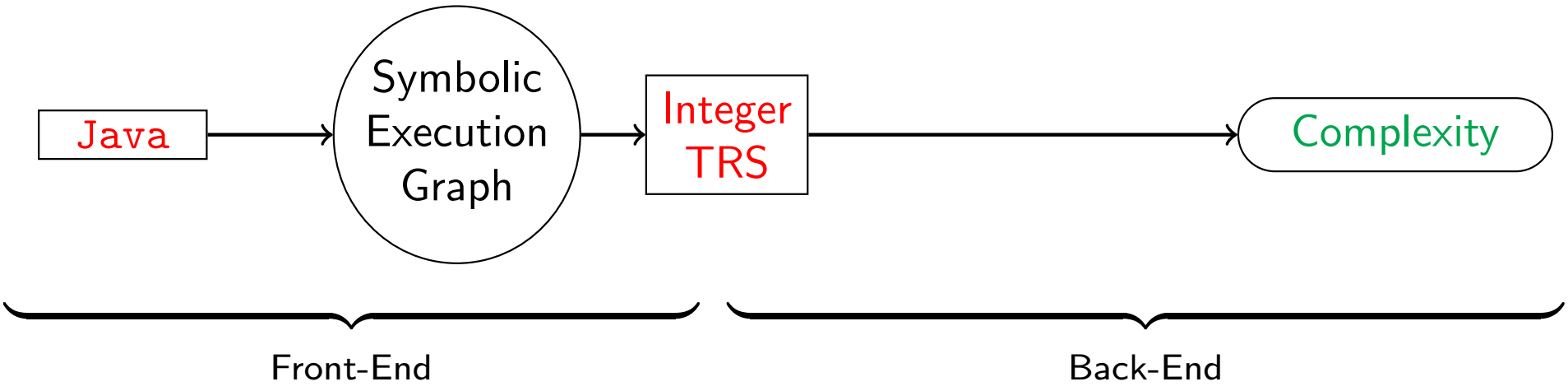


- language-specific features when generating symbolic execution graph
- back-end analyzes Term Rewrite Systems with built-in Integers
- powerful termination analysis
  - Termination Competition since 2004 (Java, C, Haskell, Prolog, TRS)
  - SV-COMP since 2014 (C)

AProVE for Complexity Analysis of Java



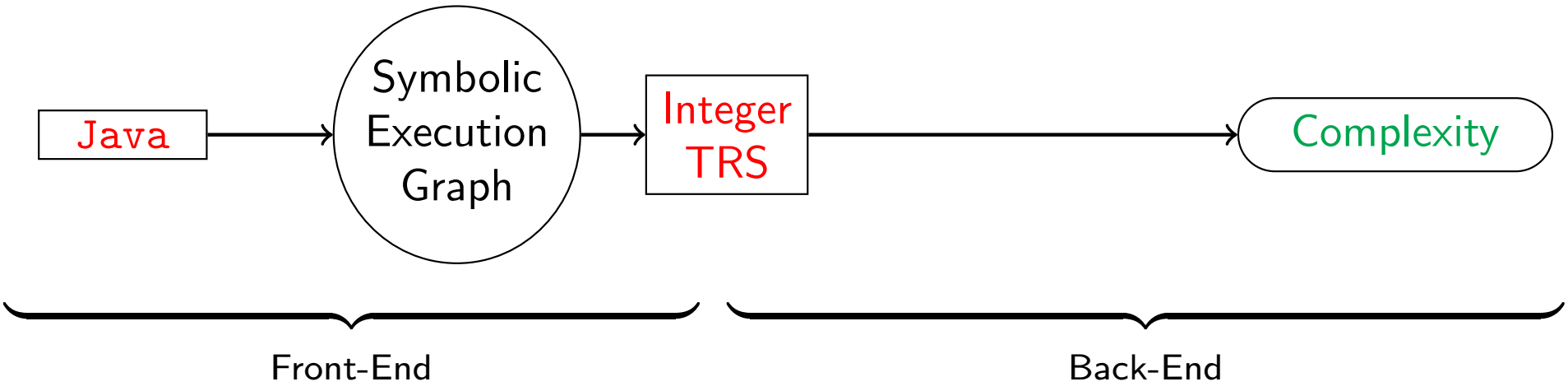
AProVE for Complexity Analysis of Java



Problems

- transformation from Symbolic Execution Graph to Integer TRSs not complexity preserving

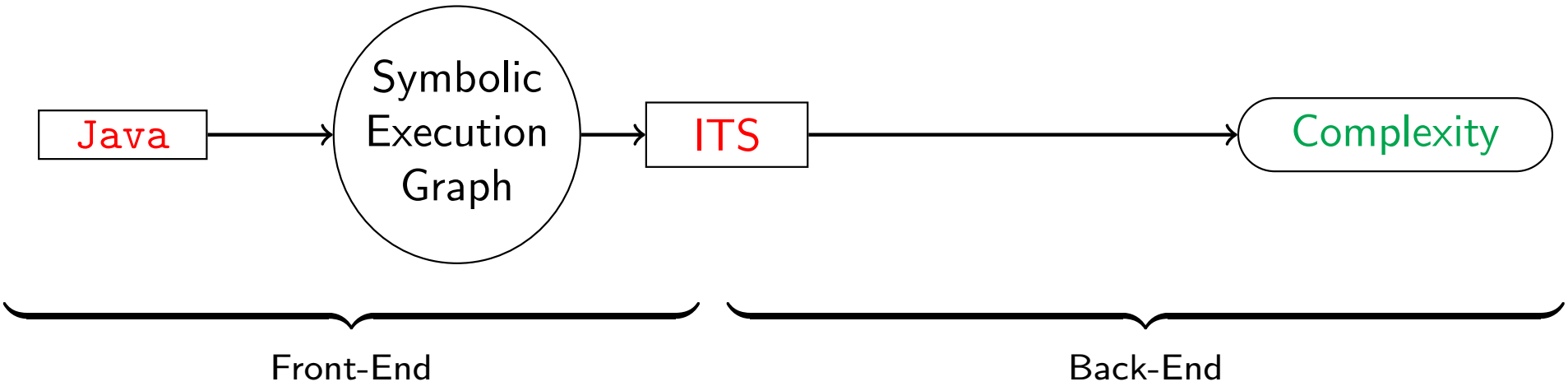
AProVE for Complexity Analysis of Java



Problems

- transformation from Symbolic Execution Graph to Integer TRSs not complexity preserving
- no back-end complexity analyzers for Integer TRSs

AProVE for Complexity Analysis of Java



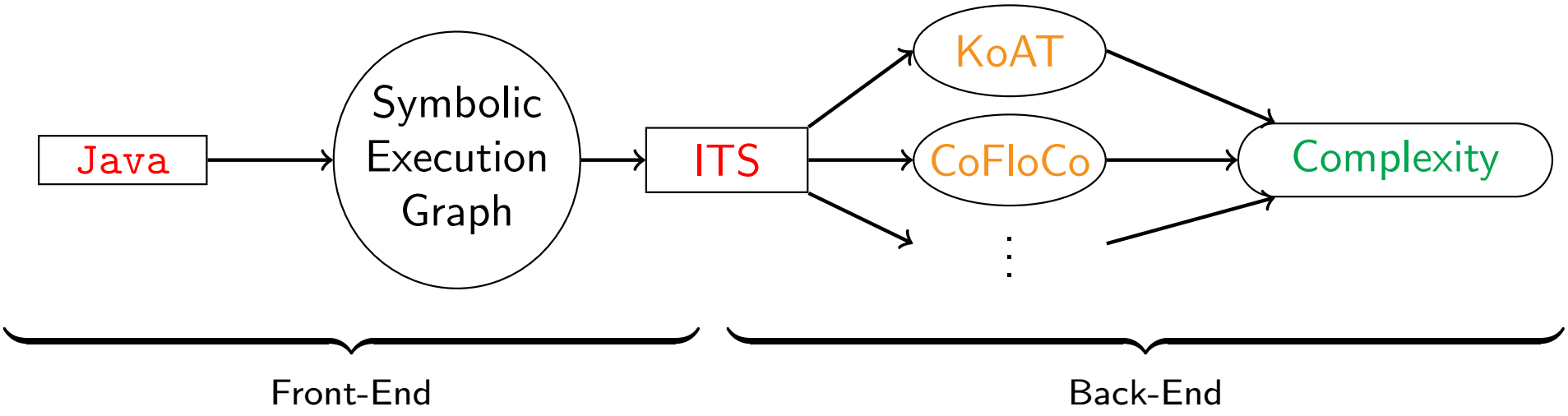
Problems

- transformation from Symbolic Execution Graph to Integer TRSs
  not complexity preserving
- no back-end complexity analyzers for Integer TRSs

Solution

- new transformation from Symbolic Execution Graph to Integer Transition Systems

AProVE for Complexity Analysis of Java



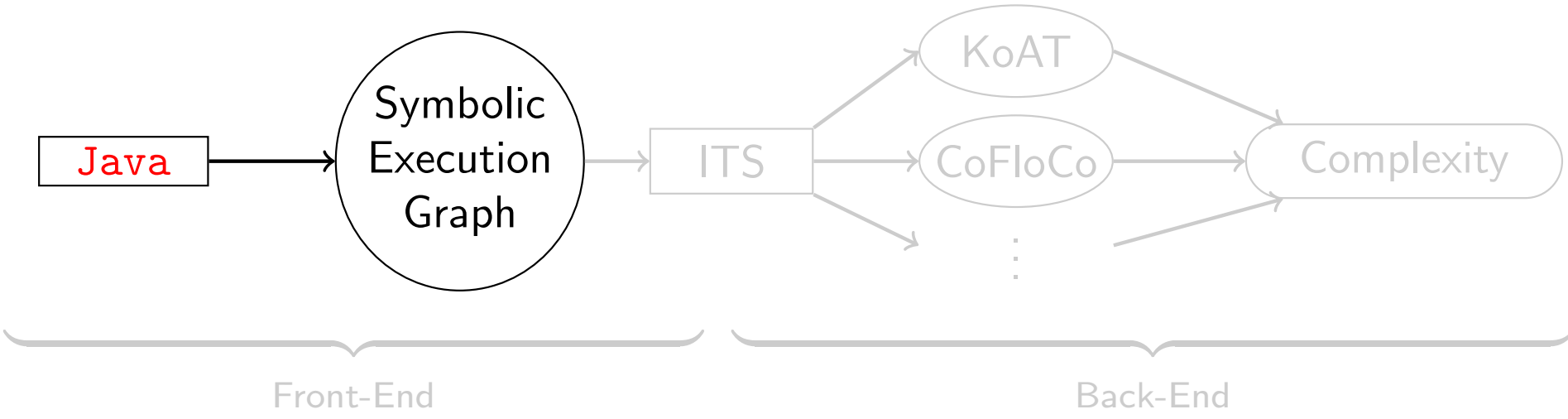
Problems

- transformation from Symbolic Execution Graph to Integer TRSs not complexity preserving
- no back-end complexity analyzers for Integer TRSs

Solution

- new transformation from Symbolic Execution Graph to Integer Transition Systems
- use existing complexity analyzers for ITSs

AProVE for Complexity Analysis of Java



Example: `max`

```
class List {
  int v;
  List n;

  static int max(List l) {
    int m = 0;
    while (l != null) {
      if (l.v > m) {
        m = l.v;
      }
      l = l.n;
    }
    return m;
  }

}
```

Example: max

```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```

```
class List {
  int v;
  List n;

  static int max(List l) {
    int m = 0;
    while (l != null) {
      if (l.v > m) {
        m = l.v;
      }
      l = l.n;
    }
    return m;
  }

}
```

Abstract States of the JVM

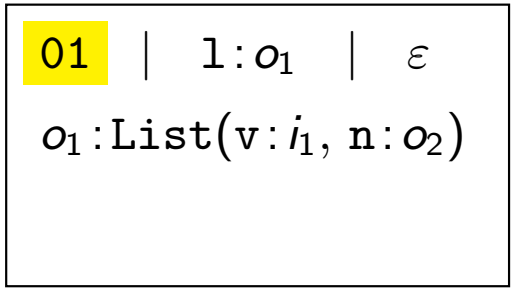
```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```

$$\texttt{01} \quad | \quad \texttt{l}:o_1 \quad | \quad \varepsilon$$

$$o_1 : \texttt{List}(\texttt{v}:i_1, \texttt{n}:o_2)$$

Abstract States of the JVM

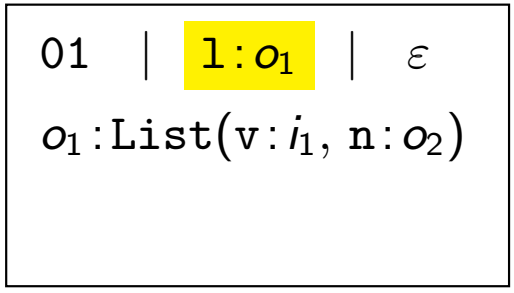
```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```



1. next program instruction

Abstract States of the JVM

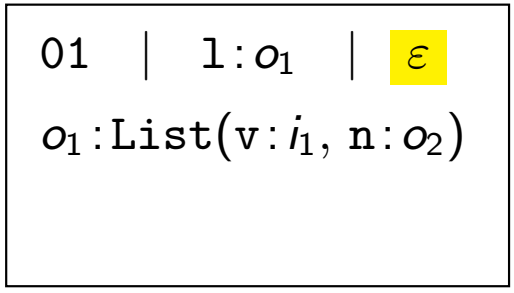
```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```



1. next program instruction
2. values of program variables (value of l is *reference* o_1)

Abstract States of the JVM

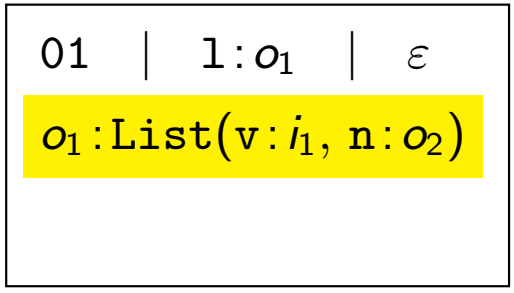
```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```



1. next program instruction
2. values of program variables (value of l is *reference* o_1)
3. values on the operand stack

Abstract States of the JVM

```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```

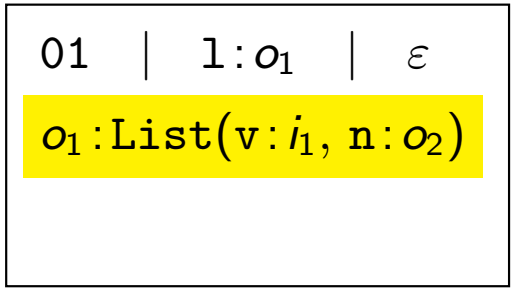


1. next program instruction
2. values of program variables (value of l is *reference* o_1)
3. values on the operand stack

information about the heap

Abstract States of the JVM

```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```



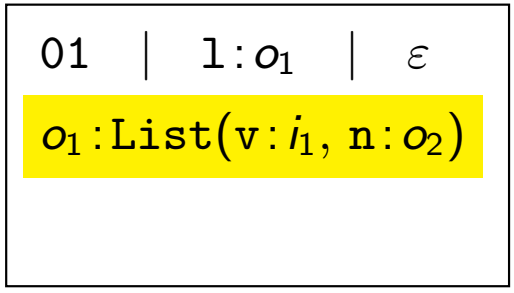
1. next program instruction
2. values of program variables (value of l is *reference* o_1)
3. values on the operand stack

information about the heap

- object at o_1 is List,
  v-field has value i_1, n-field has value o_2

Abstract States of the JVM

```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```



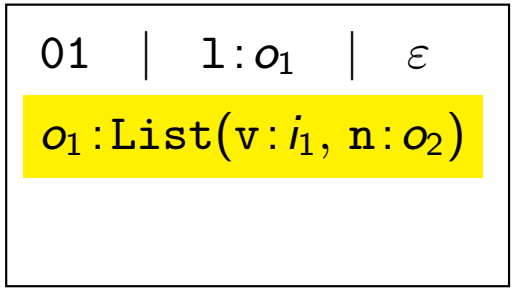
1. next program instruction
2. values of program variables (value of l is *reference* o_1)
3. values on the operand stack

information about the heap

- object at o_1 is List, v-field has value i_1, n-field has value o_2
- o_1 and o_2 do not share or alias and point to tree-shaped objects

Abstract States of the JVM

```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```



1. next program instruction
2. values of program variables (value of l is *reference* o_1)
3. values on the operand stack

information about the heap

explicit sharing information

- object at o_1 is List, v-field has value i_1, n-field has value o_2
- o_1 and o_2 do not share or alias and point to tree-shaped objects

Abstract States of the JVM

```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```

$01 \mid \mathtt{l}:o_1 \mid \varepsilon$

$o_1 : \mathtt{List}(\mathtt{v}:i_1, \mathtt{n}:o_2)$

$o_1 \not\perp o_2$

1. next program instruction
2. values of program variables (value of `l` is *reference* o_1)
3. values on the operand stack

information about the heap

explicit sharing information

- object at o_1 is `List`, `v`-field has value i_1, `n`-field has value o_2
- o_1 and o_2 may share and point to tree-shaped objects

Abstract States of the JVM

```
01: iconst_0        // load 0 to opstack
02: istore_1        // store 0 to var 1 (m)
03: aload_0         // load l to opstack
04: ifnull 16       // jump if l is null
05: aload_0         // load l to opstack
06: getfield v      // load l.v to opstack
07: iload_1         // load m to opstack
08: if_icmple 12    // jump if l.v <= m
09: aload_0         // load l to opstack
10: getfield v      // load l.v to opstack
11: istore_1        // store l.v into m
12: aload_0         // load l to opstack
13: getfield n      // load l.n to opstack
14: astore_0        // store l.n into l
15: goto 3
16: iload_1         // load m to opstack
17: ireturn         // return m
```

$$01 \mid \mathtt{l}:o_1 \mid \varepsilon$$

$$o_1 : \mathtt{List}(\mathtt{v}:i_1, \mathtt{n}:o_2)$$

$$o_1 =^{?} o_2$$

1. next program instruction
2. values of program variables (value of `l` is *reference* o_1)
3. values on the operand stack

information about the heap

- object at o_1 is `List`, `v`-field has value i_1, `n`-field has value o_2
- o_1 and o_2 may alias and point to tree-shaped objects

explicit sharing information

Abstract States of the JVM

```
01: iconst_0       // load 0 to opstack
02: istore_1       // store 0 to var 1 (m)
03: aload_0        // load l to opstack
04: ifnull 16      // jump if l is null
05: aload_0        // load l to opstack
06: getfield v     // load l.v to opstack
07: iload_1        // load m to opstack
08: if_icmple 12   // jump if l.v <= m
09: aload_0        // load l to opstack
10: getfield v     // load l.v to opstack
11: istore_1       // store l.v into m
12: aload_0        // load l to opstack
13: getfield n     // load l.n to opstack
14: astore_0       // store l.n into l
15: goto 3
16: iload_1        // load m to opstack
17: ireturn        // return m
```

$$01 \quad | \quad \mathtt{l}:o_1 \quad | \quad \varepsilon$$

$$o_1 : \mathtt{List}(\mathtt{v}:i_1, \mathtt{n}:o_2)$$

$$o_1 =^{?} o_2, \quad o_1!$$

1. next program instruction
2. values of program variables (value of `l` is *reference* o_1)
3. values on the operand stack

information about the heap

- object at o_1 is `List`, `v`-field has value i_1, `n`-field has value o_2
- o_1 and o_2 may alias and o_1 may point to a non-tree-shaped object

explicit sharing information

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

A

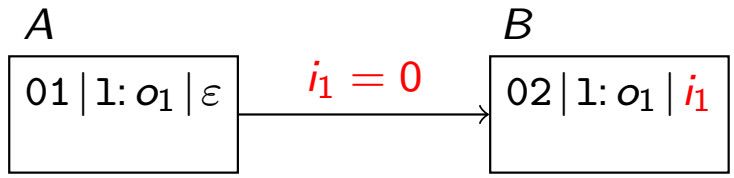
01	l: o_1	ε

State A:

- o_1 is tree-shaped `List` or `null`

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

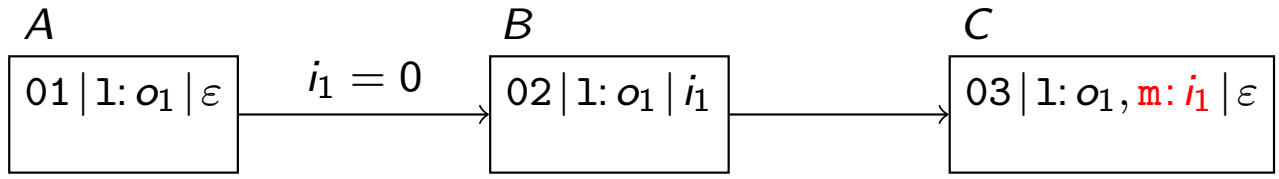


State B:

- load constant 0 on opstack
- *evaluation edge* from A to B
- labeled by condition $i_1 = 0$

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

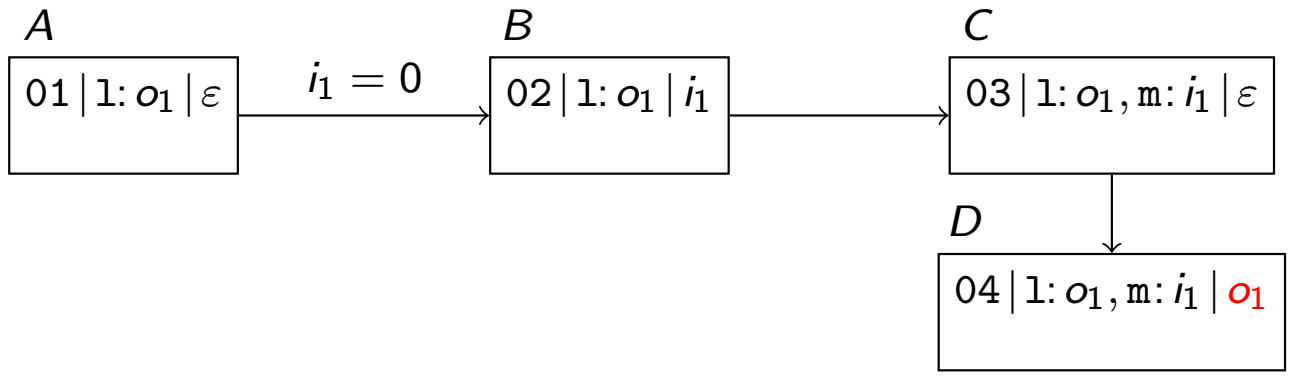


State C:

- store i_1 in program variable m

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

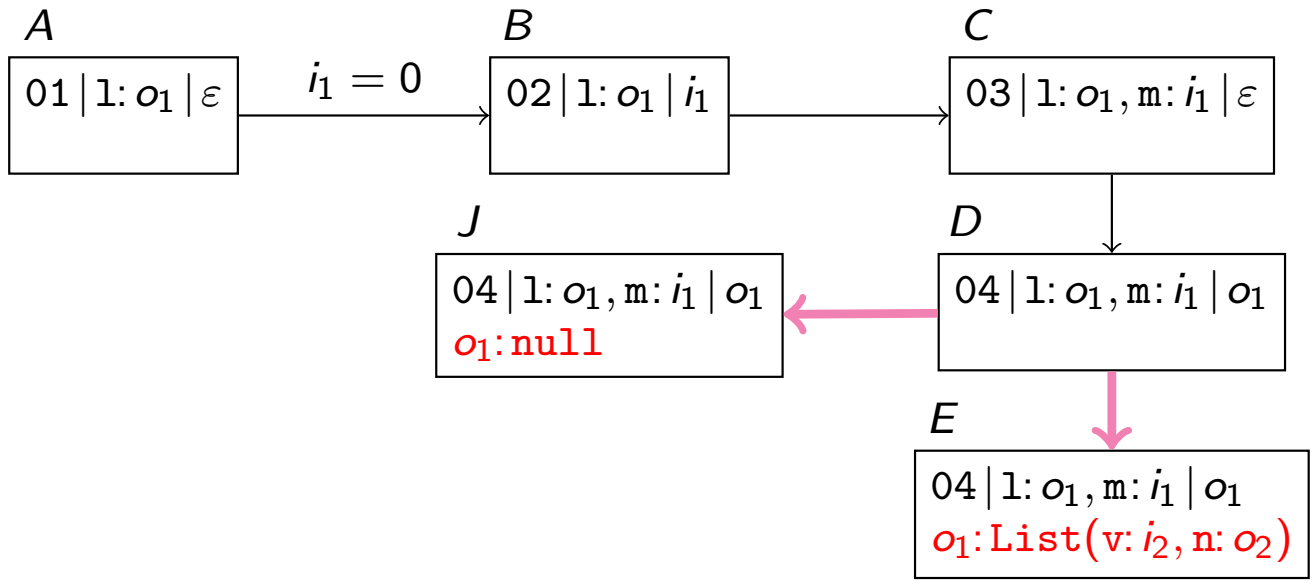


State *D*:

- load o_1 on opstack

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

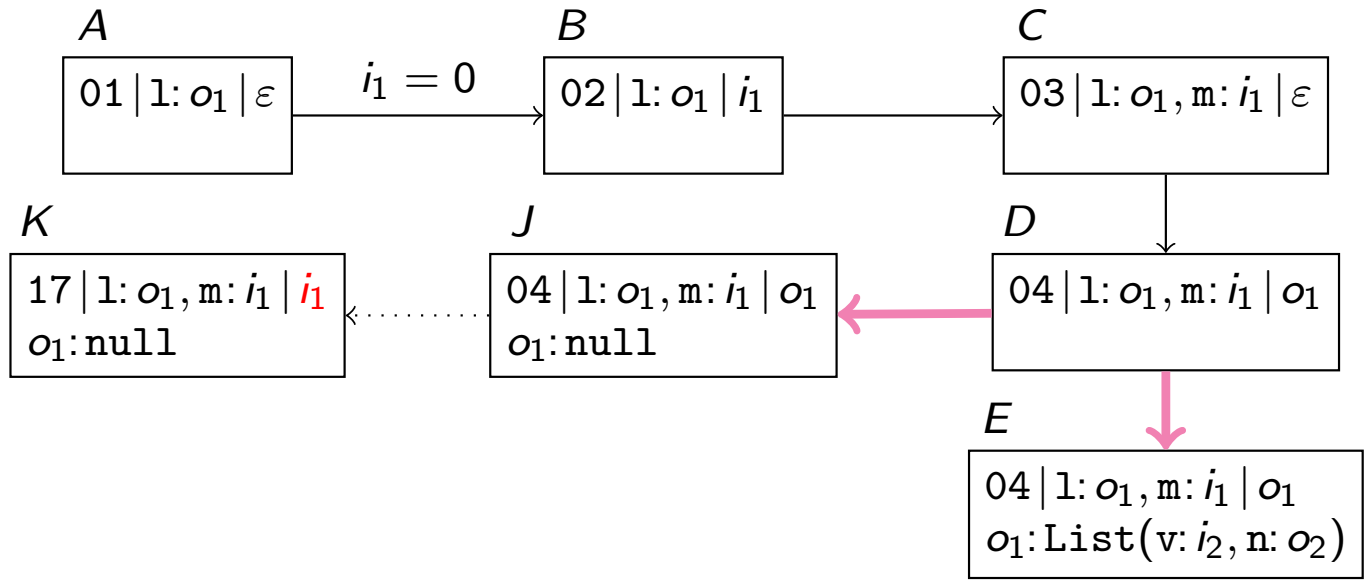


States E and J:

- need to know whether o_1 is `null`
- *refine* information about heap (refinement edges)

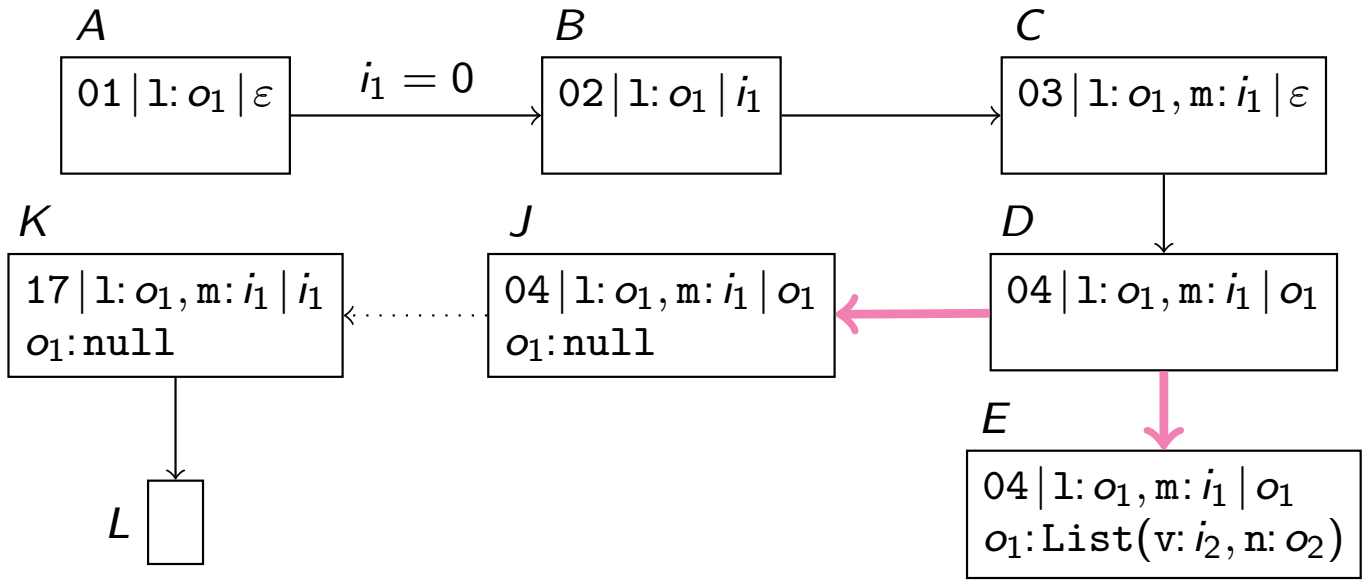
```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```



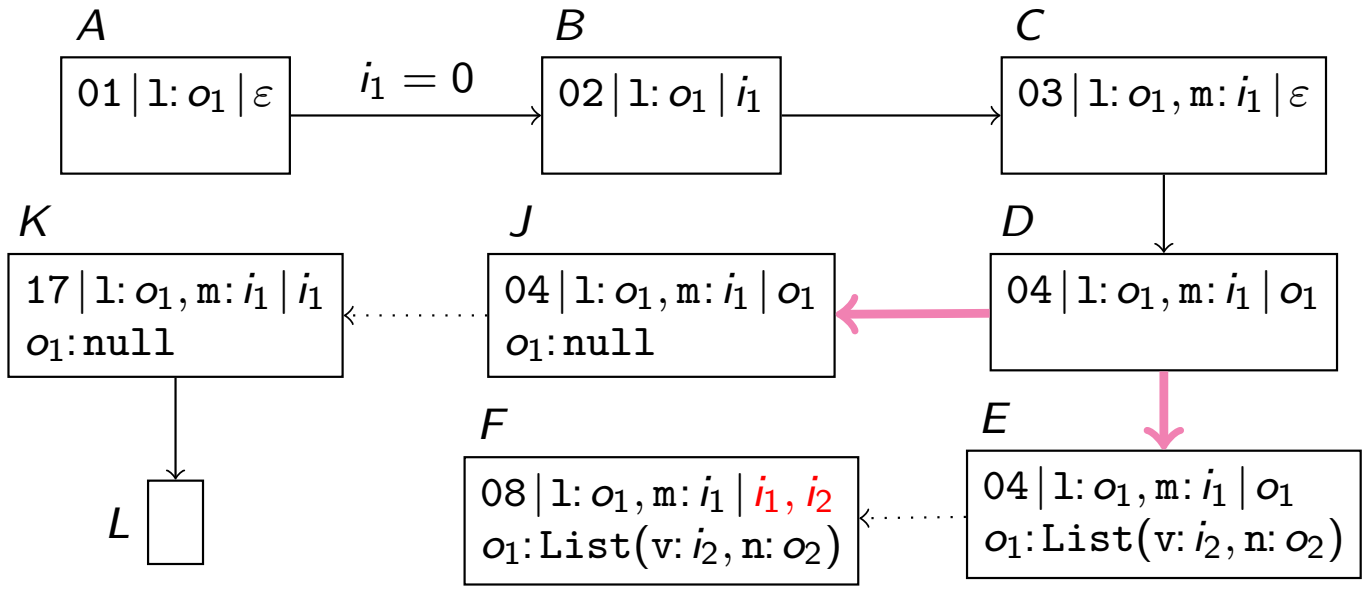

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```




```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

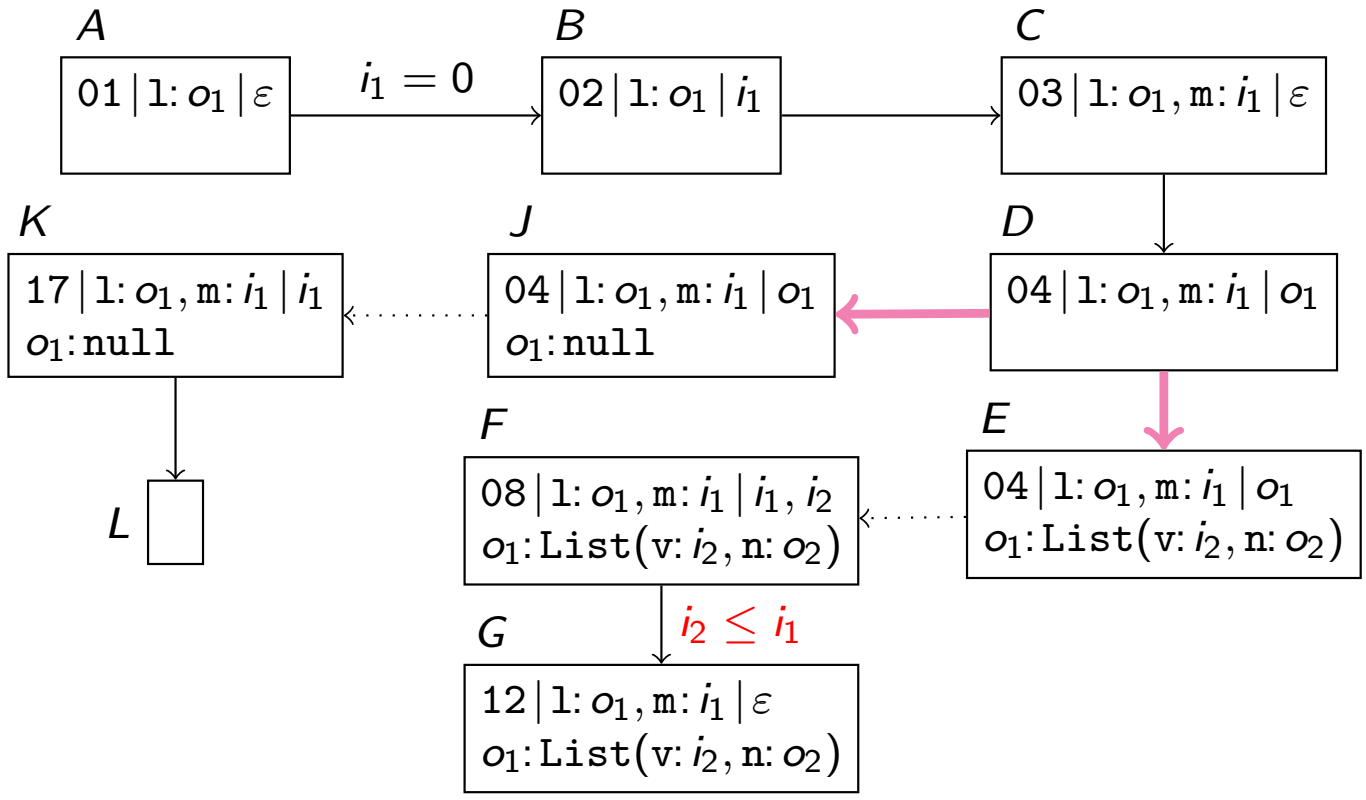


State F:

- load `l.v` (i_2) and `m` (i_1) on opstack

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

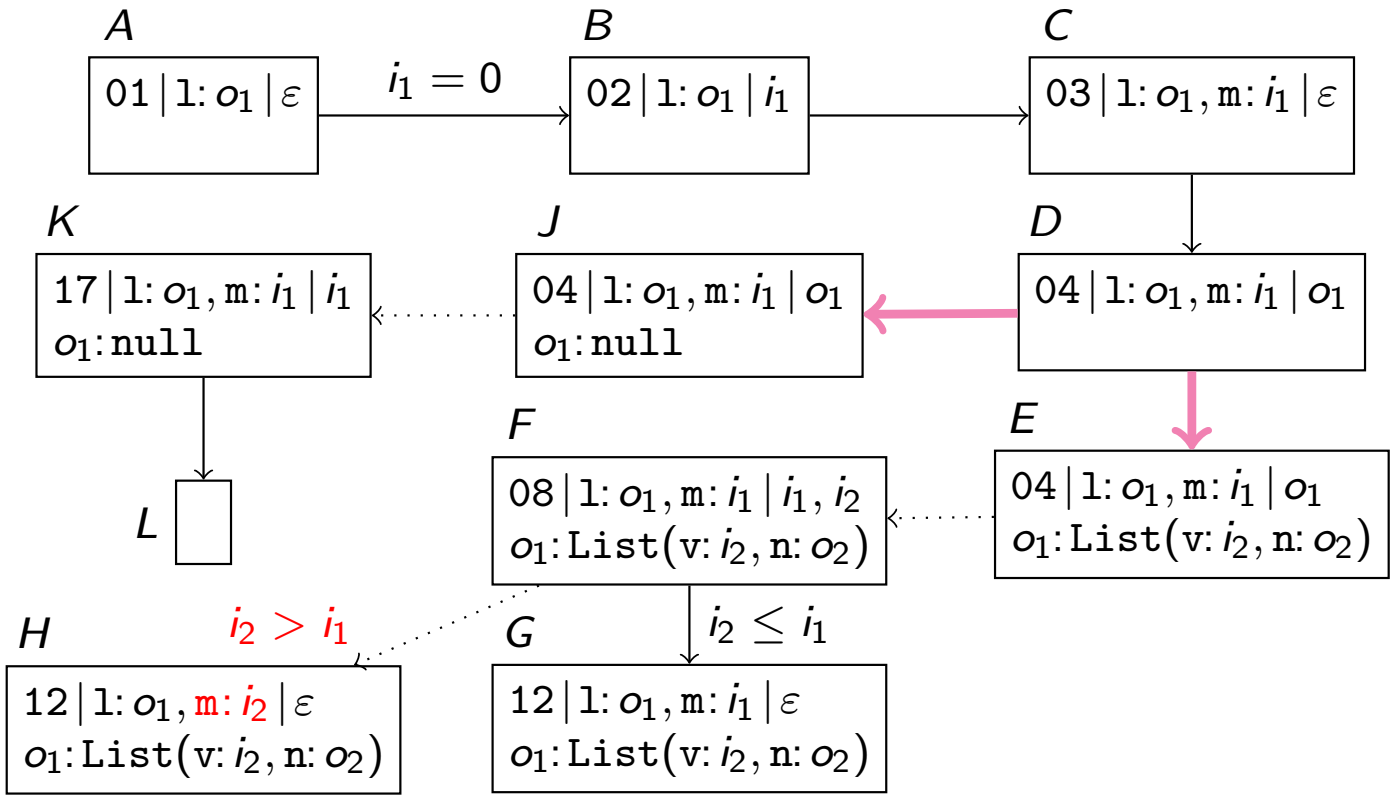



```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

State G:

- goto 12 if $\texttt{l.v} \leq \texttt{m}$
- *evaluation edge* labeled by condition $i_2 \leq i_1$

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

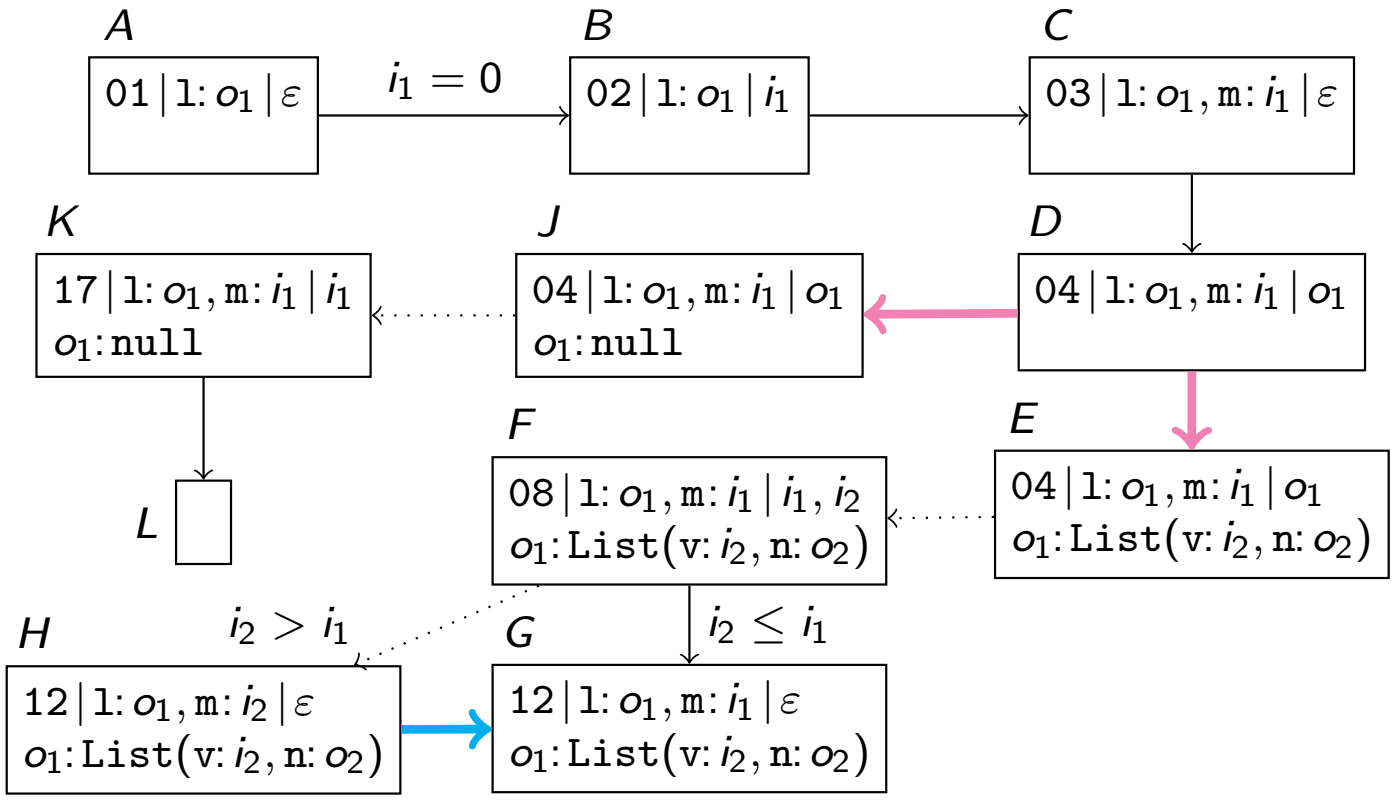



```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

State *H*:

- labeled by condition $i_2 > i_1$
- store `l.v` (i_2) in program variable `m`

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

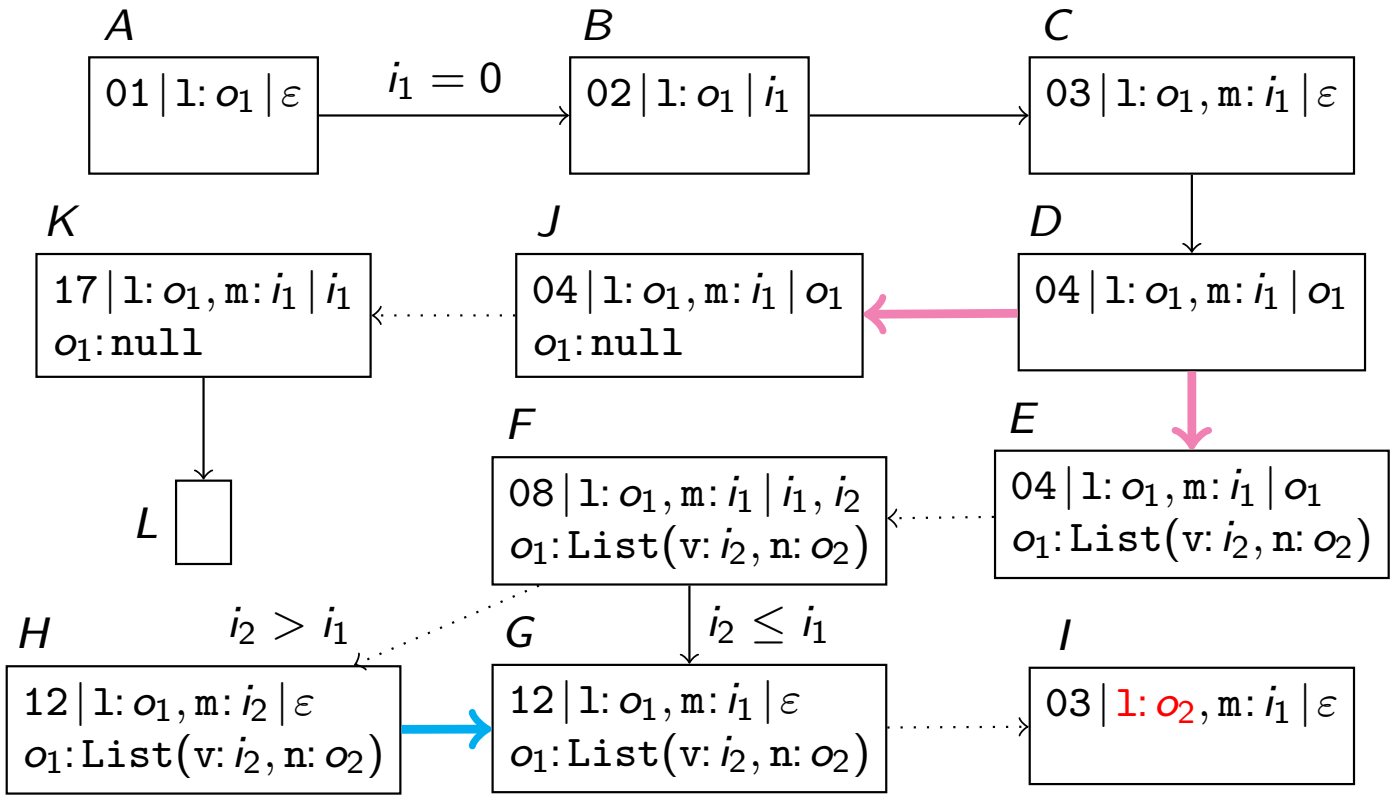



```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

State *H*:

- labeled by condition $i_2 > i_1$
- store `l.v` (i_2) in program variable `m`
- *G* more general than *H* (generalization edge)

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

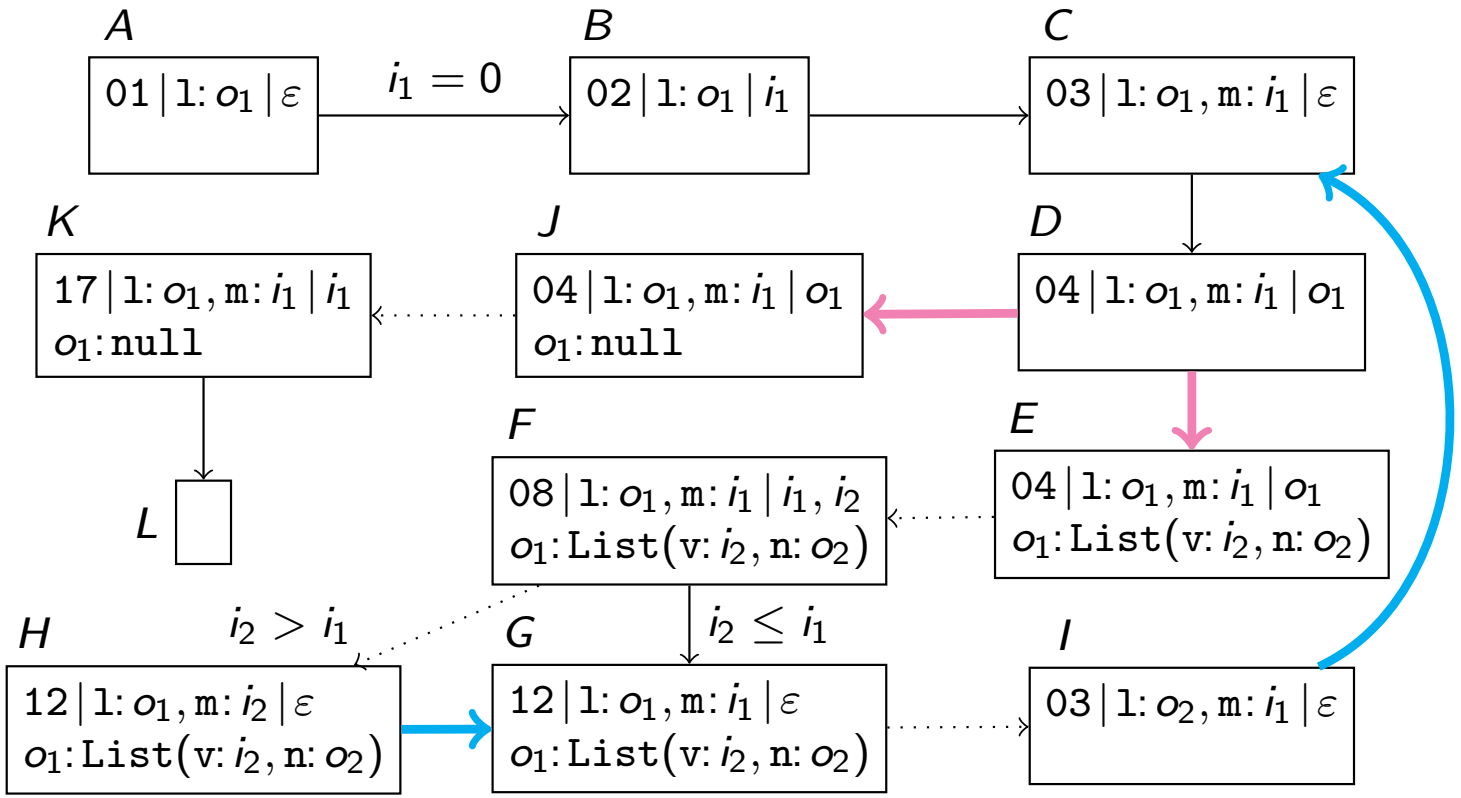



```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

State *l*:

- store `l.n` (o_2) in program variable `l`
- goto Line 3

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```

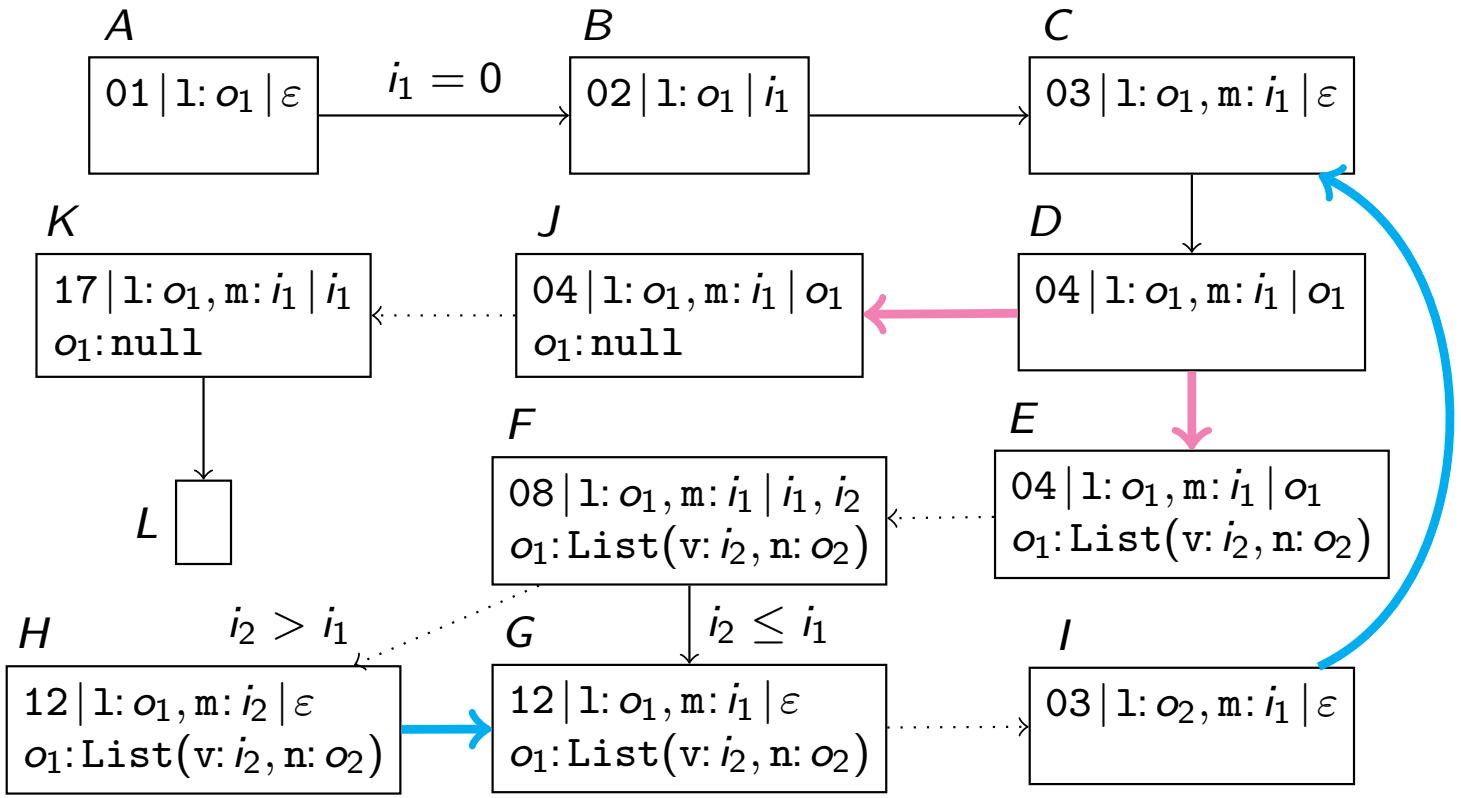



```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

State *I*:

- store `l.n` (o_2) in program variable `l`
- goto Line 3
- *I* corresponds to *C* (generalization edge)

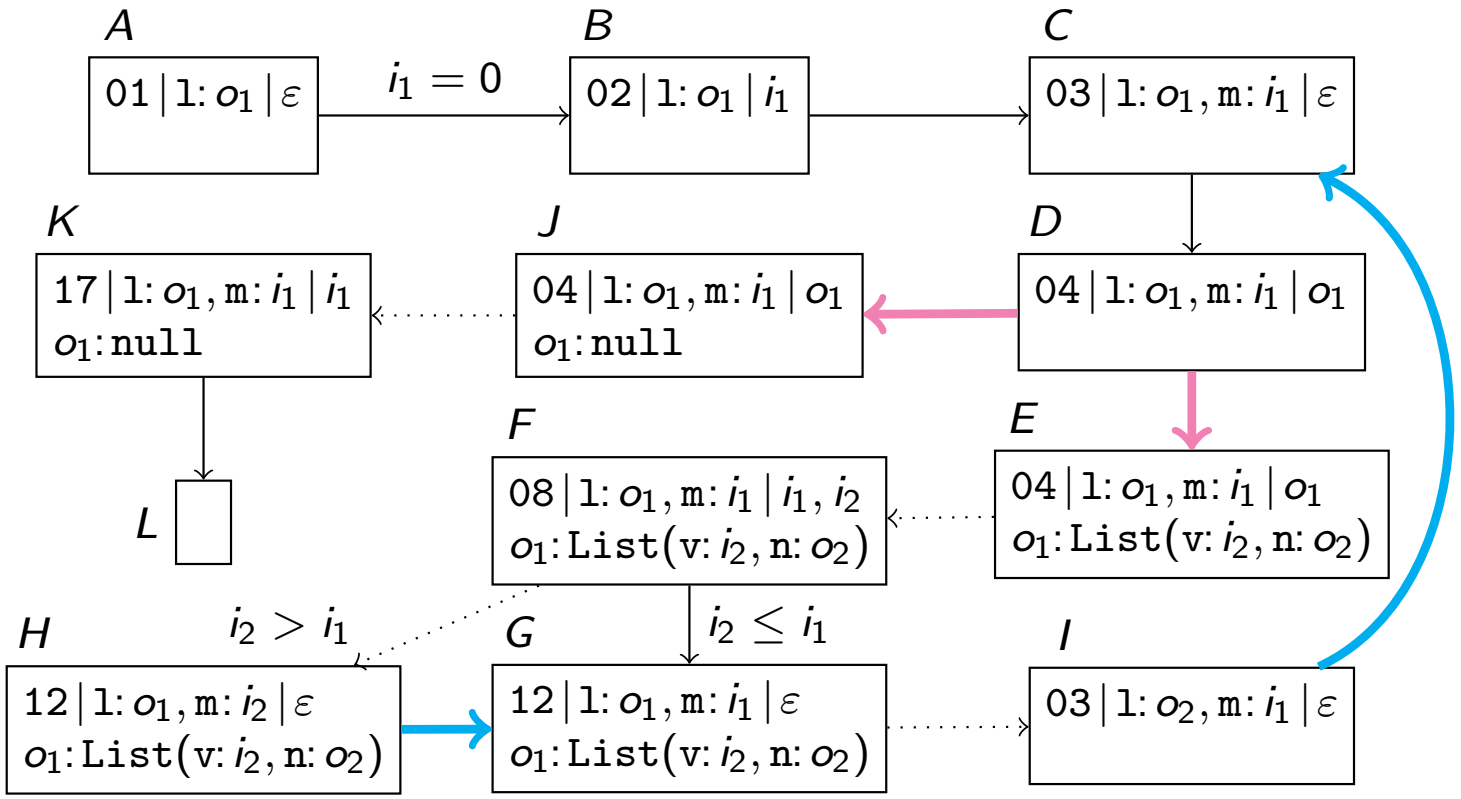
```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```



Symbolic Execution Graph

- expand nodes until all leaves correspond to program ends

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
12: aload_0
13: getfield n
14: astore_0
15: goto 3
16: iload_1
17: ireturn
```



Symbolic Execution Graph

- expand nodes until all leaves correspond to program ends
- by appropriate generalization steps,
  one always reaches a *finite* symbolic execution graph

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o

$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$

$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o

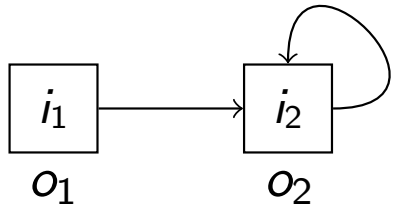
$$\texttt{01} \mid \texttt{l}{:}o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}{:}i_1, \texttt{n}{:}o_2)$$
$$o_2 : \texttt{List}(\texttt{v}{:}i_2, \texttt{n}{:}o_2)$$

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$

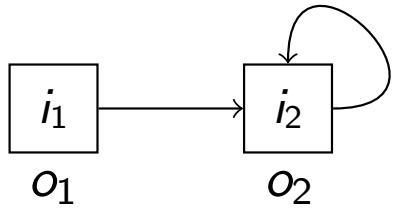


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$

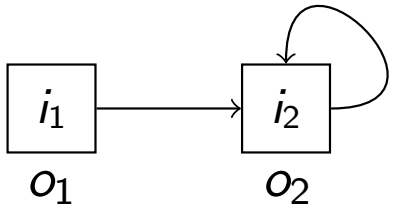


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$

$$
\begin{array}{l}
\texttt{01} \mid \texttt{l}{:}o_1 \mid \varepsilon \\
o_1 : \texttt{List}(\texttt{v}{:}i_1, \texttt{n}{:}o_2) \\
o_2 : \texttt{List}(\texttt{v}{:}i_2, \texttt{n}{:}o_2)
\end{array}
$$

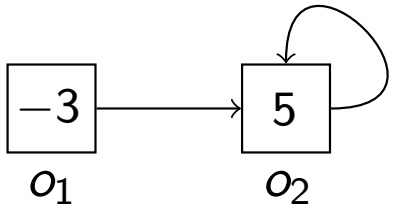


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$

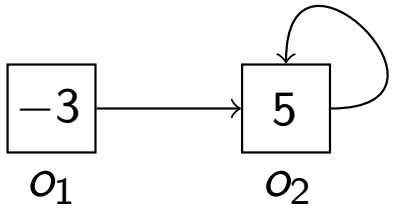


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$

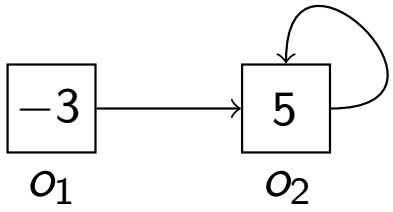


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c
- every concrete Java evaluation can be *embedded*
  into the symbolic execution graph

$$\texttt{01} \mid \texttt{l}{:}o_1 \mid \varepsilon$$
$$o_1{:}\texttt{List}(\texttt{v}{:}i_1, \texttt{n}{:}o_2)$$
$$o_2{:}\texttt{List}(\texttt{v}{:}i_2, \texttt{n}{:}o_2)$$

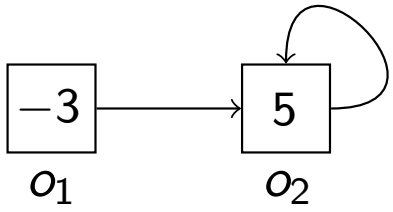


Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$



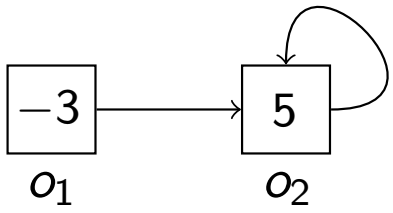
Size of object at reference o in concrete state (c, τ)

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$



Size of object at reference o in concrete state (c, τ)

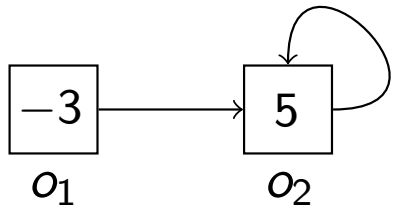
$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ) if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

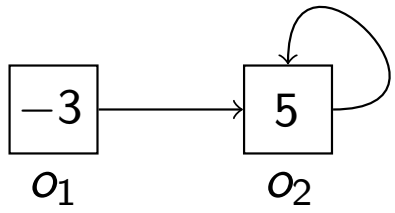
$\|o_2\| = 1 + |5| = 6$

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ) if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

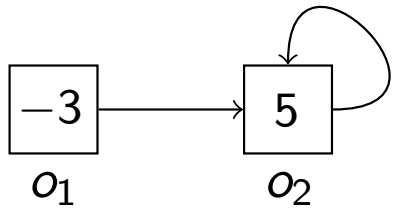
$$\|o_2\| = 1 + |5| = 6$$
$$\|o_1\| = 2 + |-3| + |5| = 10$$

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ) if s is more general than c

$$\texttt{01} \mid \texttt{l}{:}o_1 \mid \varepsilon$$
$$o_1{:}\texttt{List}(\texttt{v}{:}i_1, \texttt{n}{:}o_2)$$
$$o_2{:}\texttt{List}(\texttt{v}{:}i_2, \texttt{n}{:}o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

$$\|o_2\| = 1 + |5| = 6$$
$$\|o_1\| = 2 + |-3| + |5| = 10$$

Complexity Bound b for abstract state s

Complexity of Java Methods

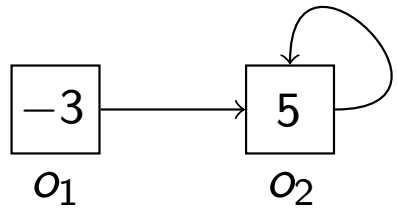
Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ) if s is more general than c

o_1 | 1 : o_1 | ε

o_1 : List(v : i_1, n : o_2)

o_2 : List(v : i_2, n : o_2)



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

$\|o_2\| = 1 + |5| = 6$

$\|o_1\| = 2 + |-3| + |5| = 10$

Complexity Bound b for abstract state s

b is arithmetic term with

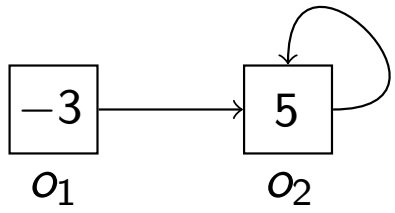
$b \geq$ length of any Java evaluation starting with corresponding concrete state represented by s

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}: i_1, \texttt{n}: o_2)$$
$$o_2 : \texttt{List}(\texttt{v}: i_2, \texttt{n}: o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

$$\|o_2\| = 1 + |5| = 6$$
$$\|o_1\| = 2 + |-3| + |5| = 10$$

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with

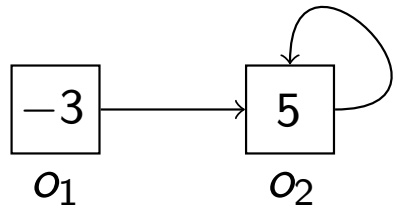
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ)
  if s is more general than c

$$\texttt{01} \mid \texttt{l}:o_1 \mid \varepsilon$$
$$o_1 : \texttt{List}(\texttt{v}:i_1, \texttt{n}:o_2)$$
$$o_2 : \texttt{List}(\texttt{v}:i_2, \texttt{n}:o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

$$\|o_2\| = 1 + |5| = 6$$
$$\|o_1\| = 2 + |-3| + |5| = 10$$

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

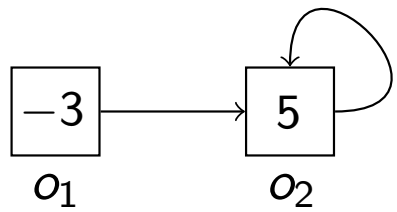
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

Complexity of Java Methods

Concrete State (c, τ)

- state c with full information on symbolic references o
- τ maps symb. integers i to $\mathbb{Z}$: $\tau(i_1) = -3, \quad \tau(i_2) = 5$
- abstract state s *represents* (c, τ) if s is more general than c

$$\texttt{01} \mid \texttt{l}{:}o_1 \mid \varepsilon$$
$$o_1{:}\texttt{List}(\texttt{v}{:}i_1, \texttt{n}{:}o_2)$$
$$o_2{:}\texttt{List}(\texttt{v}{:}i_2, \texttt{n}{:}o_2)$$



Size of object at reference o in concrete state (c, τ)

$\|o\|$ = number of objects reachable from o + absolute values of integers in their fields

$$\|o_2\| = 1 + |5| = 6$$
$$\|o_1\| = 2 + |-3| + |5| = 10$$

Complexity Bound b for abstract state s with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

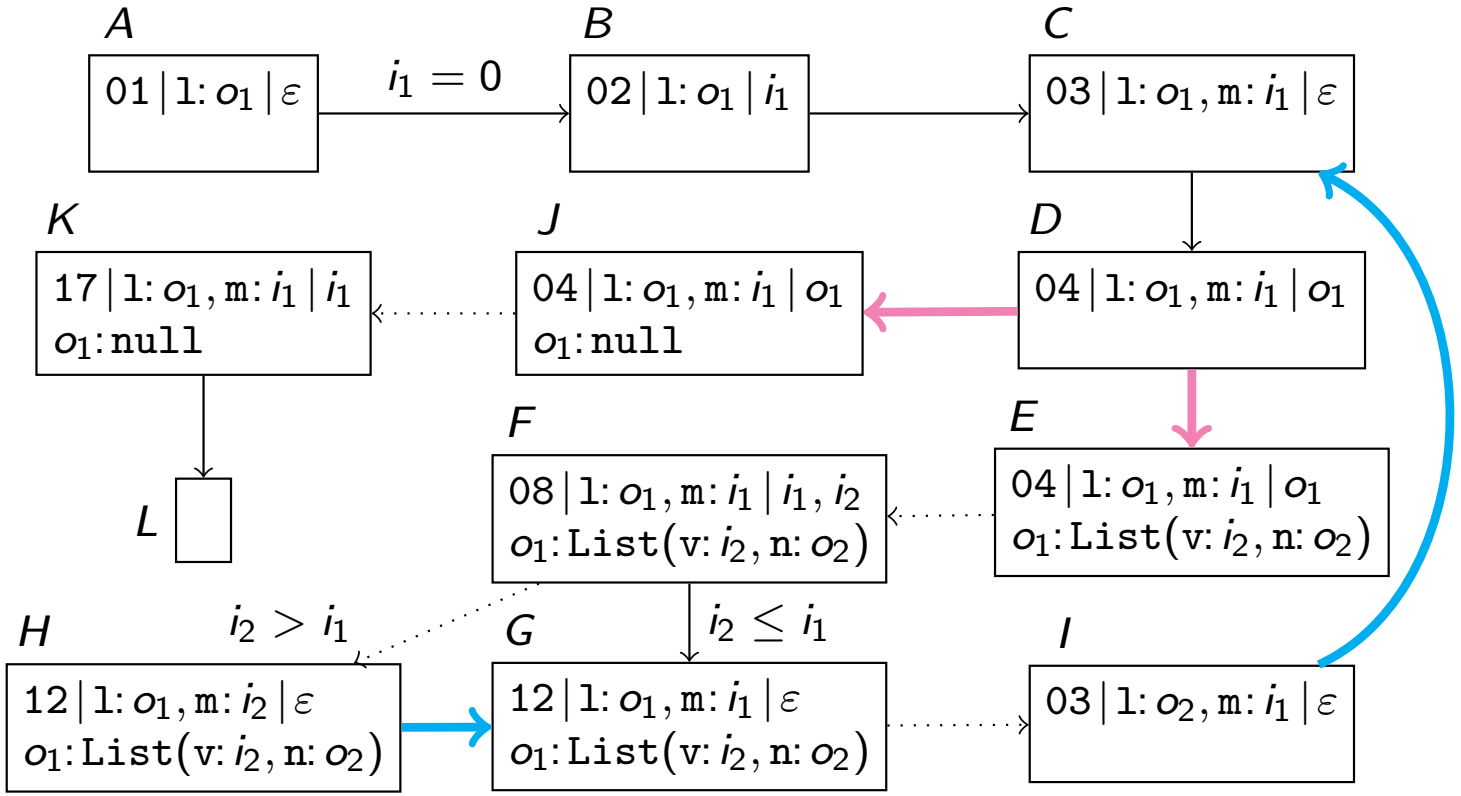
b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

$b \geq$ length of any Java evaluation starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
      ...
```



Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

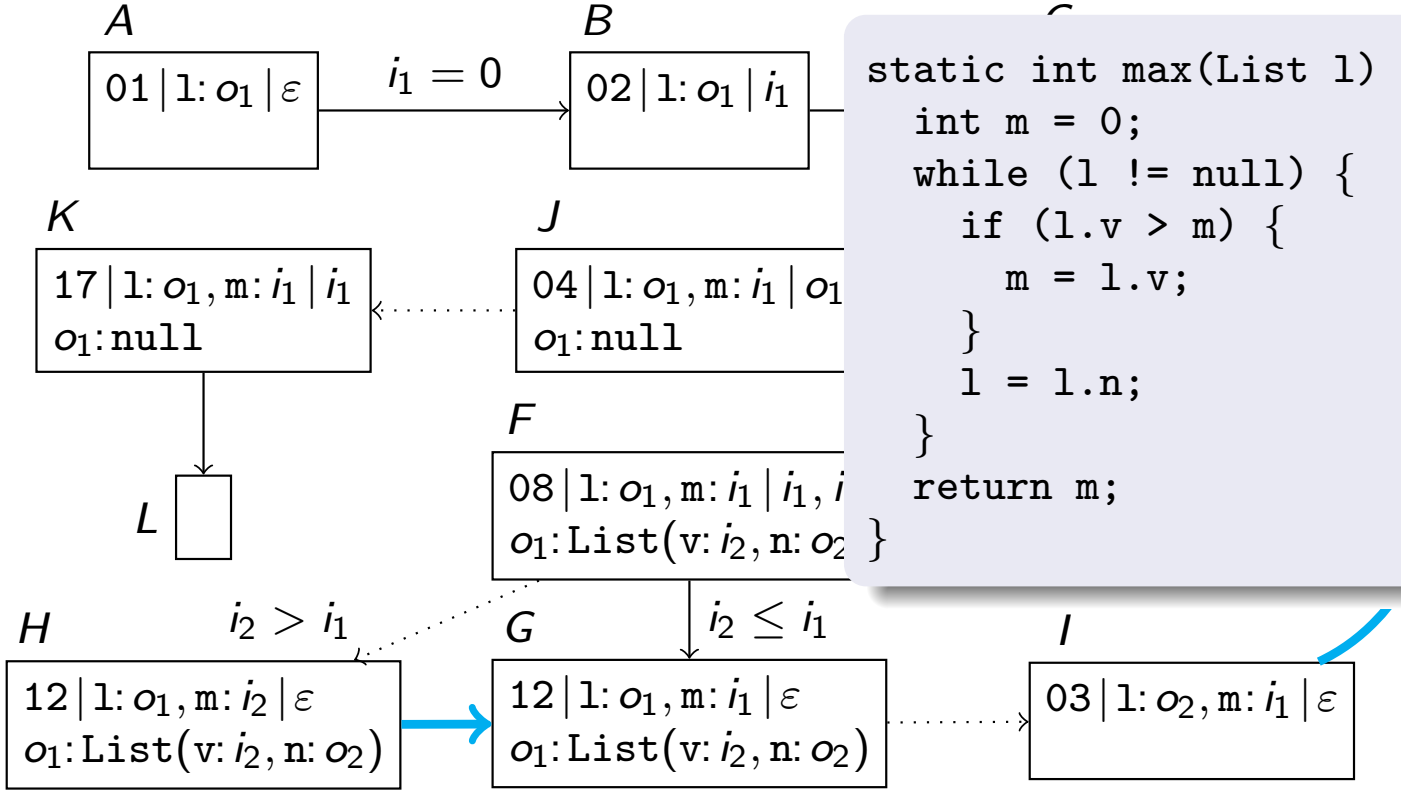
b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
02: istore_1
03: aload_0
04: ifnull 16
05: aload_0
06: getfield v
07: iload_1
08: if_icmple 12
09: aload_0
10: getfield v
11: istore_1
      ...
```



Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
      ...
```

A

$$01 \mid \mathtt{l}: o_1 \mid \varepsilon$$

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

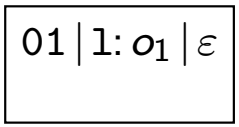
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
    ...
```

A




```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

$13 \cdot \|o_1\| + 6$ is complexity bound for State A

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
      ...
```

A

$01 \mid l{:}\, o_1 \mid \varepsilon$

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

$13 \cdot \|o_1\| + 6$ is complexity bound for State A

- for every concrete state at begin of method `max`

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

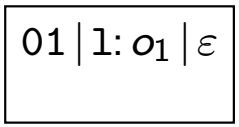
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
    ...
```

A




```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

$13 \cdot \|o_1\| + 6$ is complexity bound for State A

- for every concrete state at begin of method `max` where o_1 is a `List` of size k,

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

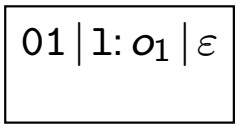
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
    ...
```

A




```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

$13 \cdot \|o_1\| + 6$ is complexity bound for State A

- for every concrete state at begin of method `max` where o_1 is a `List` of size k,
  Java evaluation has at most length $13 \cdot k + 6$

Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

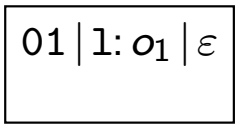
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

Complexity of Java Methods

```
01: iconst_0
    ...
```

A




```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

$13 \cdot \|o_1\| + 6$ is complexity bound for State A

- for every concrete state at begin of method `max` where o_1 is a `List` of size k,
  Java evaluation has at most length $13 \cdot k + 6$
- $13 \cdot \|\texttt{l}\| + 6$ is upper bound for runtime of `max`

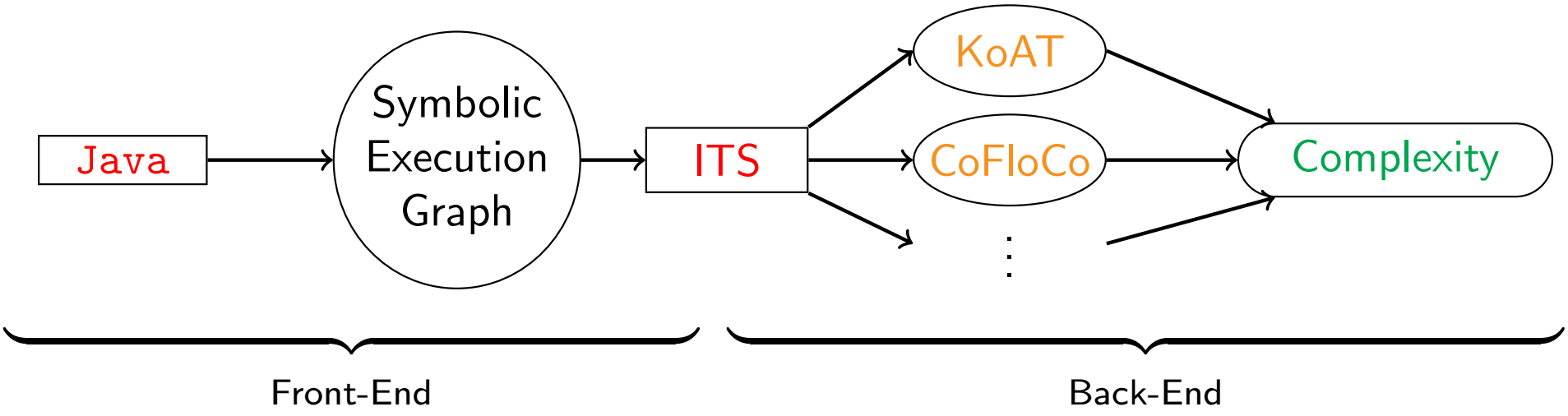
Complexity Bound b for abstract state s
with symb. integers $i_1, \ldots, i_n$ and symb. references $o_1, \ldots, o_m$

b is arithmetic term with variables $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$ such that:

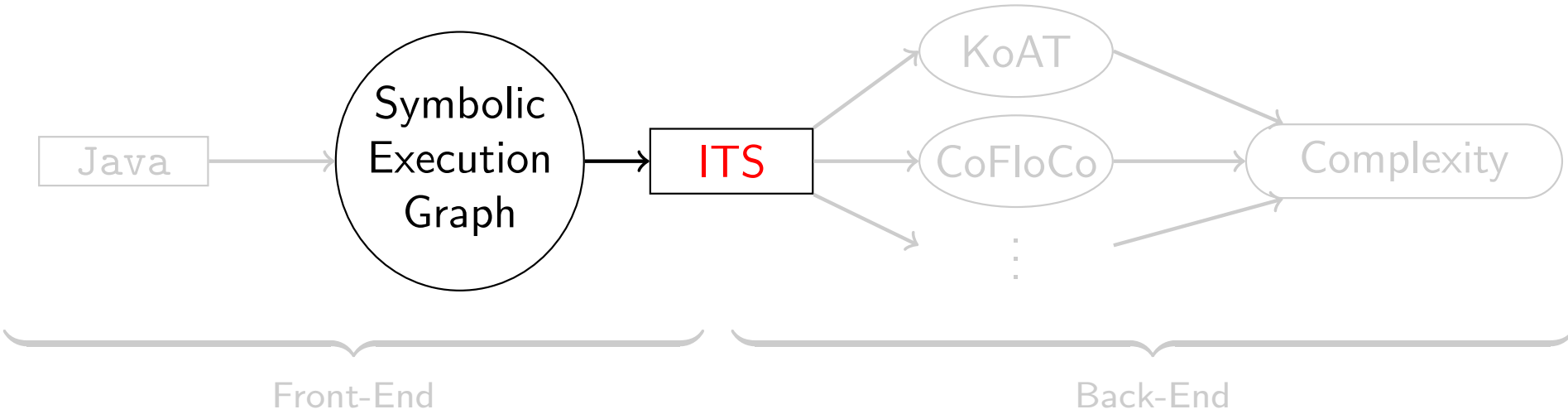
$b \geq$ length of any Java evaluation
starting with corresponding concrete state represented by s

for all instantiations of $i_1, \ldots, i_n, \|o_1\|, \ldots, \|o_m\|$

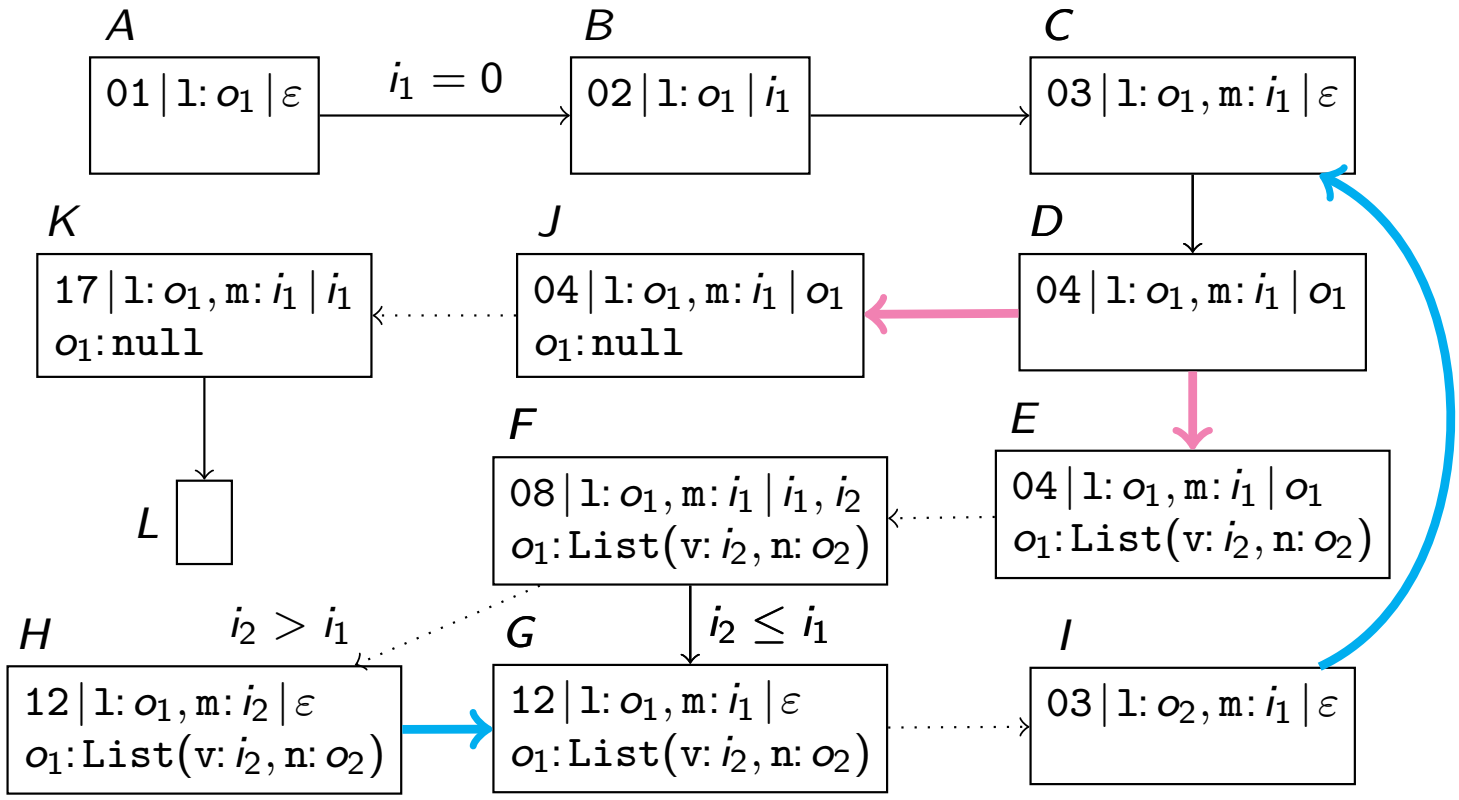
AProVE for Complexity Analysis of Java



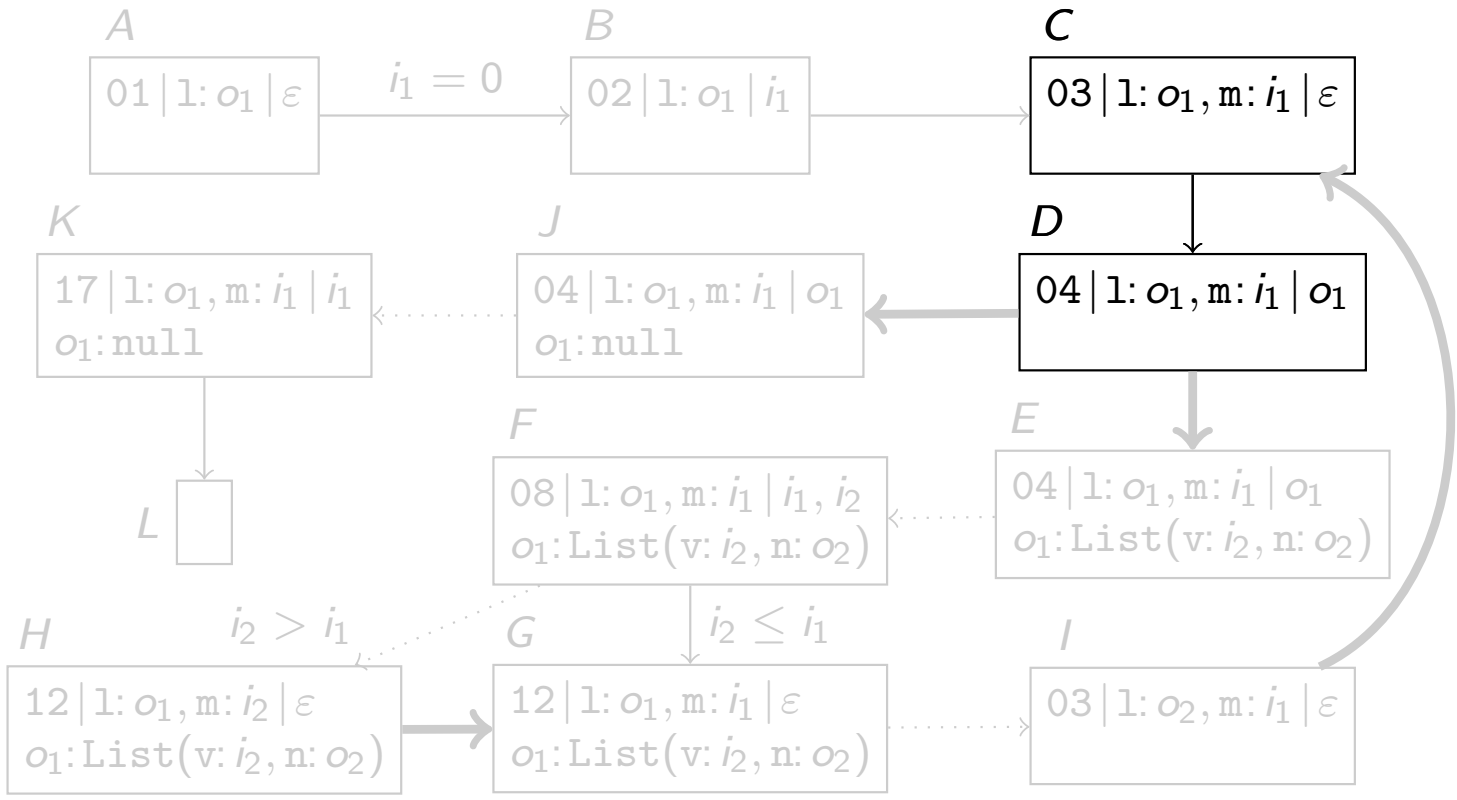
AProVE for Complexity Analysis of Java



Transform Evaluation Edges



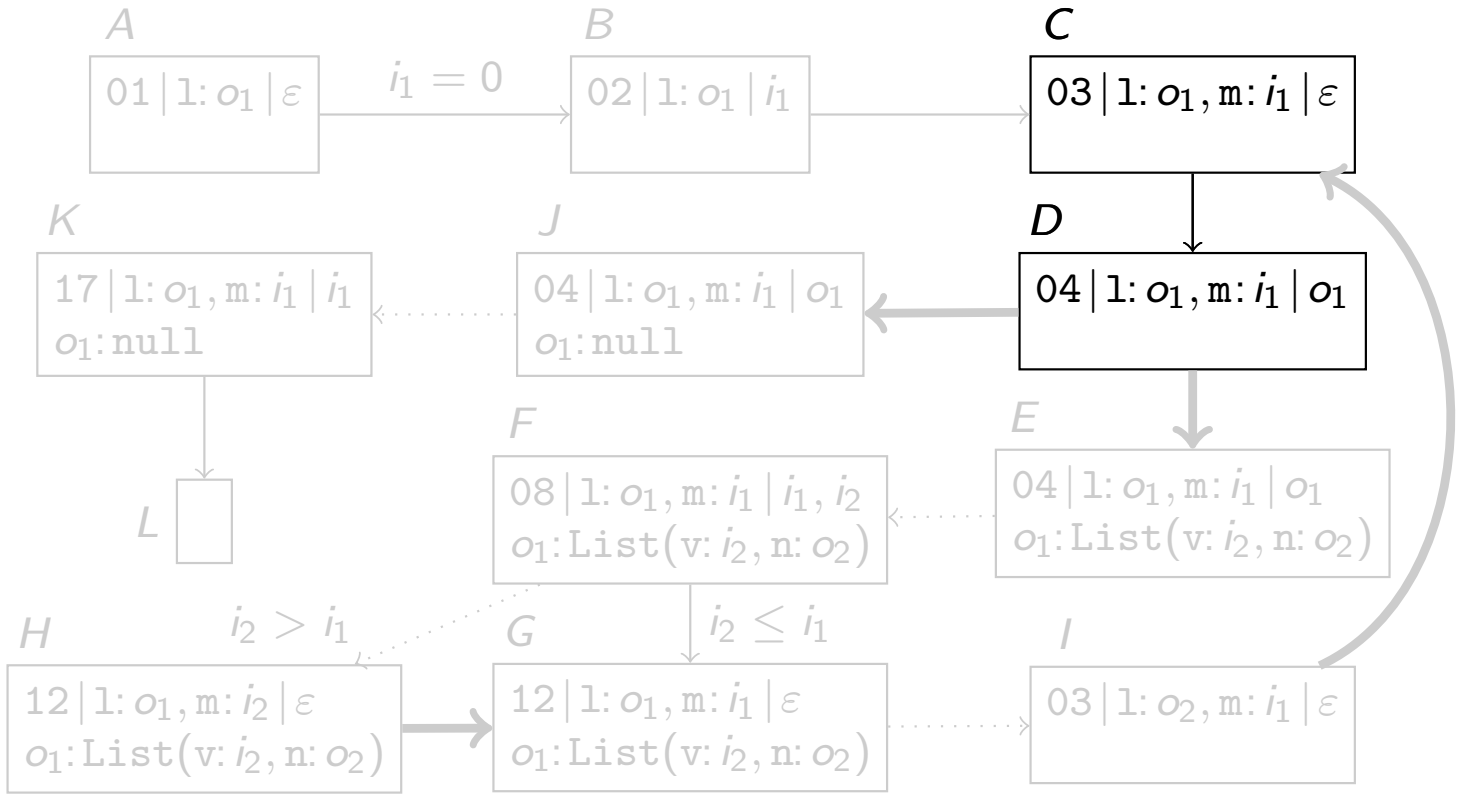
Transform Evaluation Edges



Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

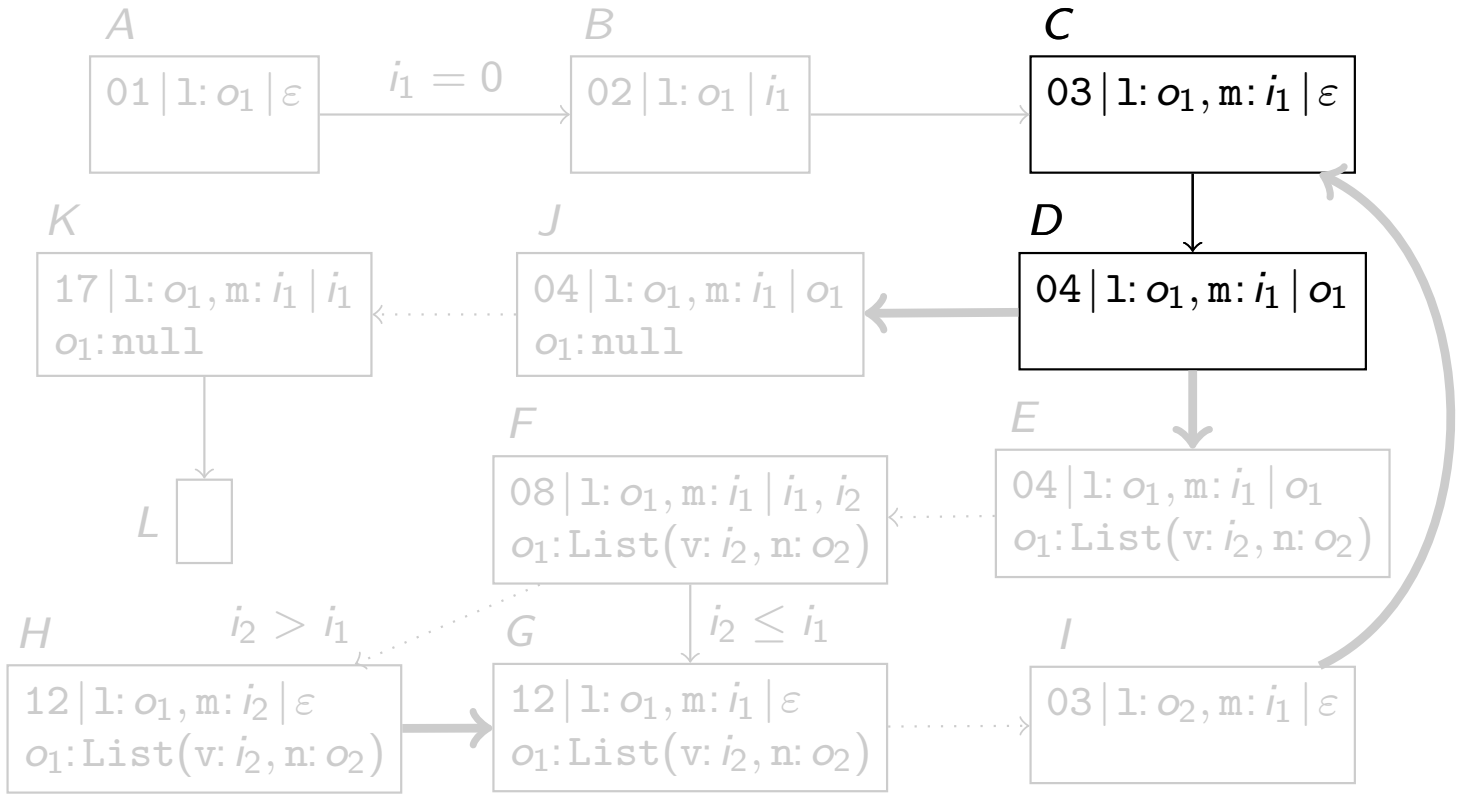


Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$

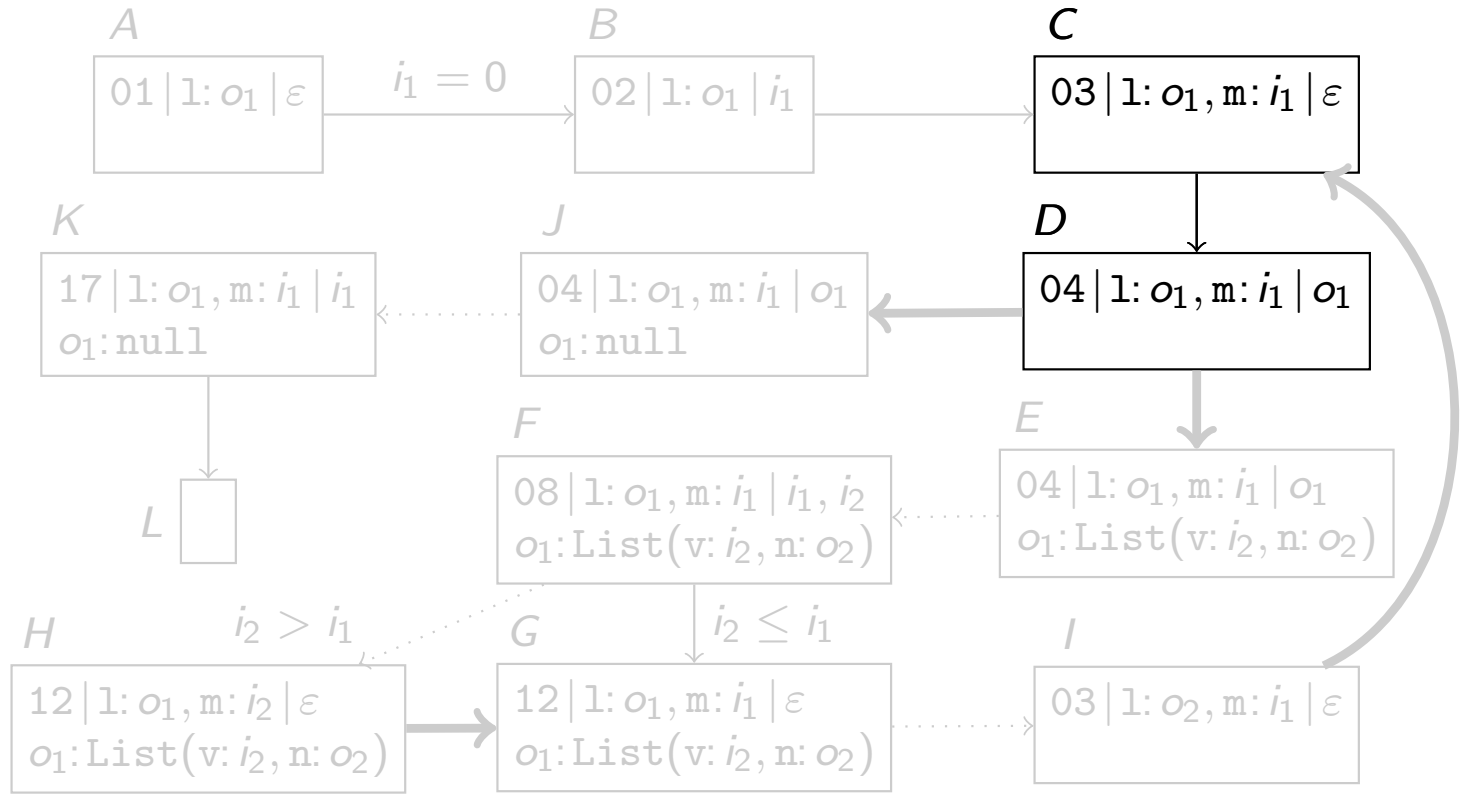


Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$



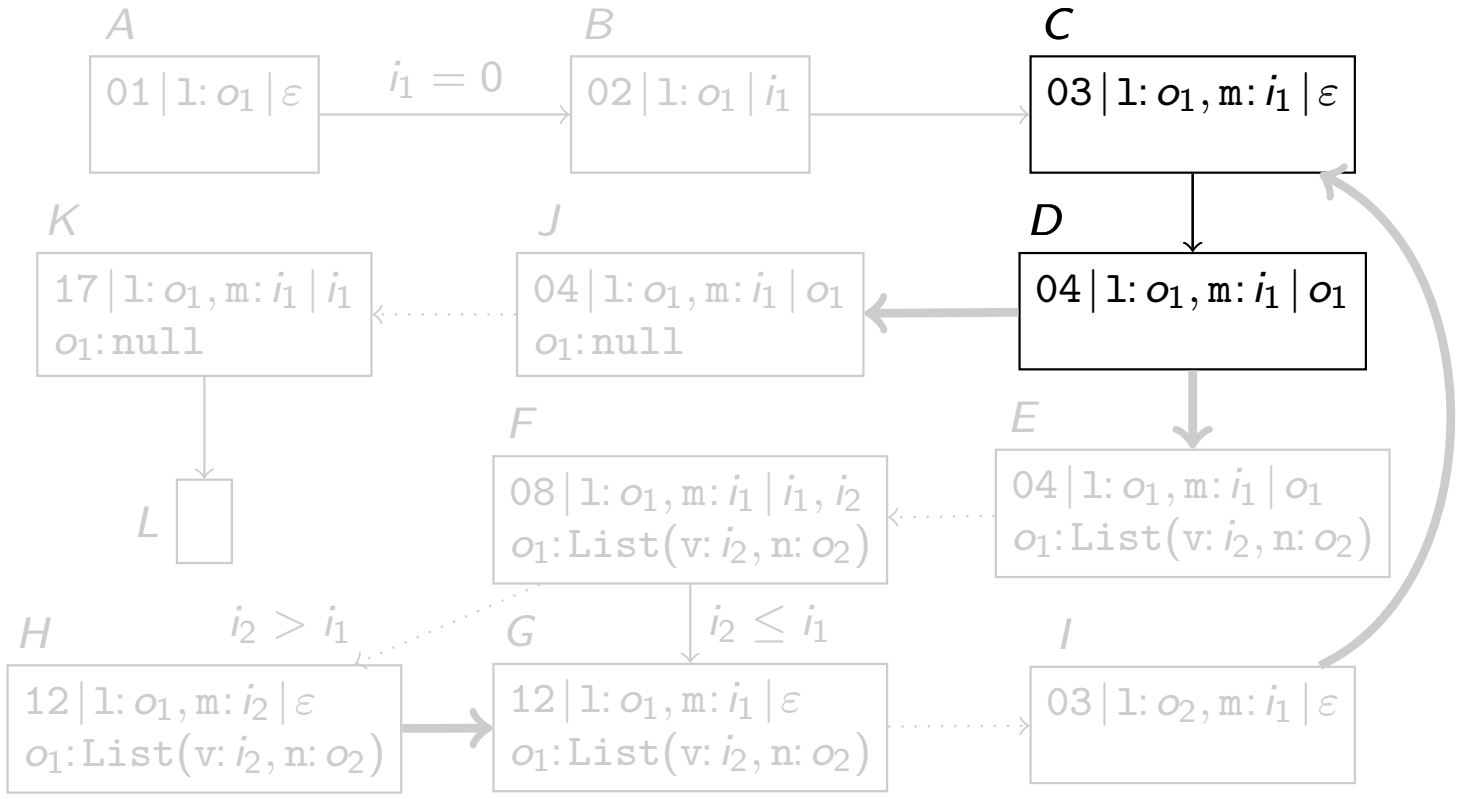
$C \longrightarrow D$

Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

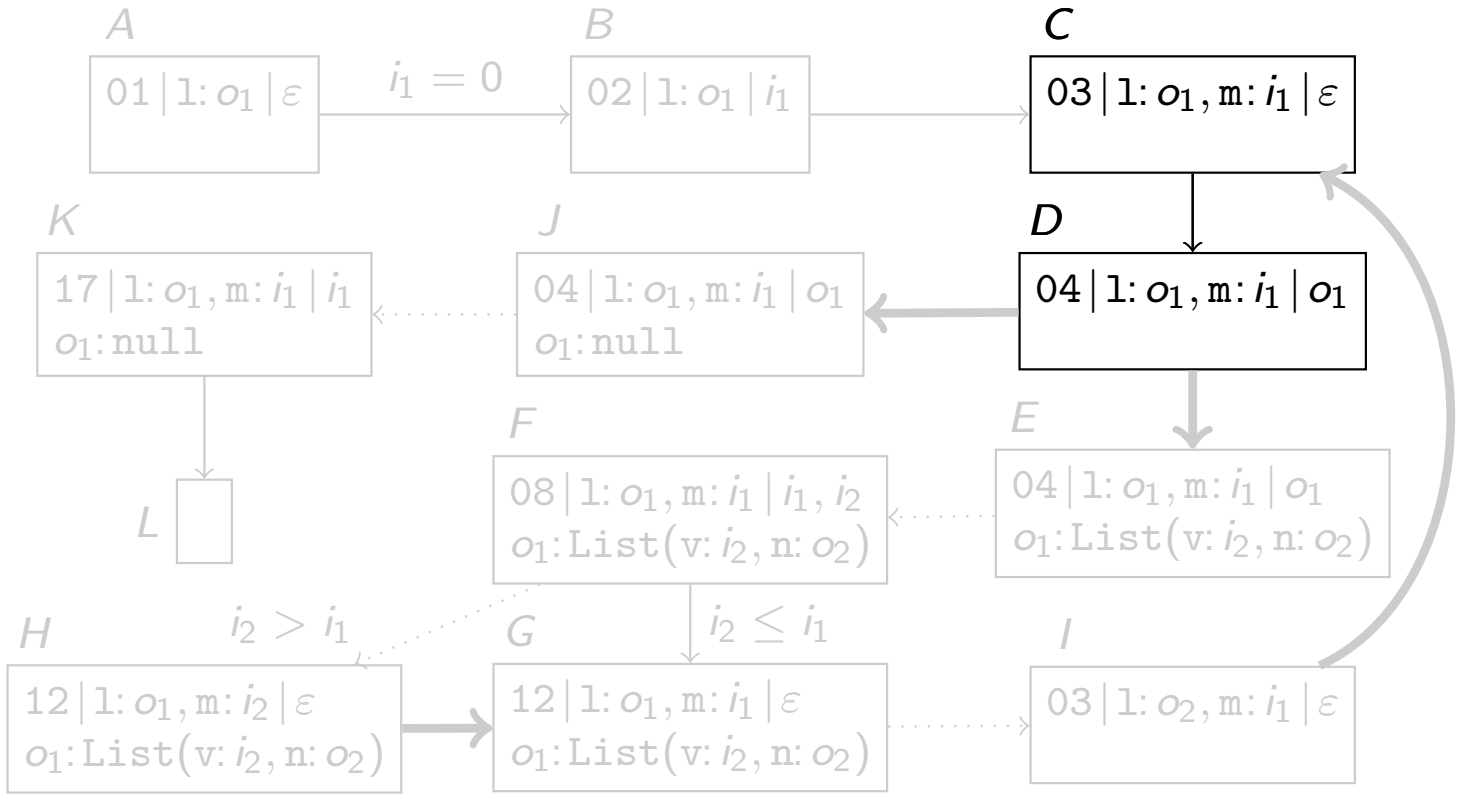
i_1', $\|o_1\|'$



$C \xrightarrow{1} D$

weight: 1

Transform Evaluation Edges



ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$

$C \xrightarrow{1} D$

weight: 1

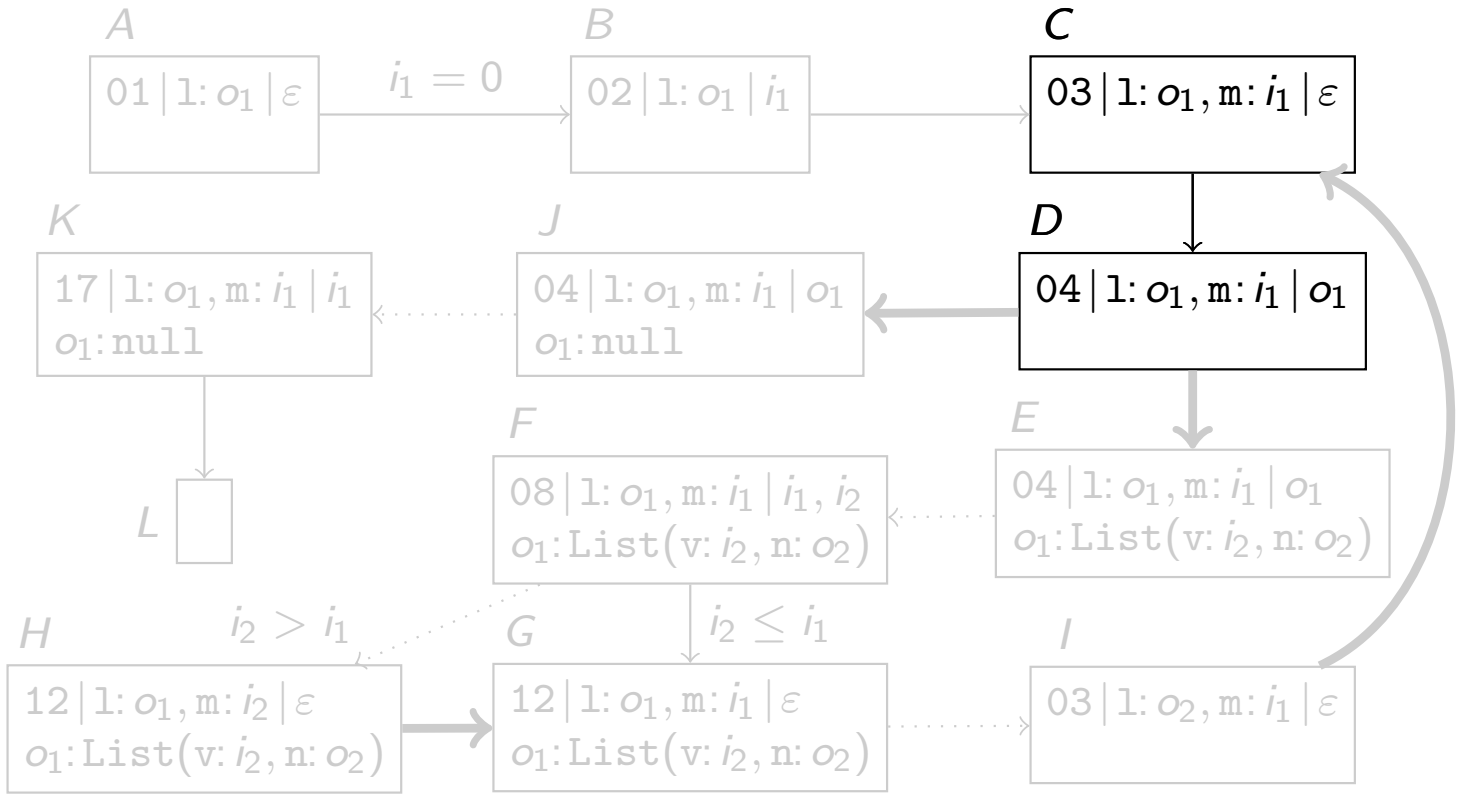
condition: $\|o\| = 0$ if $o : \texttt{null}$

Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$



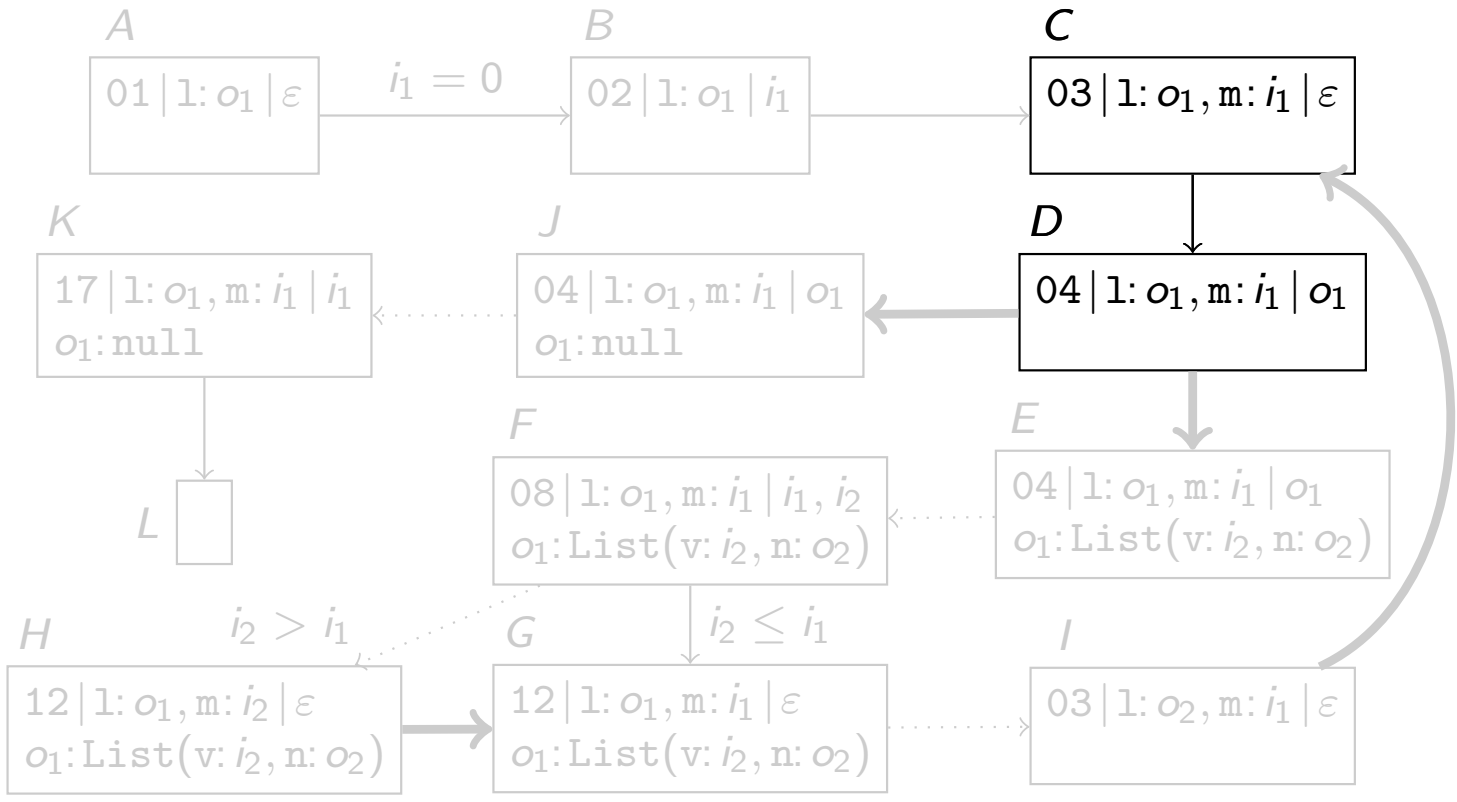
$C \xrightarrow{1} D$

weight: 1

condition: $\|o\| = 0$ if $o : \texttt{null}$

$\|o\| \geq 1$ if $o : \texttt{List}(\ldots)$

Transform Evaluation Edges



ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$

$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0$

weight: 1

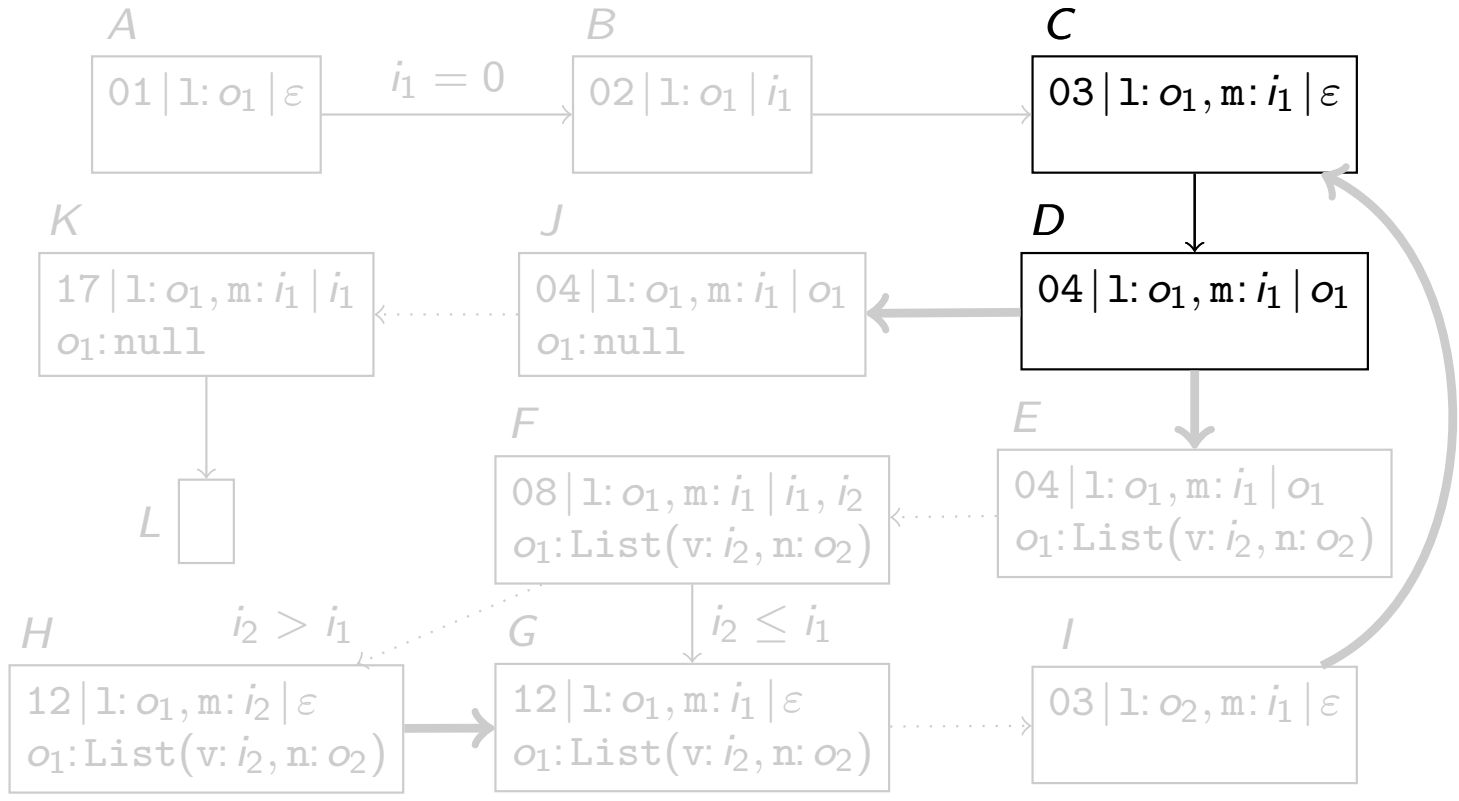
condition:

$\|o\| = 0$ if $o : \texttt{null}$

$\|o\| \geq 1$ if $o : \texttt{List}(\ldots)$

$\|o\| \geq 0$ otherwise

Transform Evaluation Edges



ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$

$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0 \wedge \|o_1\|' = \|o_1\| \wedge i_1' = i_1$

weight: 1

condition:

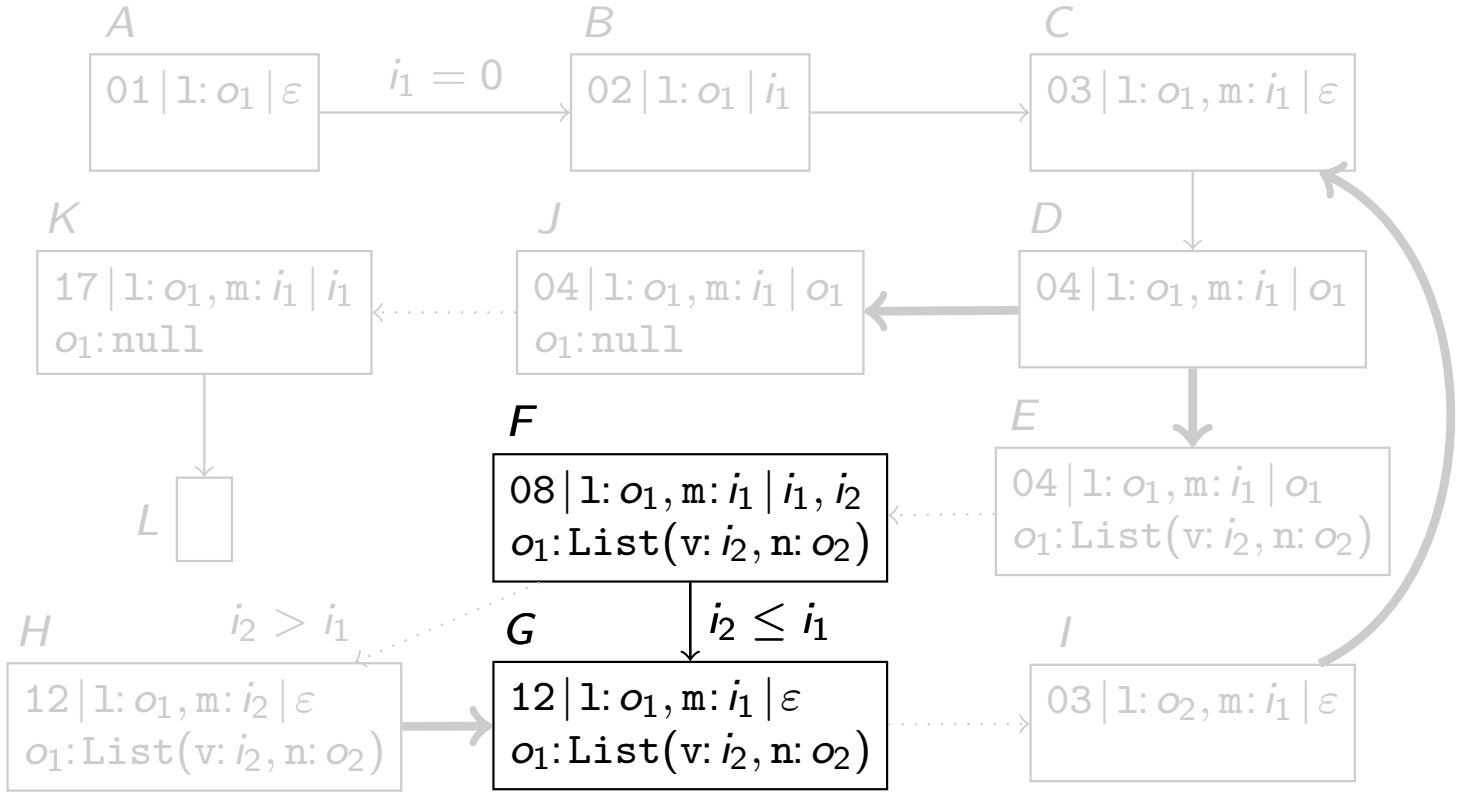
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Evaluation Edges

ITS over variables

i_1, $\|o_1\|$

i_1', $\|o_1\|'$



$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0 \wedge \|o_1\|' = \|o_1\| \wedge i_1' = i_1$

weight: 1

condition:

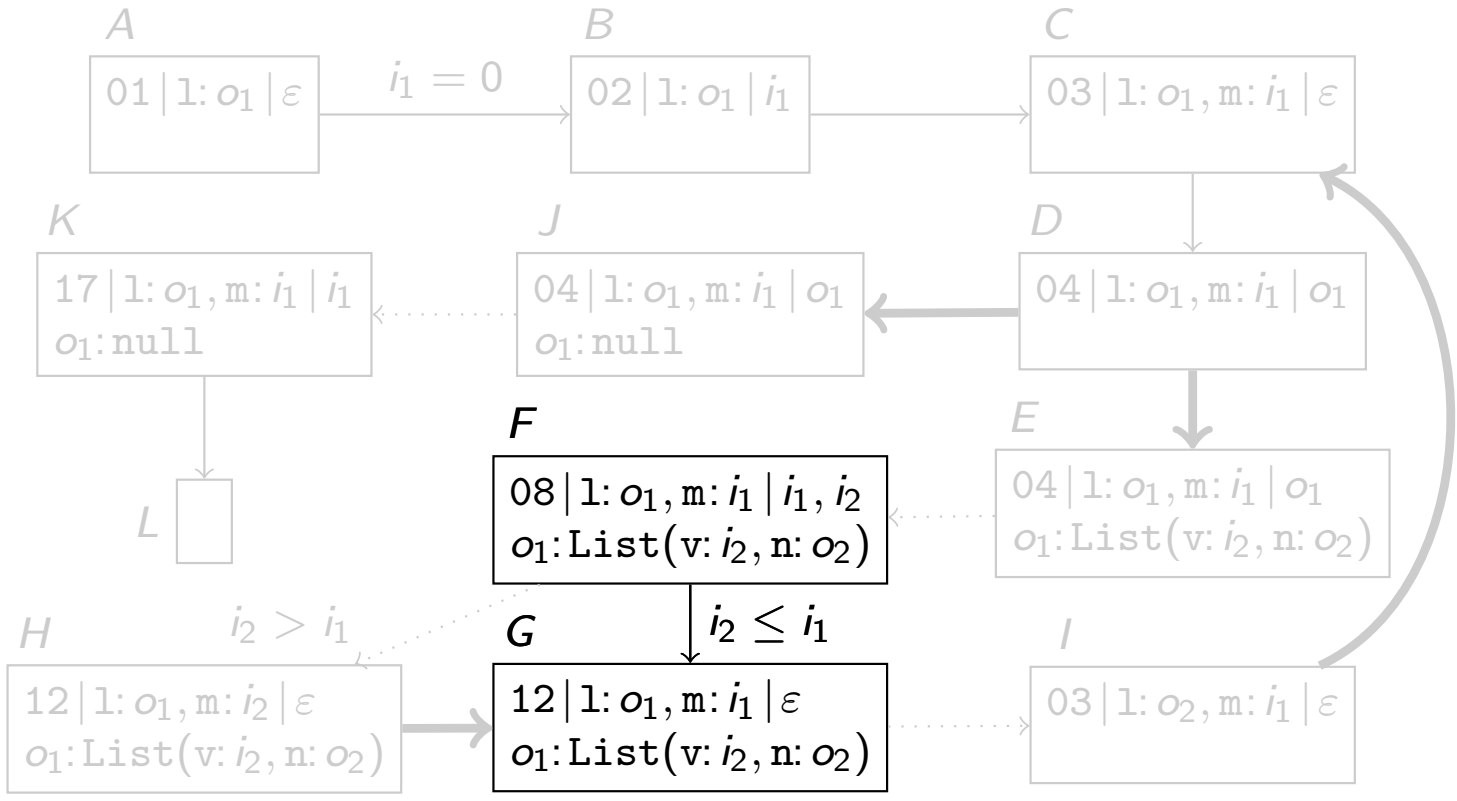
$\|o\| = 0$	if	o : `null`
$\|o\| \geq 1$	if	o : `List(...)`
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Evaluation Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

weight: 1

condition:

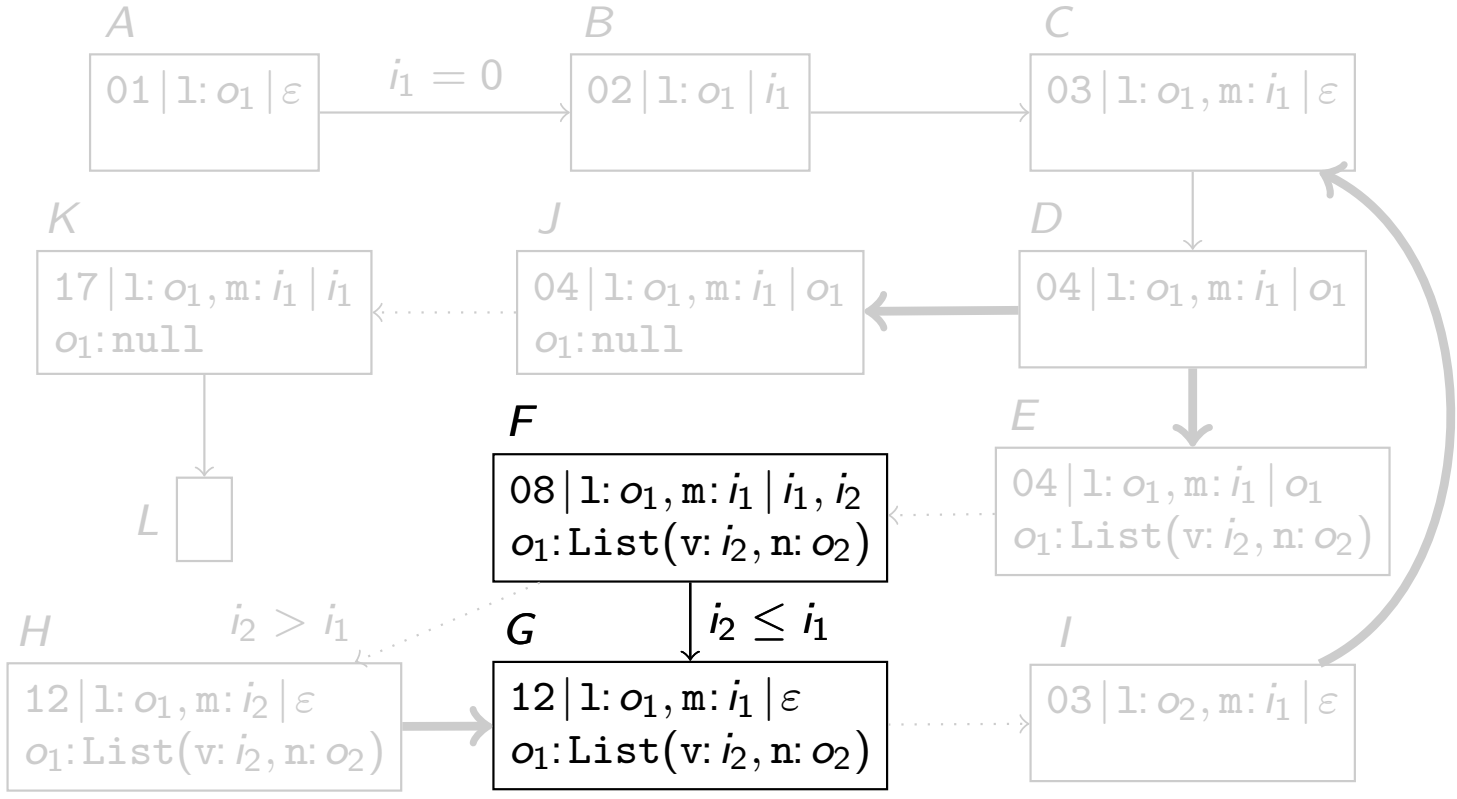
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Evaluation Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

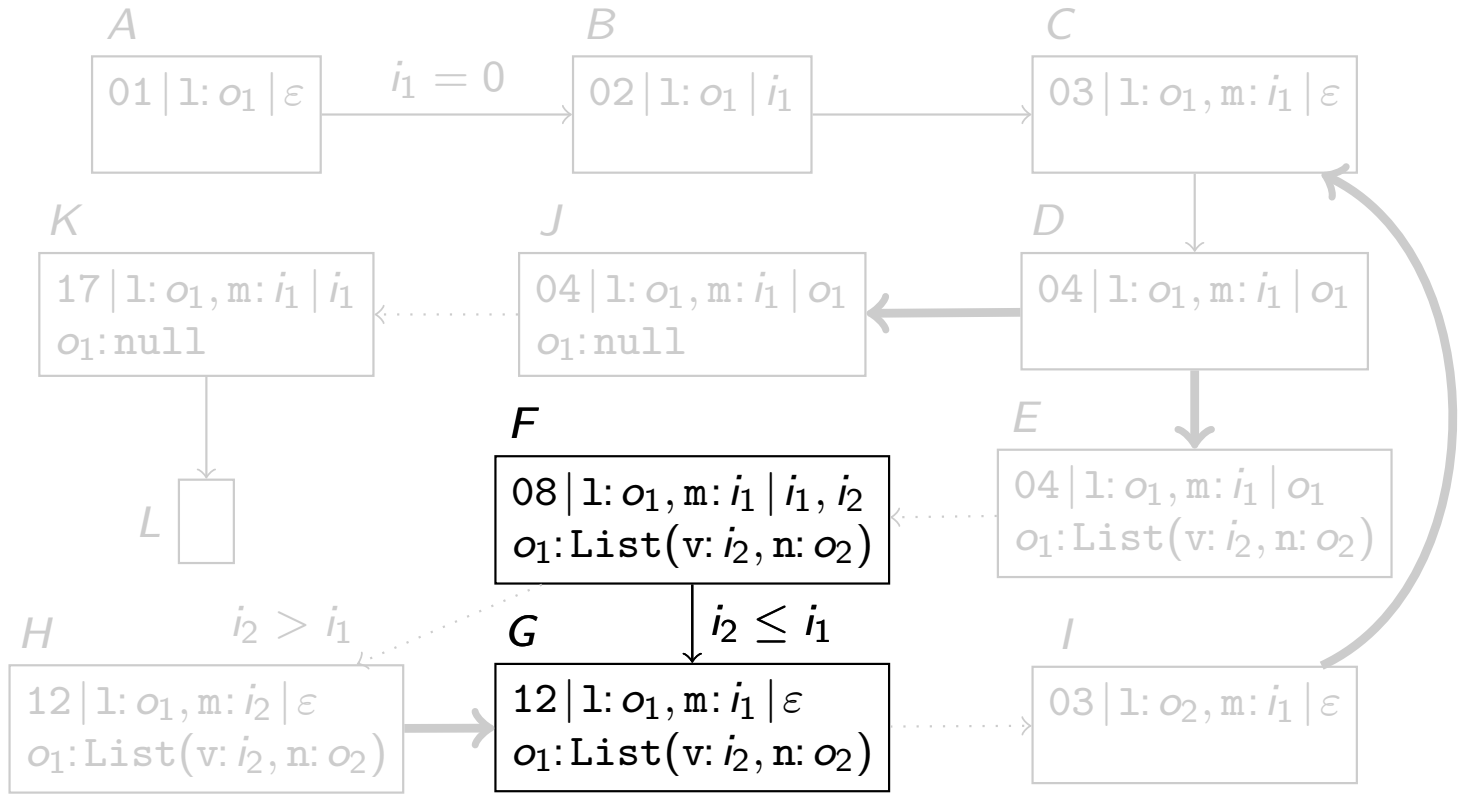
$F \xrightarrow{1} G$

weight: 1

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Evaluation Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$C \xrightarrow{1} D$ **if** $\|o_1\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

$F \xrightarrow{1} G$ **if** $i_2 \leq i_1$

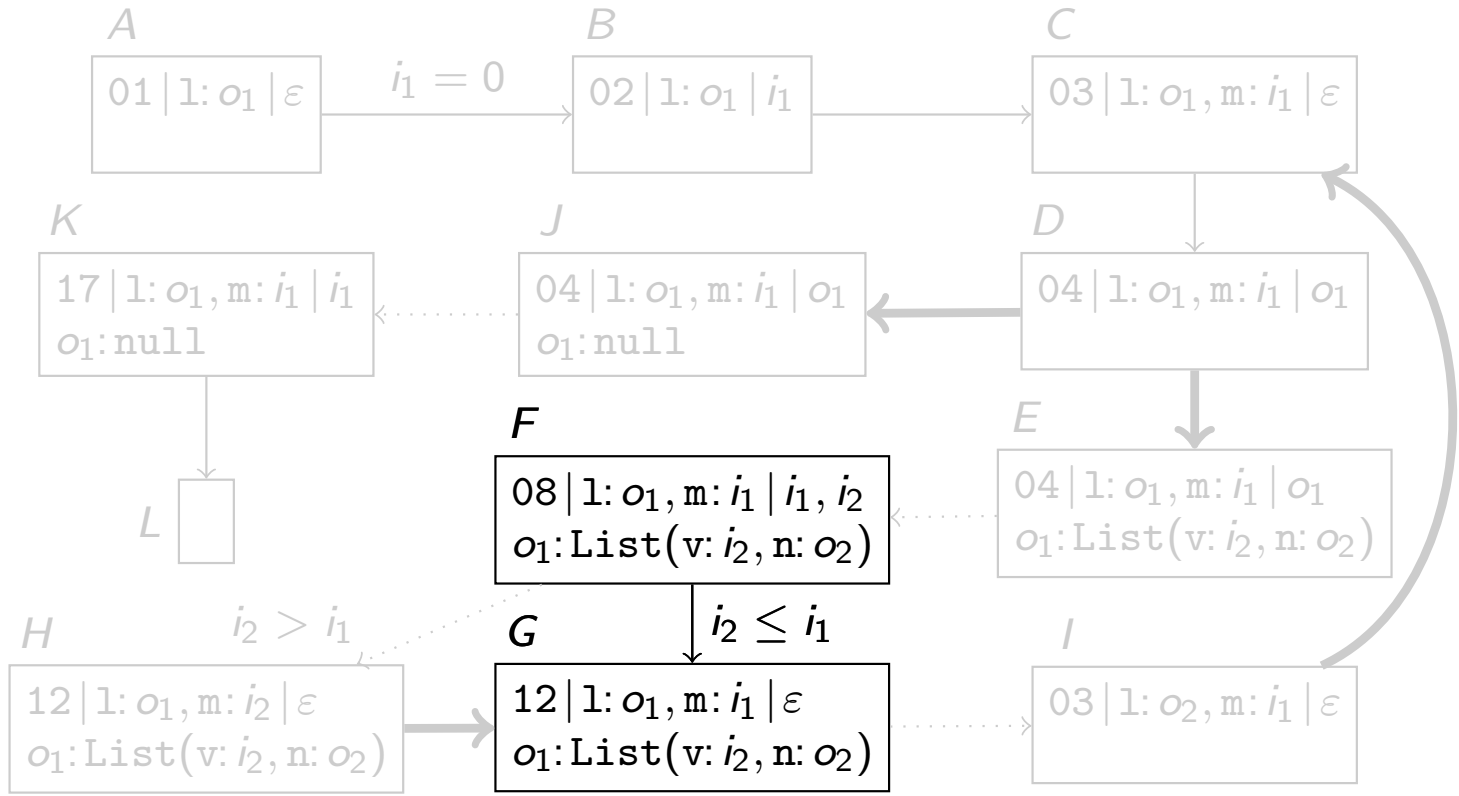
weight: 1

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

condition of the edge in the symbolic execution graph

Transform Evaluation Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$C \xrightarrow{1} D \quad \textbf{if} \quad \|o_1\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

$$F \xrightarrow{1} G \quad \textbf{if} \quad i_2 \leq i_1 \ \wedge\ \|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0$$

weight: 1

condition:

$\|o\| = 0$ if $o : \texttt{null}$

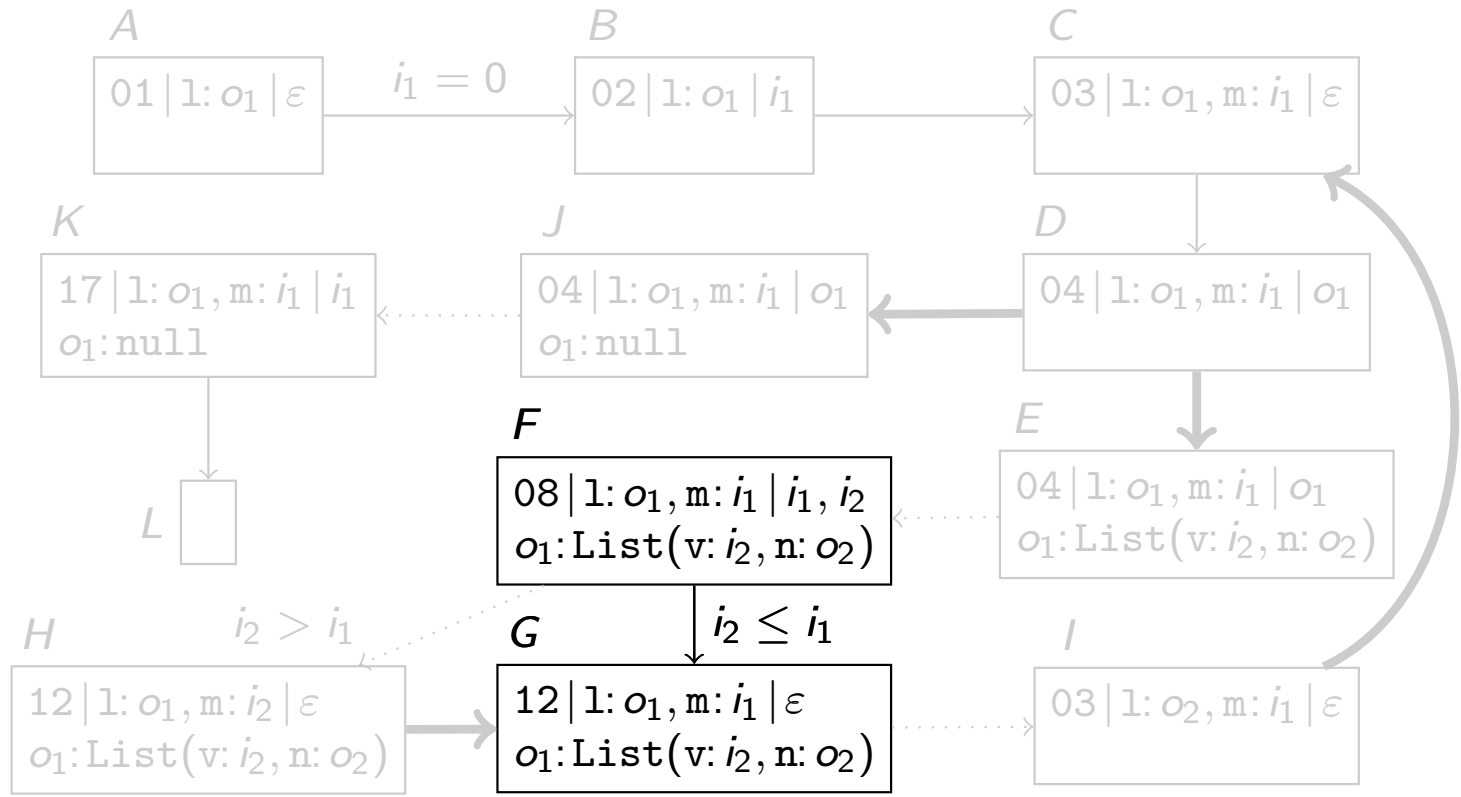
$\|o\| \geq 1$ if $o : \texttt{List}(\ldots)$

$\|o\| \geq 0$ otherwise

$x' = x$ for all variables x

condition of the edge in the symbolic execution graph

Transform Evaluation Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$C \xrightarrow{1} D \quad \textbf{if} \quad \|o_1\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

$$F \xrightarrow{1} G \quad \textbf{if} \quad i_2 \leq i_1 \ \wedge\ \|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ \|o_2\|' = \|o_2\| \ldots$$

weight: 1

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

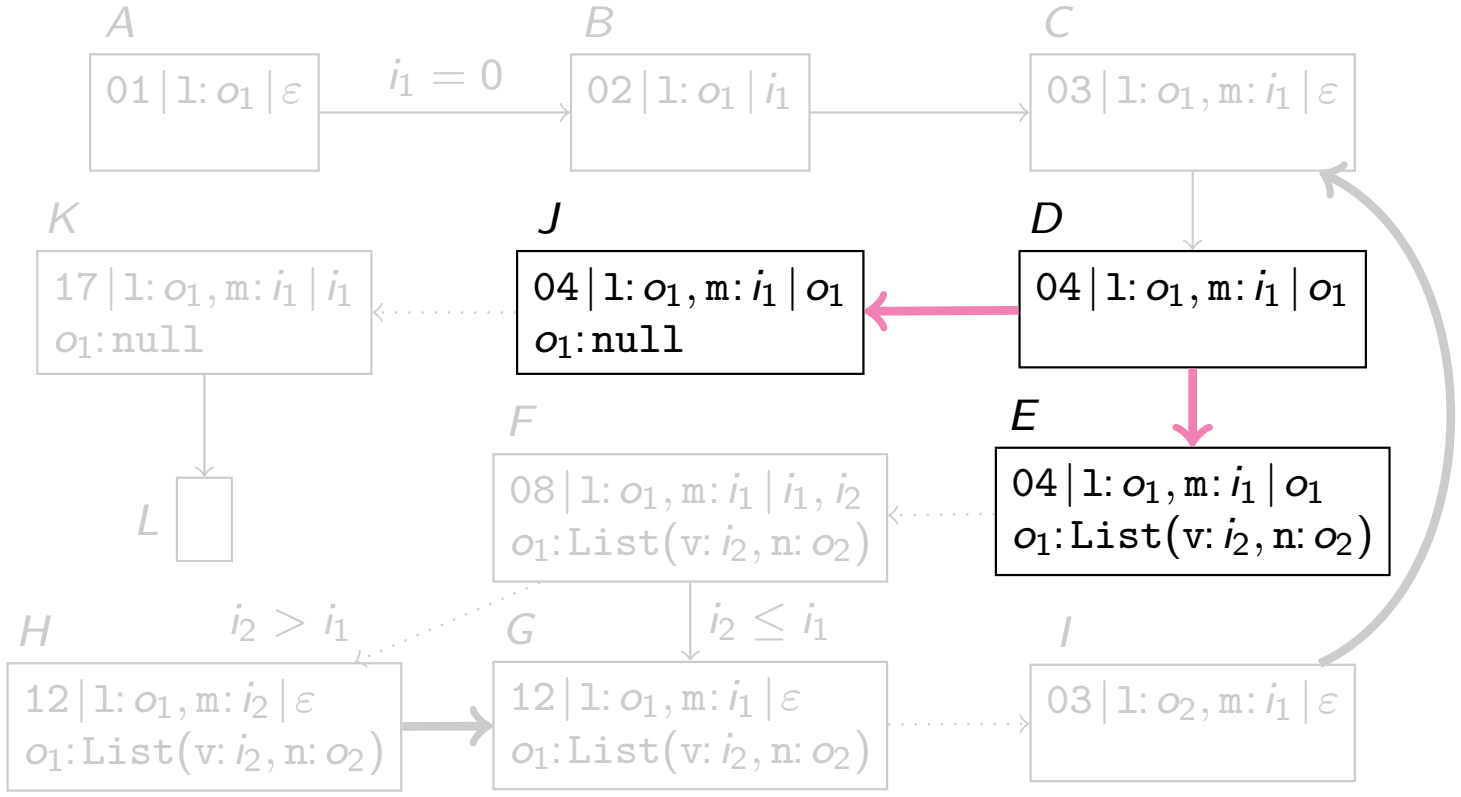
condition of the edge in the symbolic execution graph

Transform Refinement Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



weight: 1

condition:

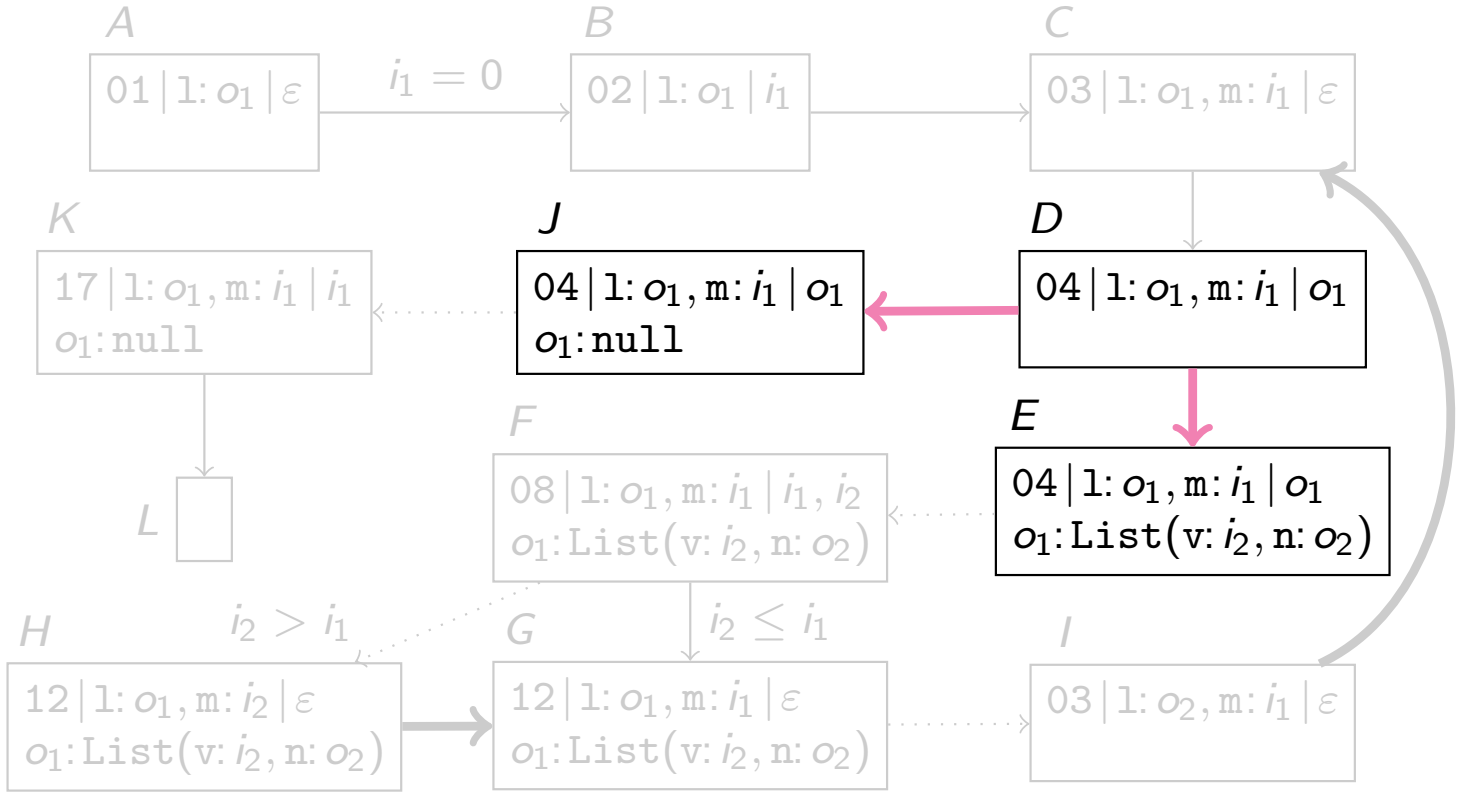
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



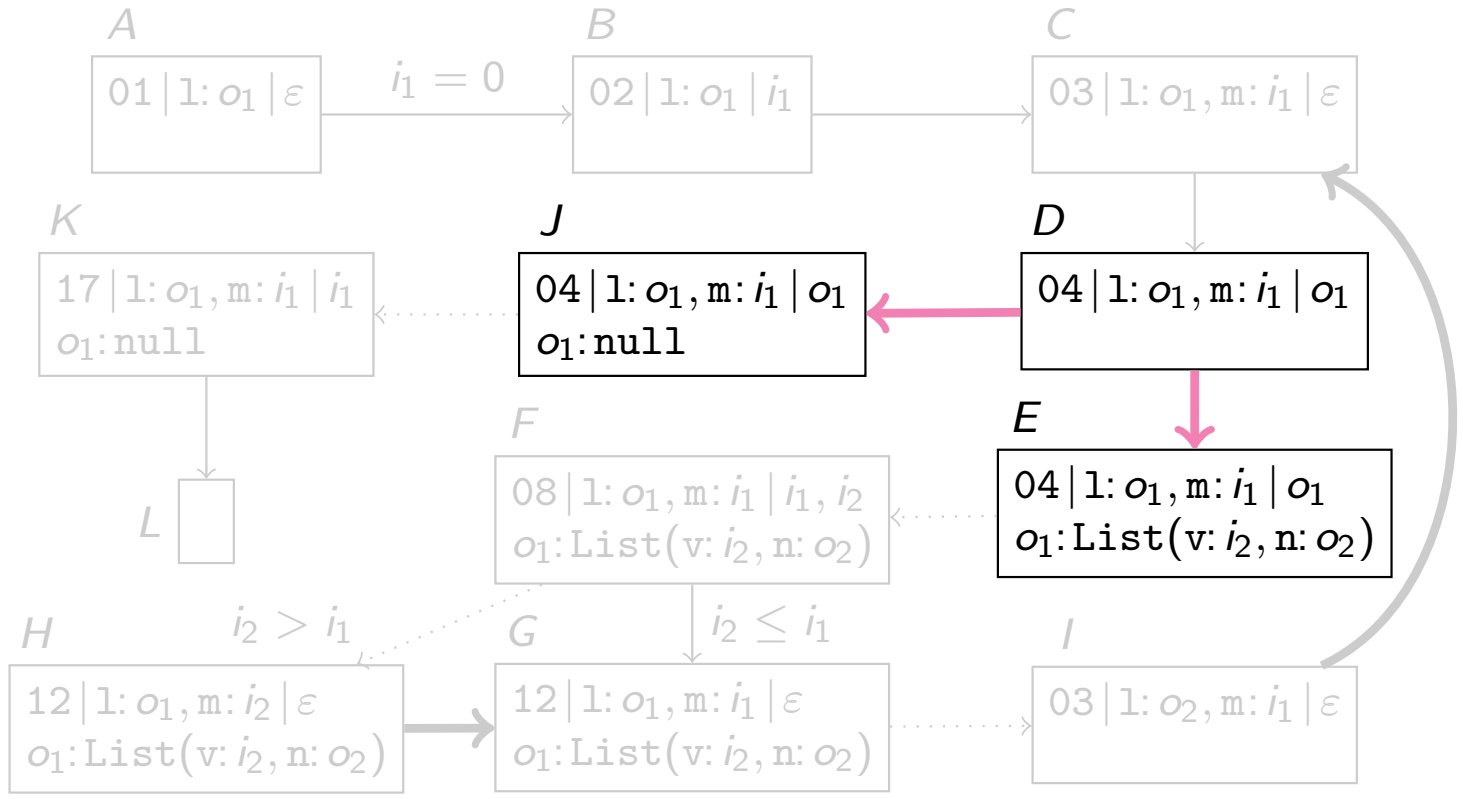
$D \xrightarrow{0} J$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$D \xrightarrow{0} J$ **if** $\|o_1\| = 0$

weight: 0

condition:

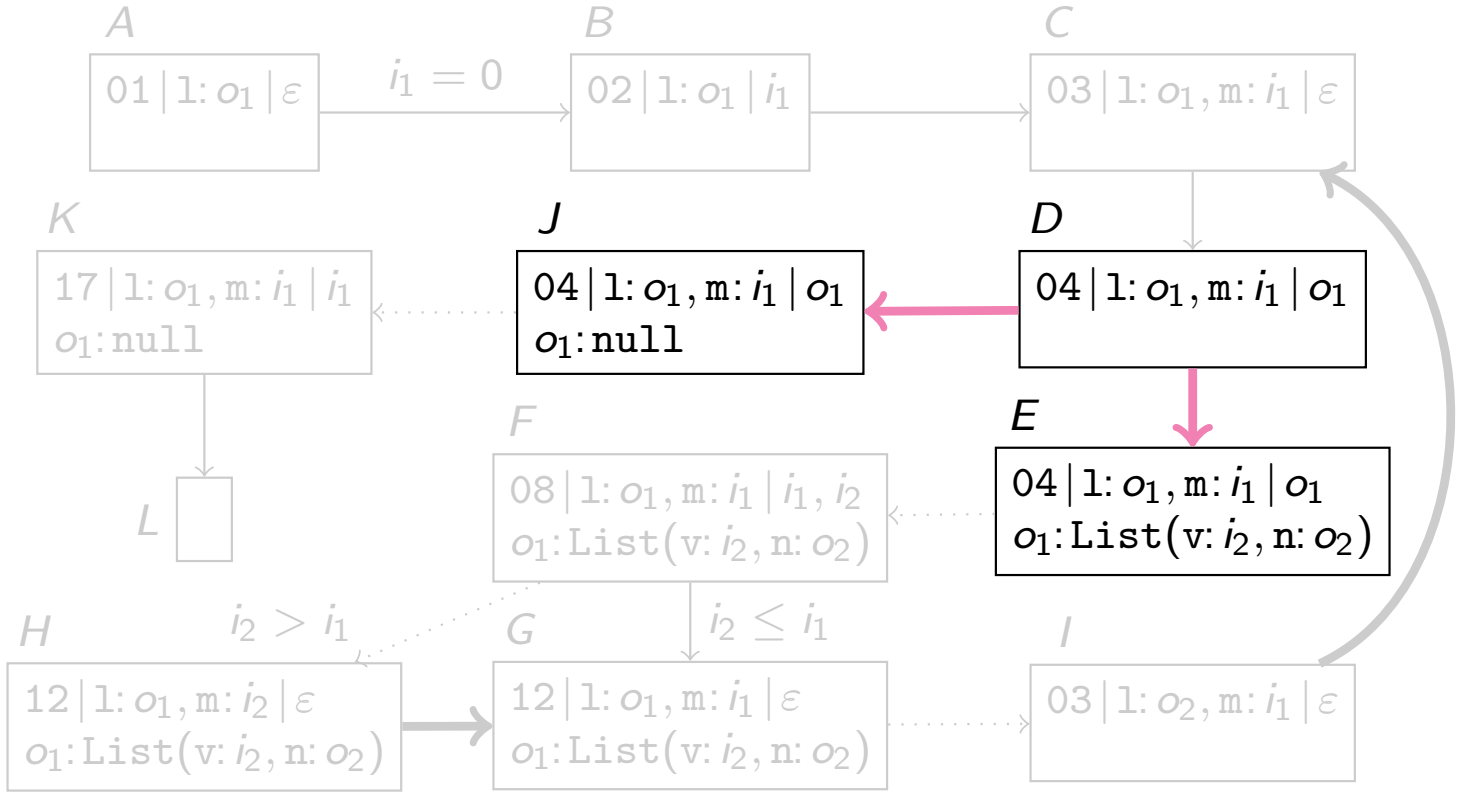
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$D \xrightarrow{0} J$ **if** $\|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

weight: 0

condition:

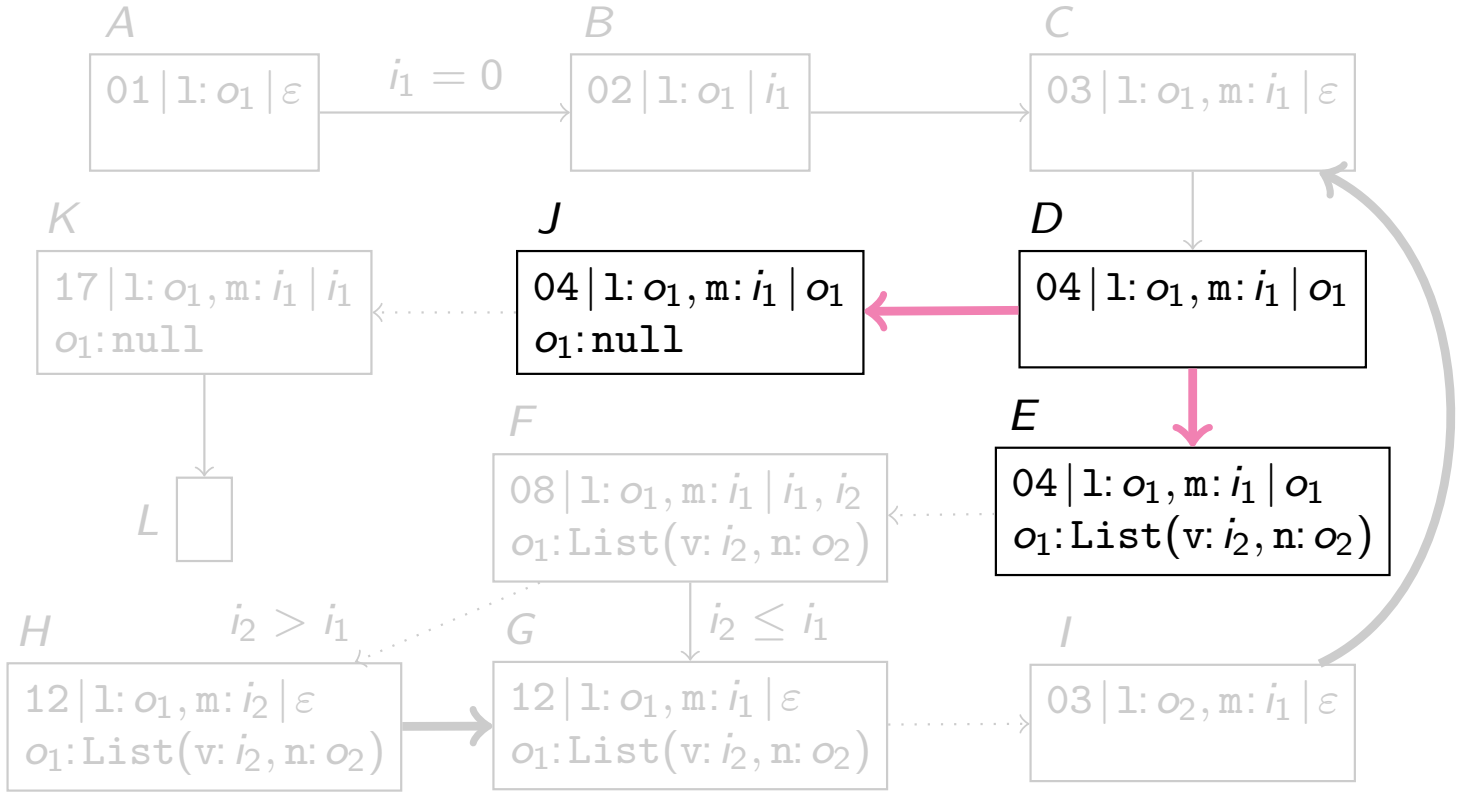
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$D \xrightarrow{0} J$ **if** $\|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

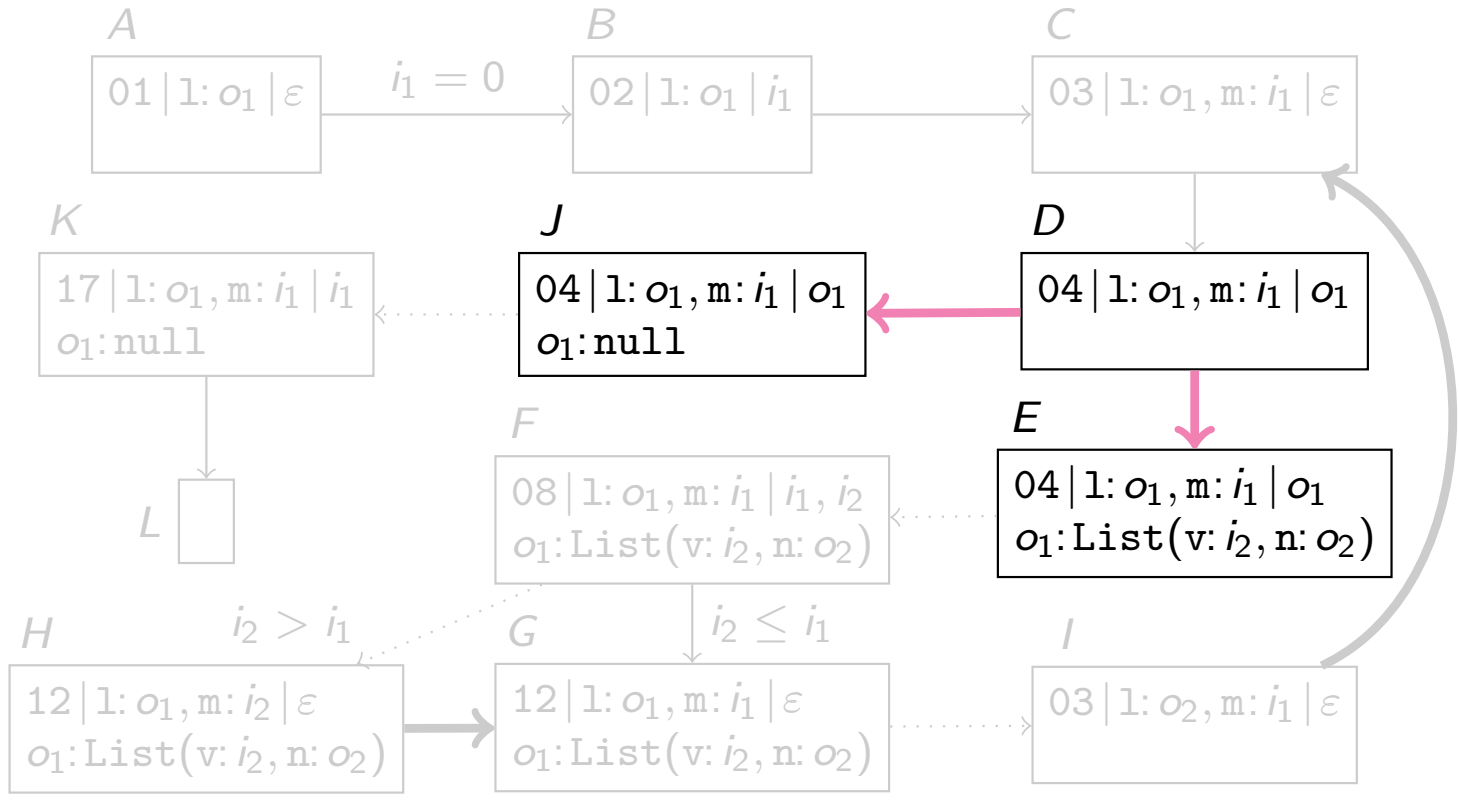
$D \xrightarrow{0} E$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \ge 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \ge 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$D \xrightarrow{0} J \quad \textbf{if} \quad \|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

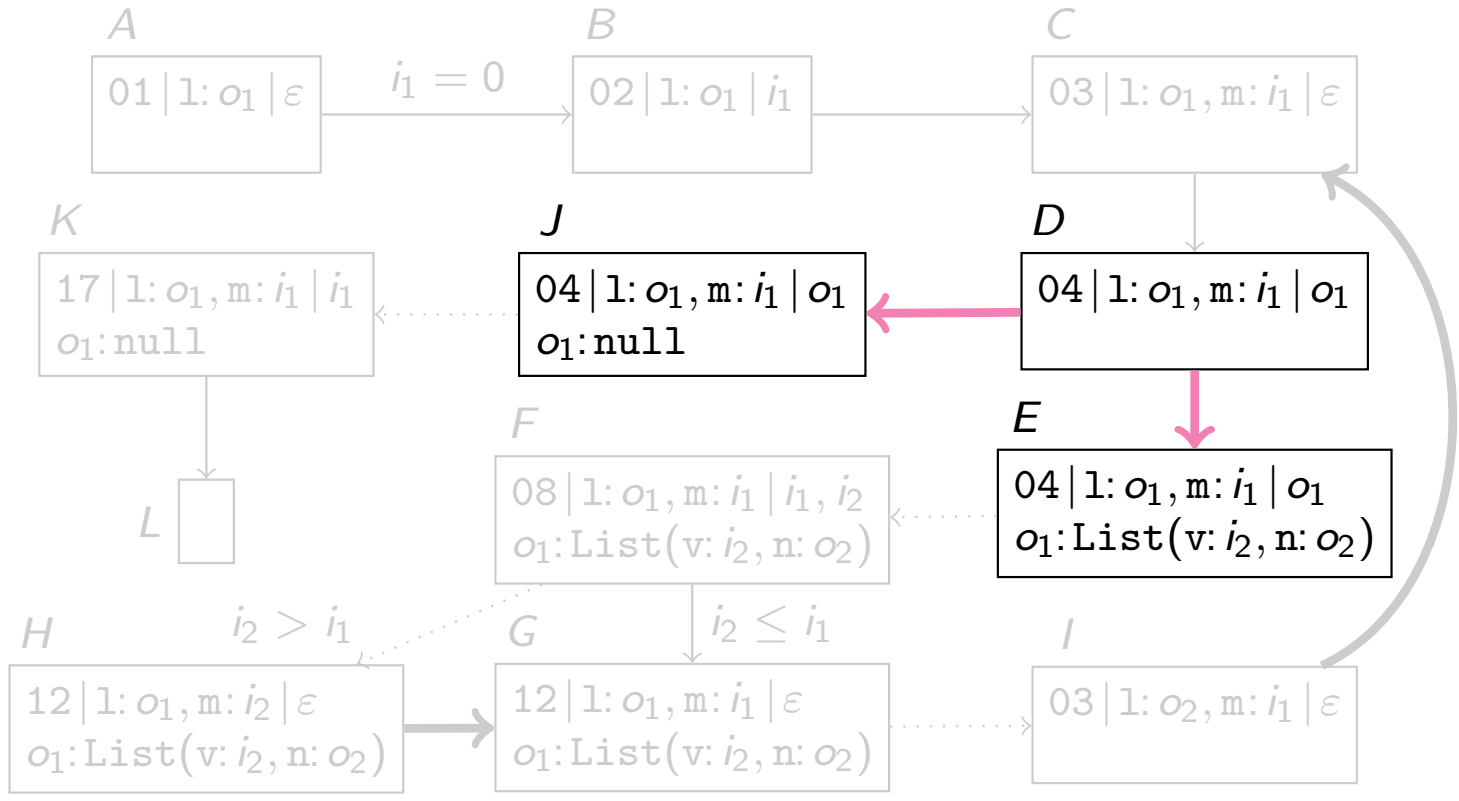
$$D \xrightarrow{0} E \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0$$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$D \xrightarrow{0} J \quad \textbf{if} \quad \|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

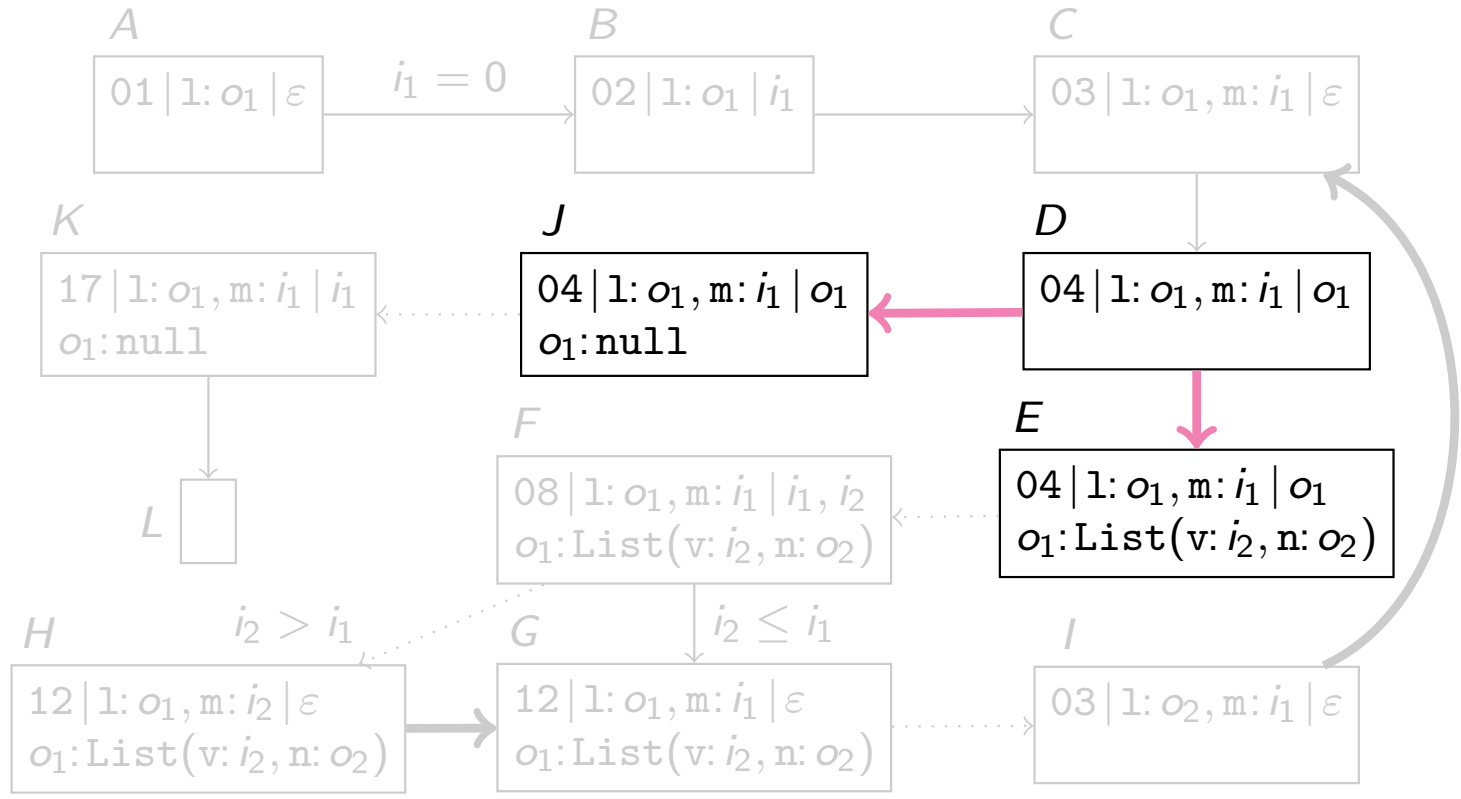
$$D \xrightarrow{0} E \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$D \xrightarrow{0} J \quad \textbf{if} \quad \|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

$$D \xrightarrow{0} E \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

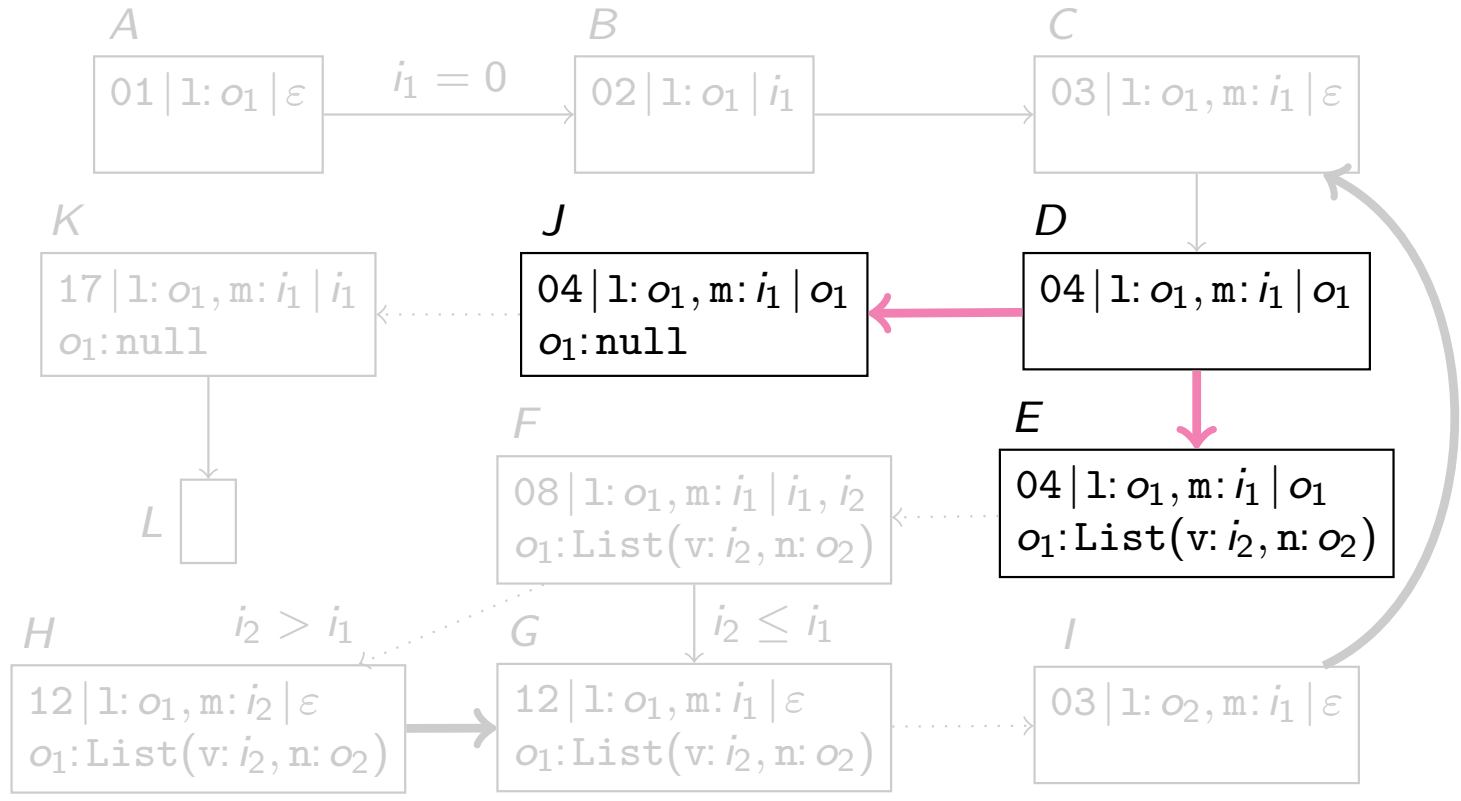
weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

$\|o_1\| \geq \|o_2\|'$ if o_1 is refined and o_1 reaches o_2

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$D \xrightarrow{0} J \quad \textbf{if} \quad \|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

$$D \xrightarrow{0} E \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

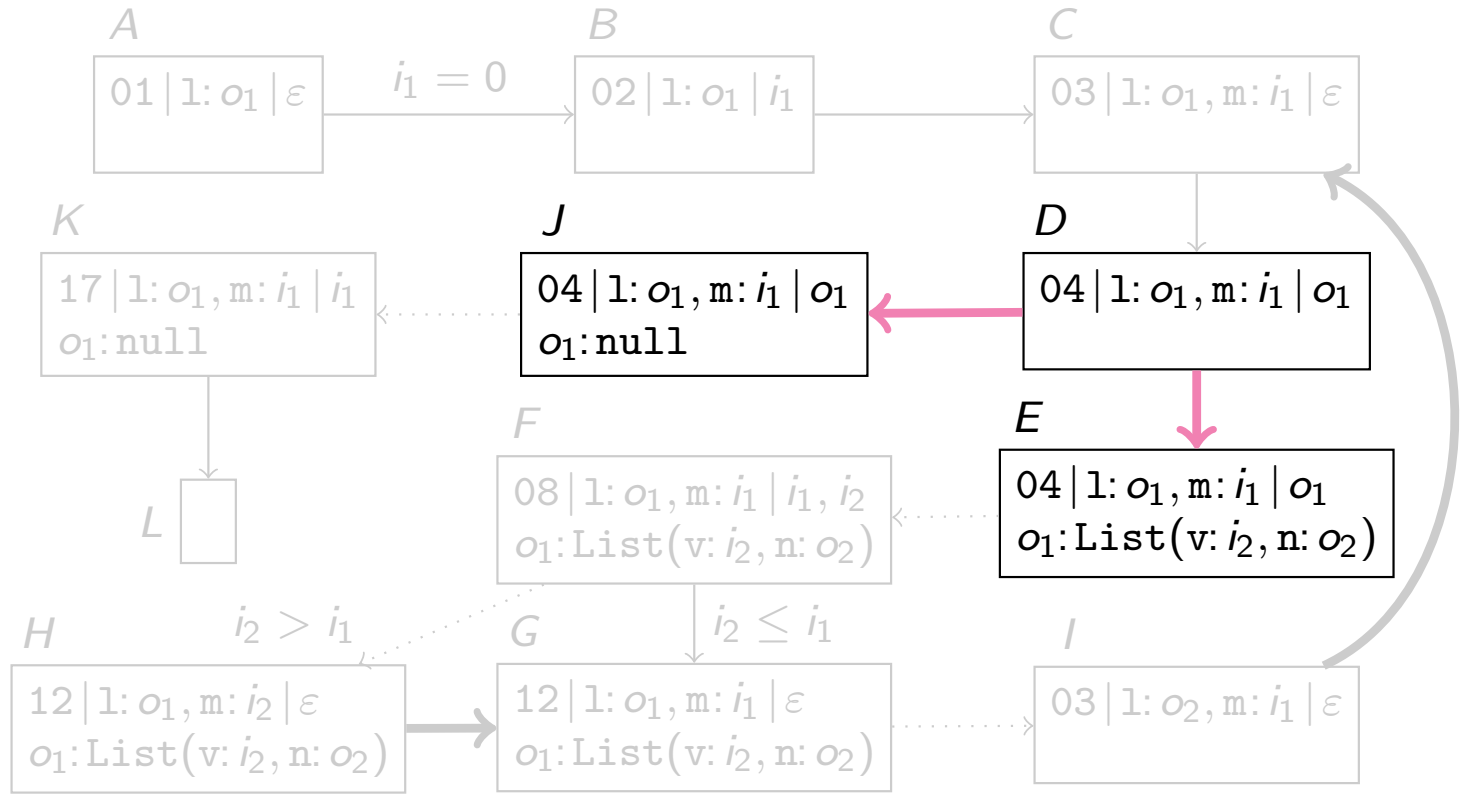
weight: 0

condition:

$\|o\| = 0$	if	o : `null`
$\|o\| \geq 1$	if	o : `List(...)`
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

$\|o_1\| > \|o_2\|'$ if o_1 is refined and o_1 reaches o_2 and o_1! not in refined state

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$D \xrightarrow{0} J$ **if** $\|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

$D \xrightarrow{0} E$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge \ldots \wedge \|o_1\| > \|o_2\|'$

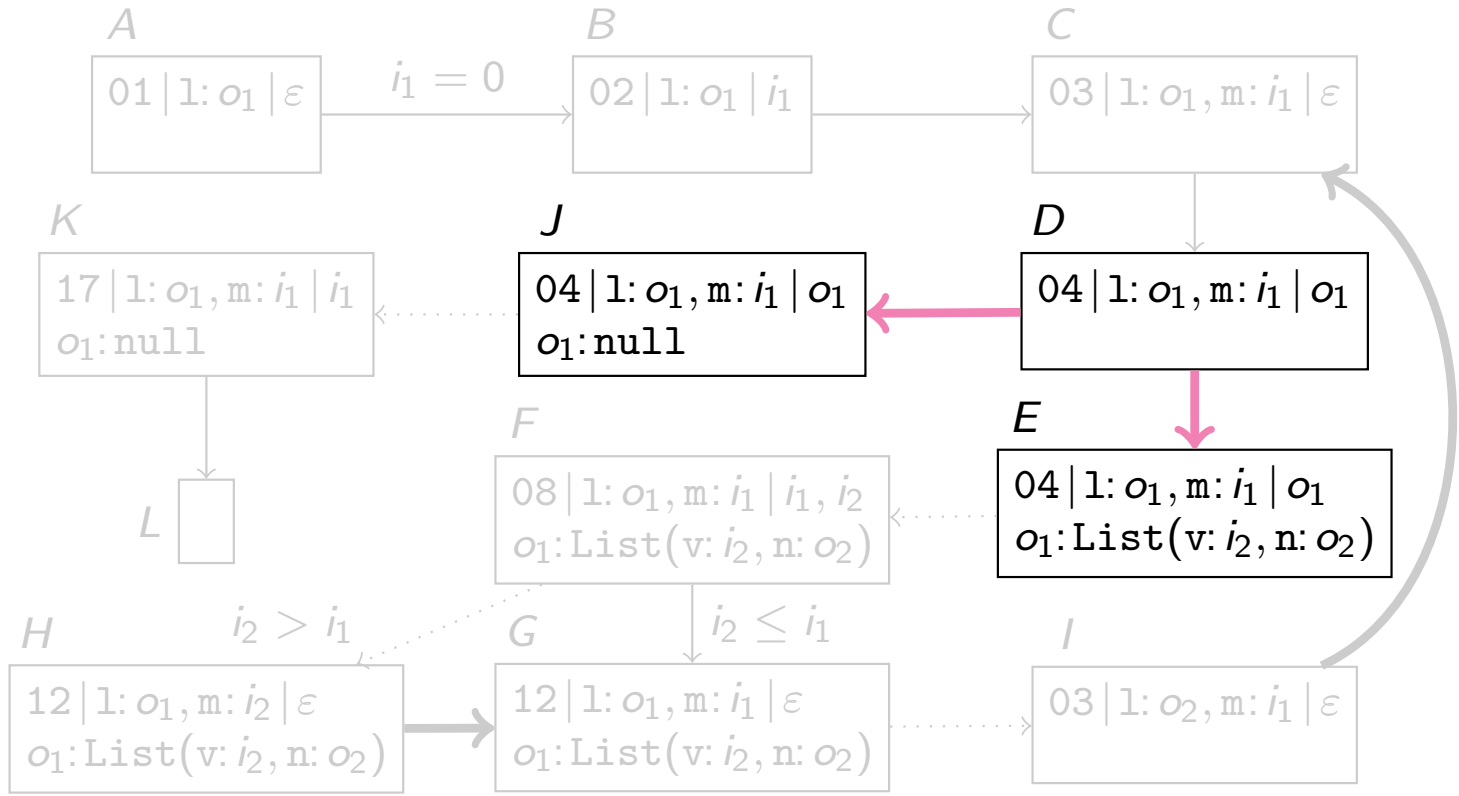
weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

$\|o_1\| > \|o_2\|'$ if o_1 is refined and o_1 reaches o_2 and o_1! not in refined state

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$D \xrightarrow{0} J \quad \textbf{if} \quad \|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$$

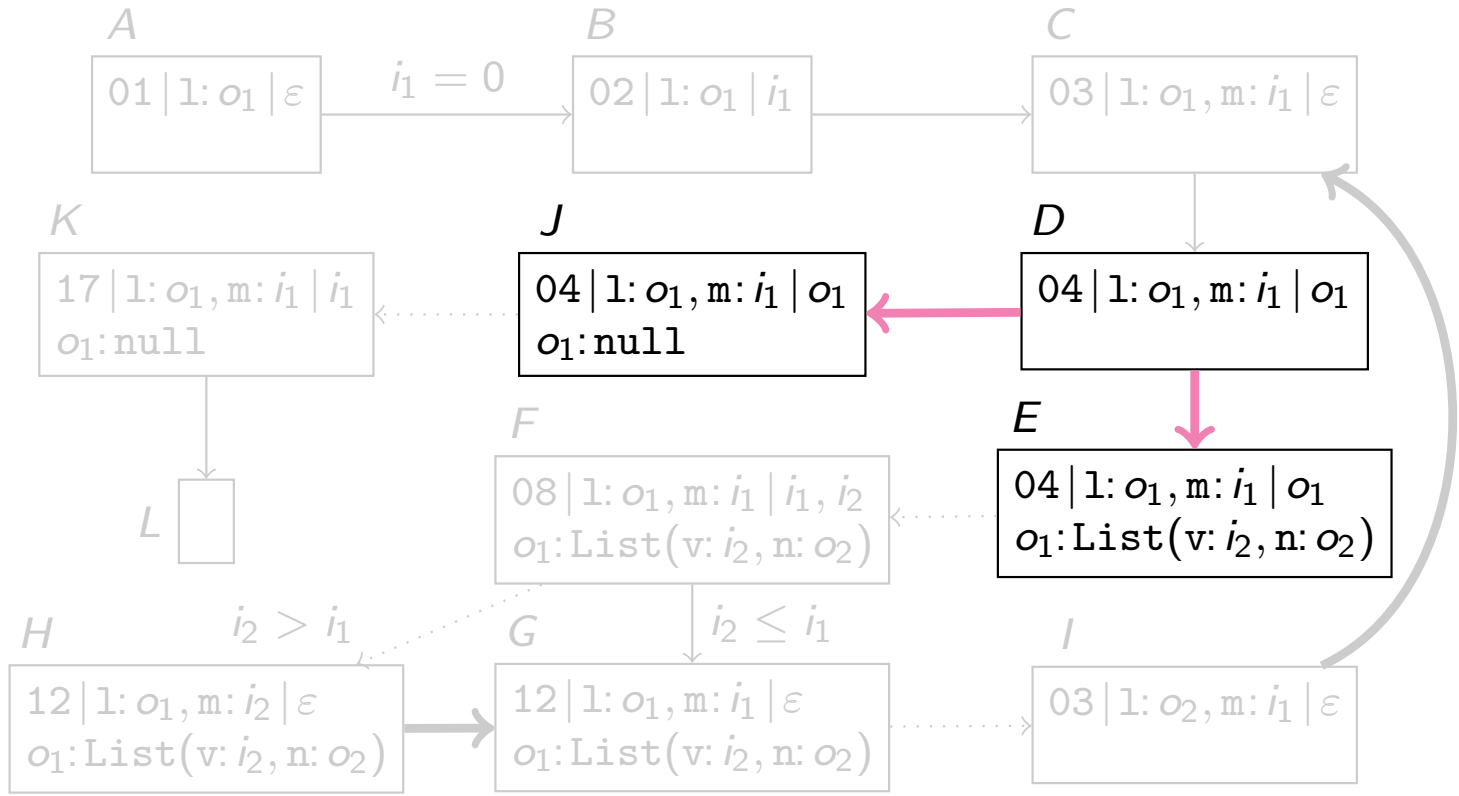
$$D \xrightarrow{0} E \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge \ldots \wedge \|o_1\| > \|o_2\|'$$

weight: 0

condition:

$\|o\| = 0$	if o : null	$\|o_1\| > \|o_2\|'$	if o_1 is refined and o_1 reaches o_2 and o_1! not in refined state
$\|o\| \geq 1$	if o : List(...)		
$\|o\| \geq 0$	otherwise		
$x' = x$	for all variables x	$\|o_1\| > \|i'\|$	if o_1 is refined and o_1 reaches i

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$D \xrightarrow{0} J$ **if** $\|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

$D \xrightarrow{0} E$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge \ldots \wedge \|o_1\| > \|o_2\|'$

weight: 0

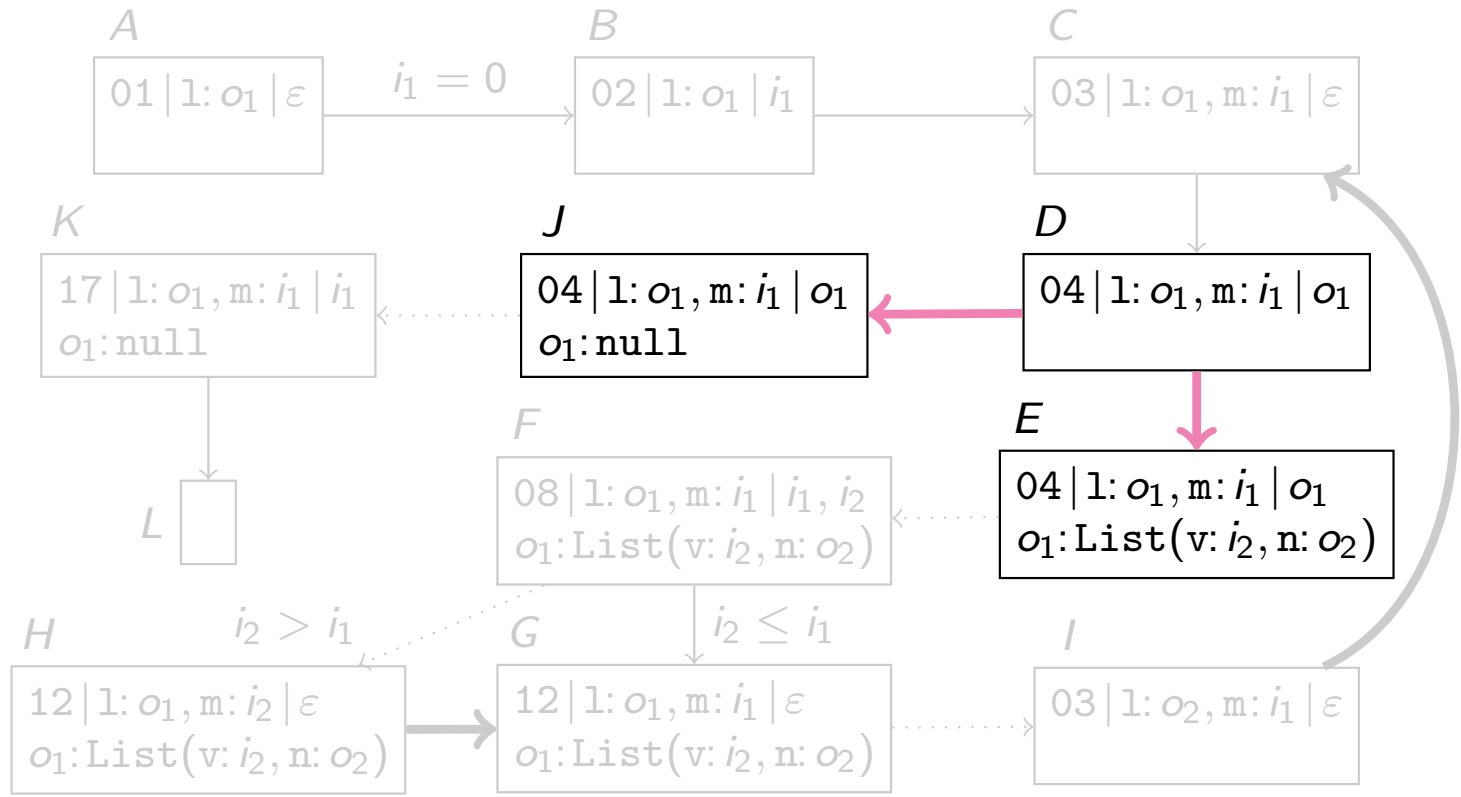
condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

$\|o_1\| > \|o_2\|'$ if o_1 is refined and o_1 reaches o_2 and o_1! not in refined state

$\|o_1\| > i' > -\|o_1\|$ if o_1 is refined and o_1 reaches i

Transform Refinement Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$D \xrightarrow{0} J$ **if** $\|o_1\| = 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_1$

$D \xrightarrow{0} E$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\|' \geq 0 \ \wedge \ldots \wedge \|o_1\| > \|o_2\|' \ \wedge\ \|o_1\| > i_2' > -\|o_1\|$

weight: 0

condition:

$\|o\| = 0$	if	o : `null`
$\|o\| \geq 1$	if	o : `List(...)`
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

$\|o_1\| > \|o_2\|'$ if o_1 is refined and o_1 reaches o_2 and o_1! not in refined state

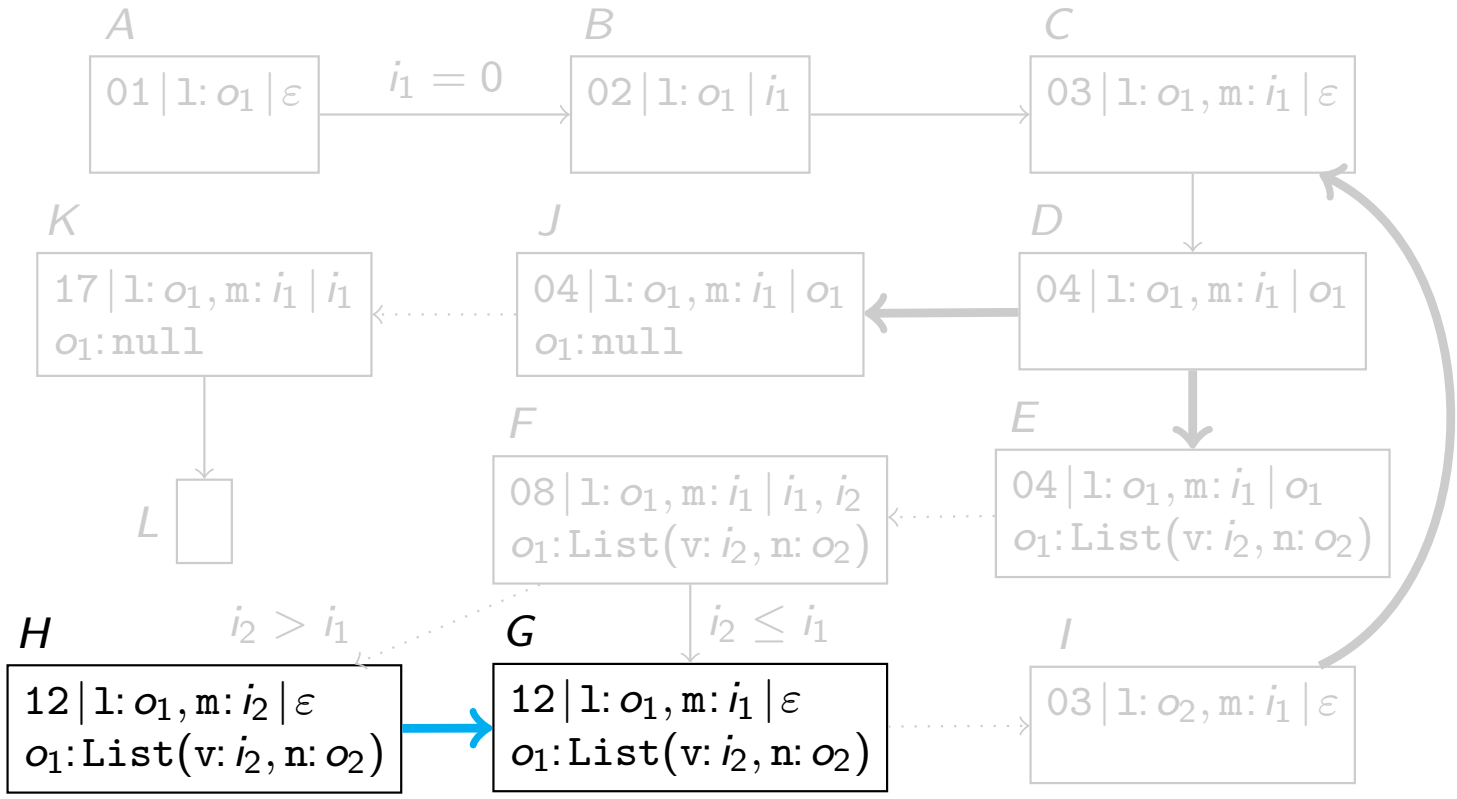
$\|o_1\| > i' > -\|o_1\|$ if o_1 is refined and o_1 reaches i

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



weight: 0

condition:

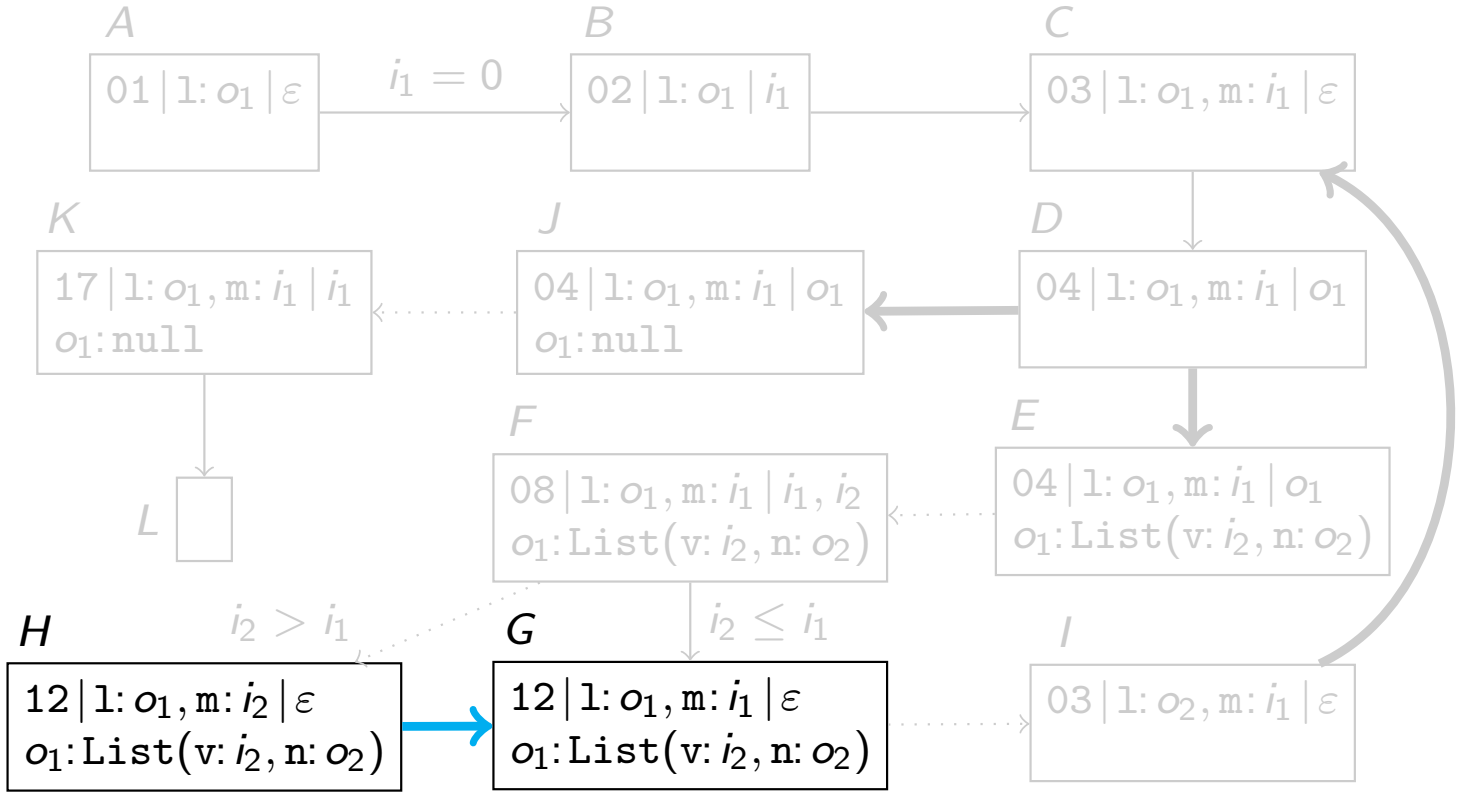
$\|o\| = 0$	if	o : null
$\|o\| \geq 1$	if	o : List(...)
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$H \xrightarrow{0} G$

weight: 0

condition:

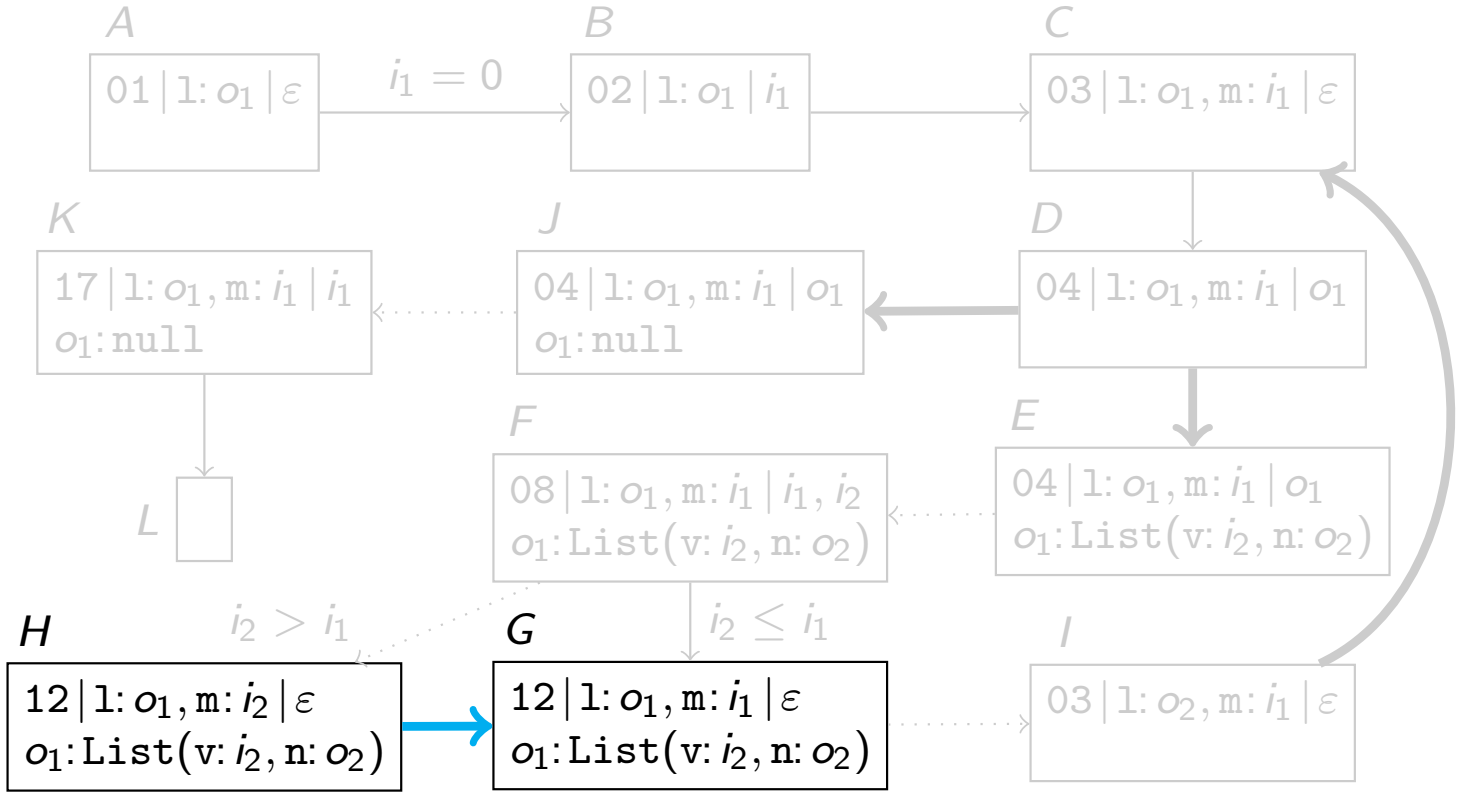
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0$

weight: 0

condition:

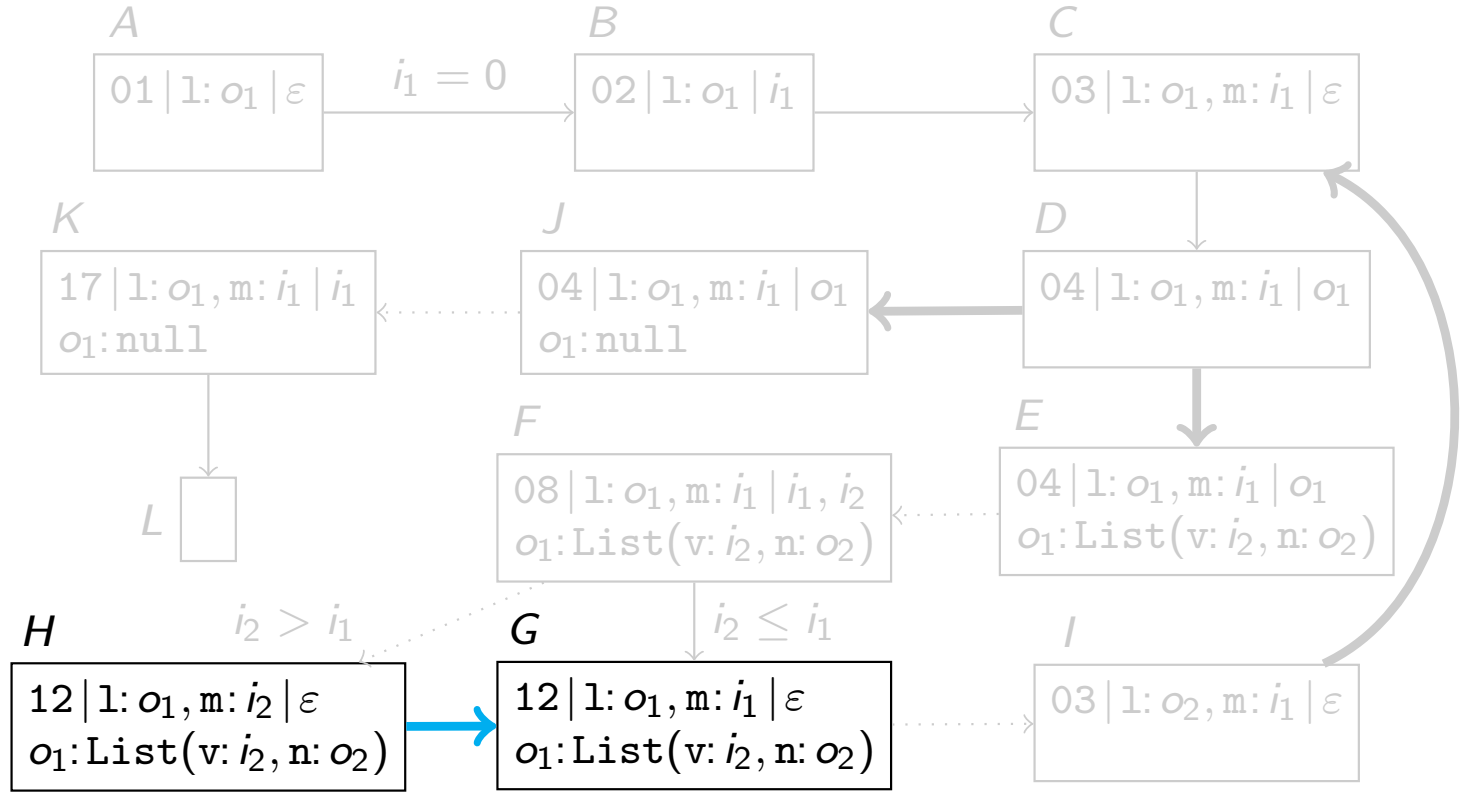
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = x$	for all variables x	

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0$

weight: 0

condition:

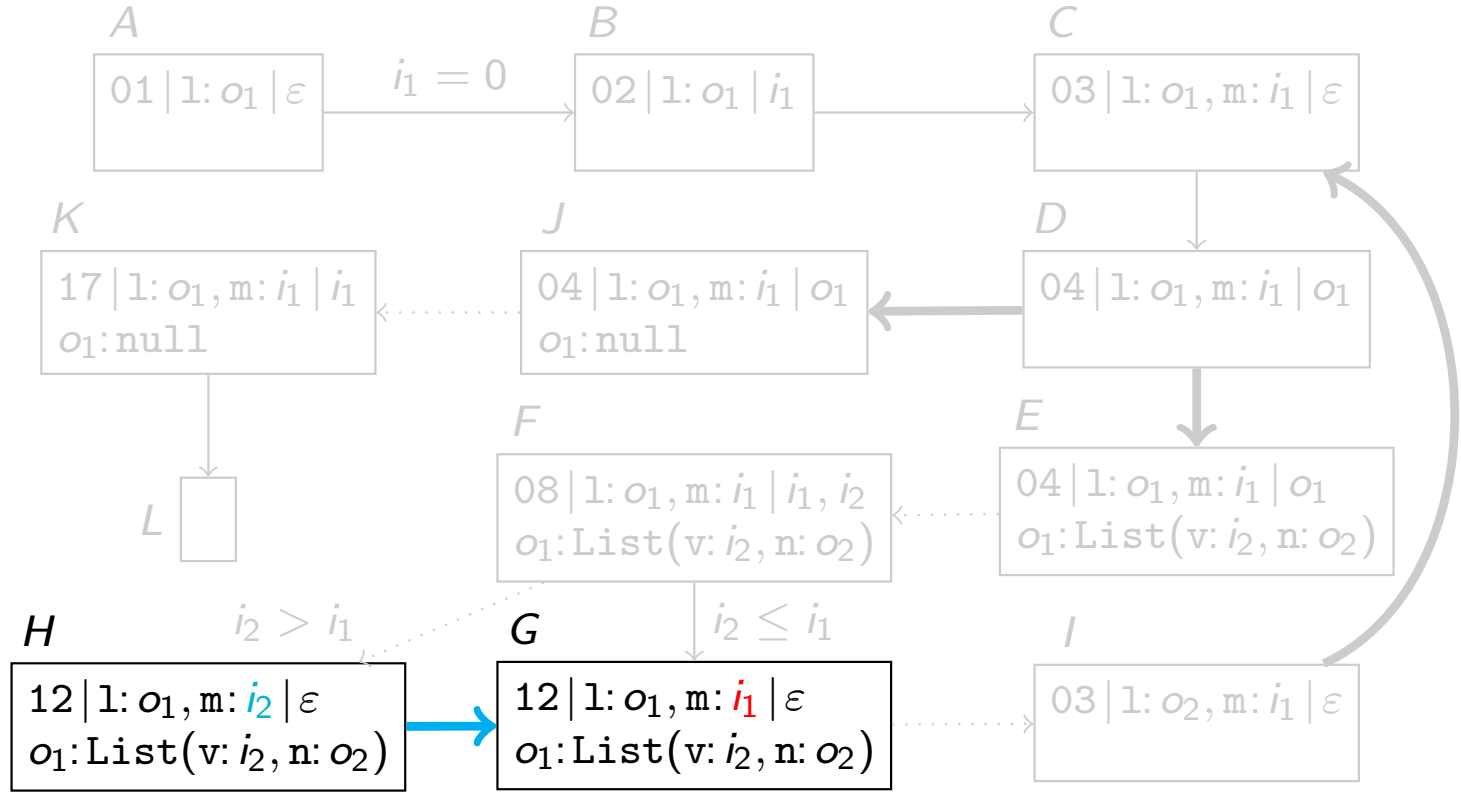
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = y$	if x in general state corresponds to y in special state	

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge \ \|o_2\| \geq 0$

weight: 0

condition:

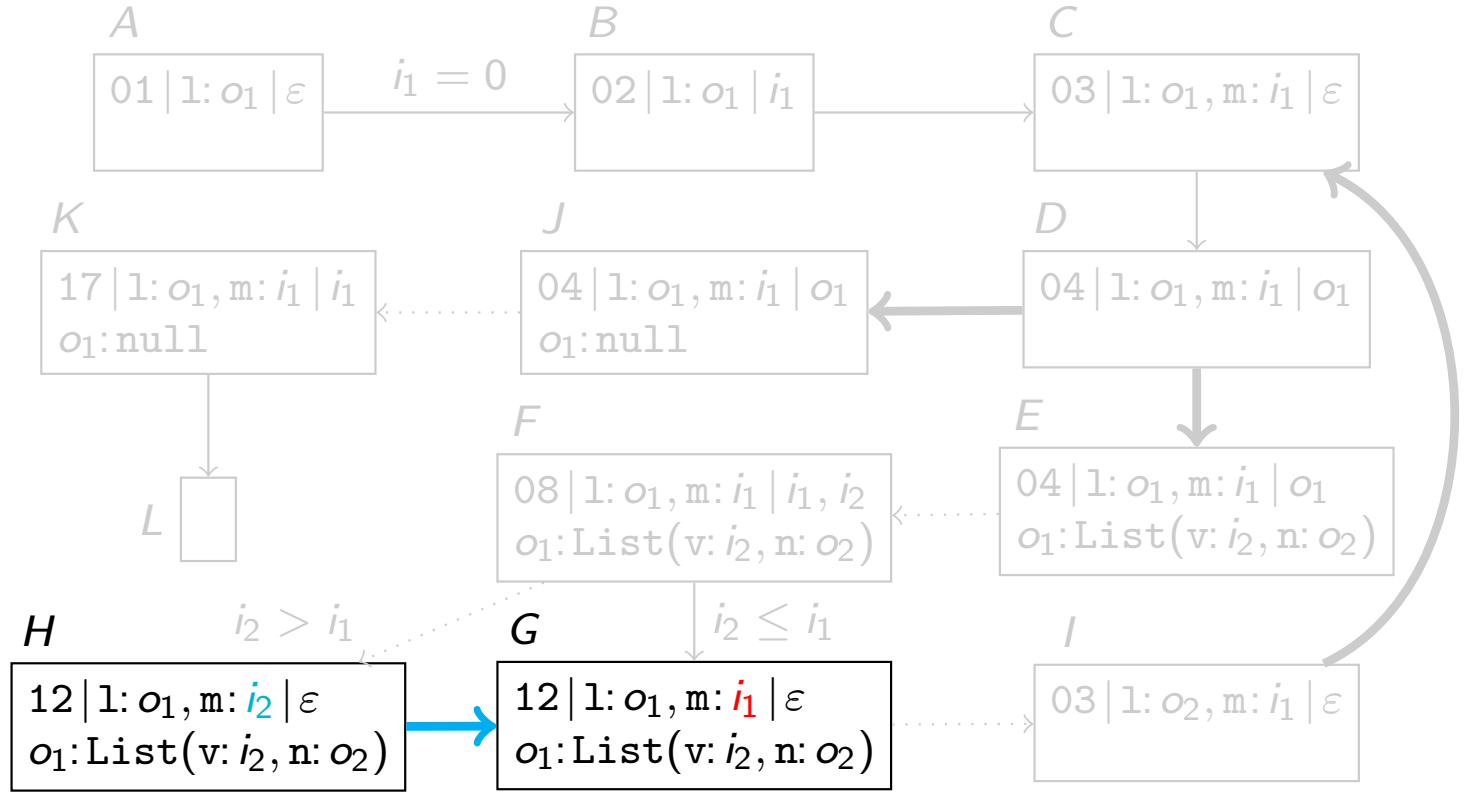
$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = y$	if x in general state corresponds to y in special state	

Transform Generalization Edges

ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$



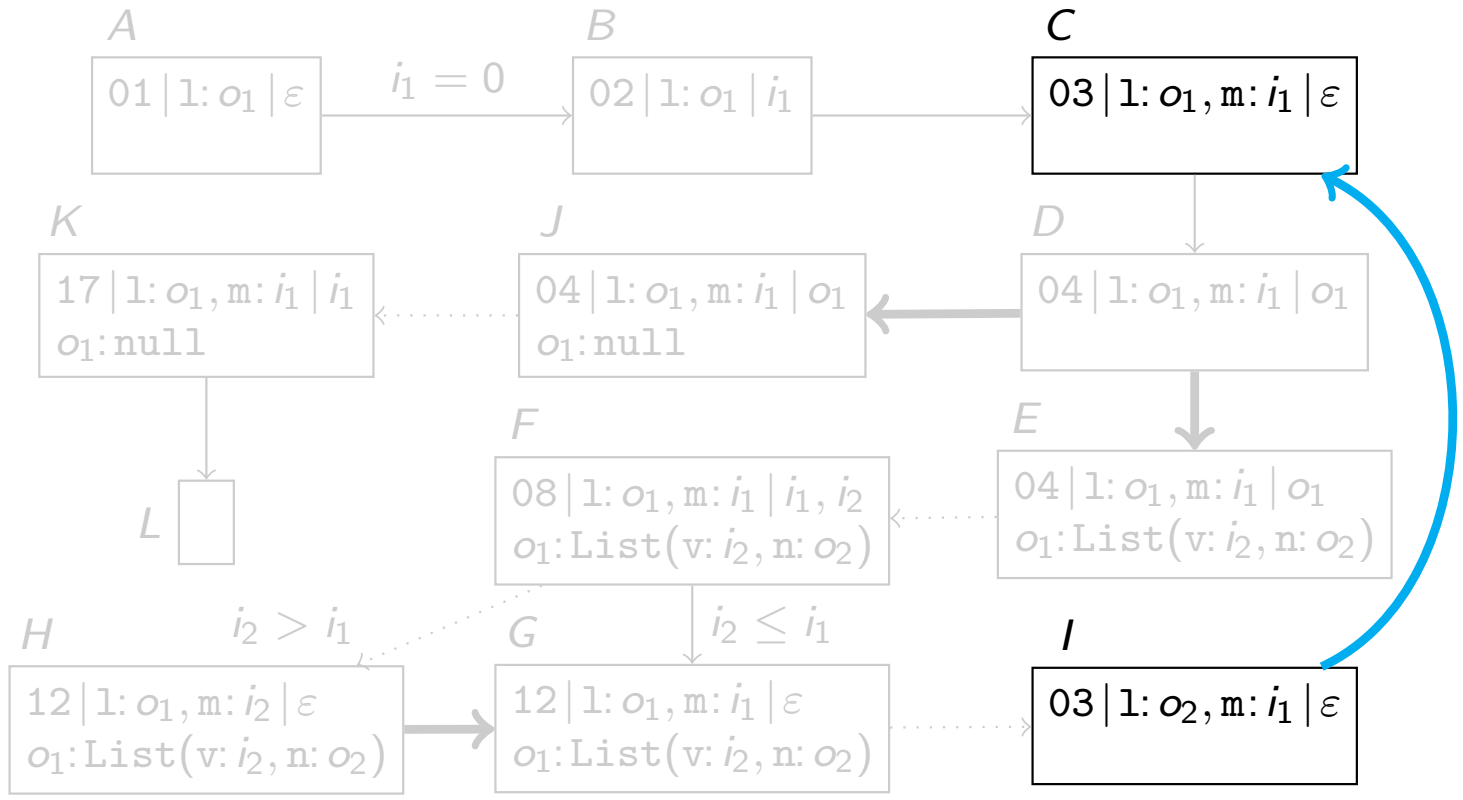
$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_2 \ \wedge \ldots$

weight: 0

condition:

$\|o\| = 0$	if $o : \texttt{null}$
$\|o\| \geq 1$	if $o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise
$x' = y$	if x in general state corresponds to y in special state

Transform Generalization Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

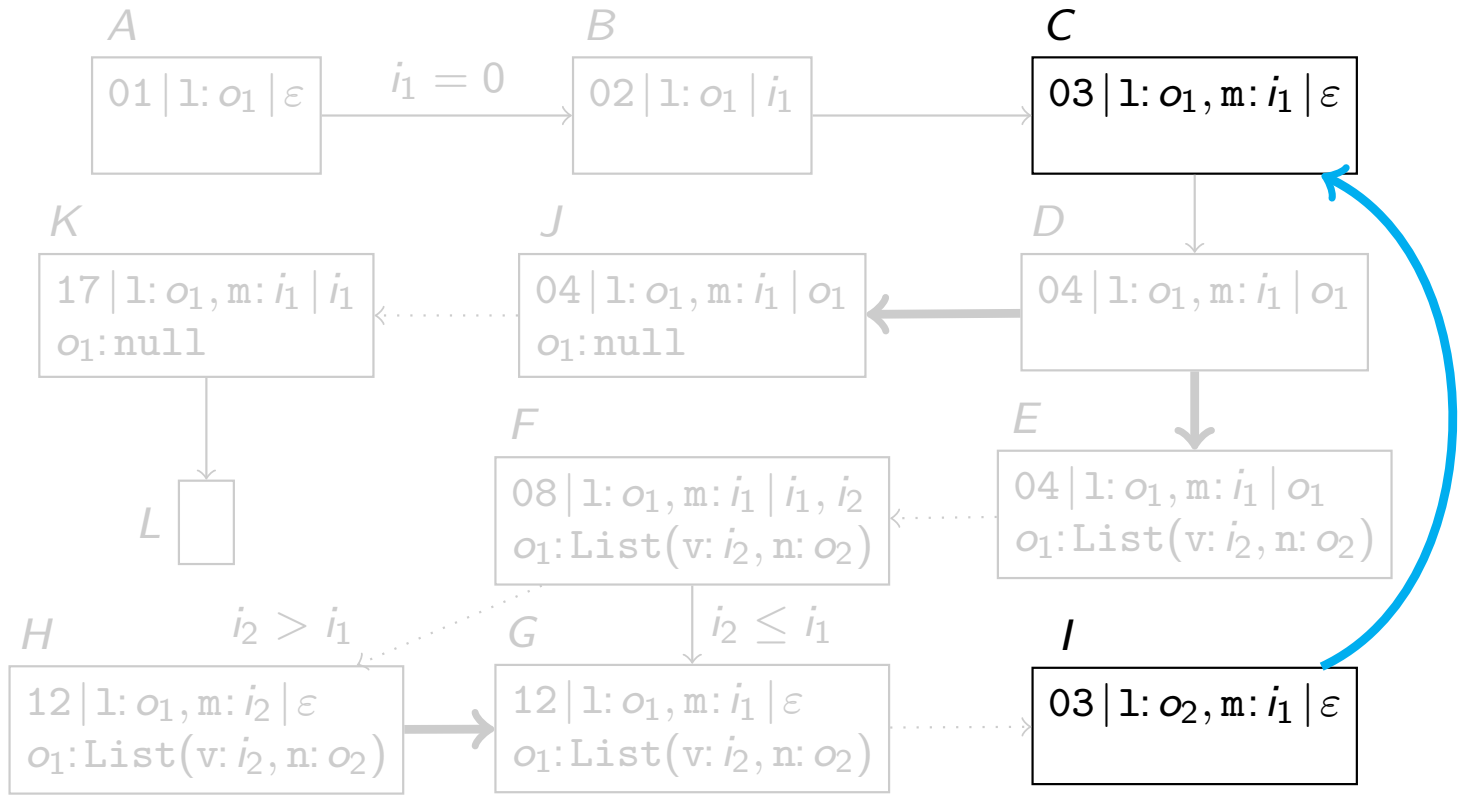
$$H \xrightarrow{0} G \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_2 \ \wedge \ldots$$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = y$	if x in general state corresponds to y in special state	

Transform Generalization Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_2 \ \wedge \ldots$

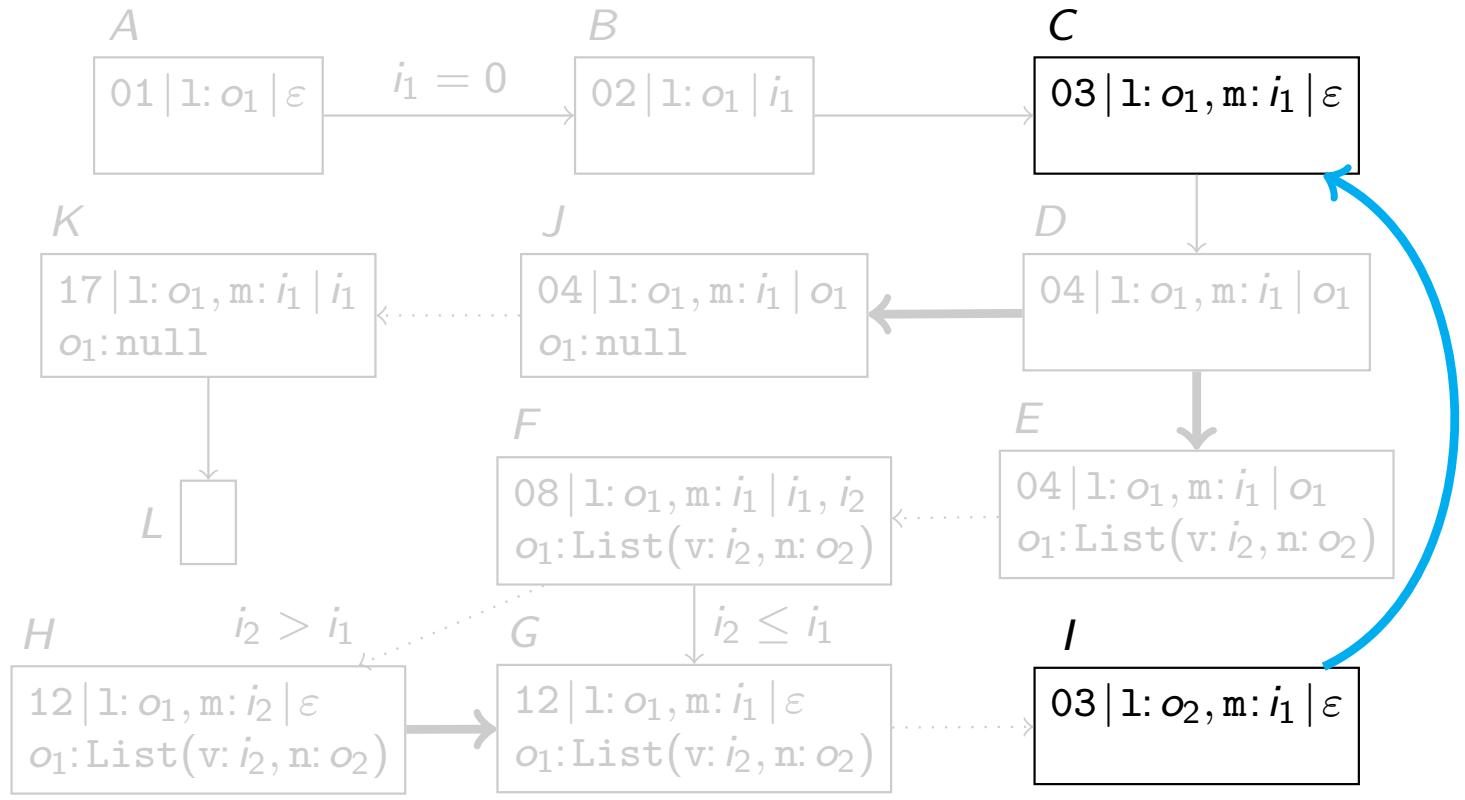
$I \xrightarrow{0} C$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = y$	if x in general state corresponds to y in special state	

Transform Generalization Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$H \xrightarrow{0} G$ **if** $\|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_2 \ \wedge \ldots$

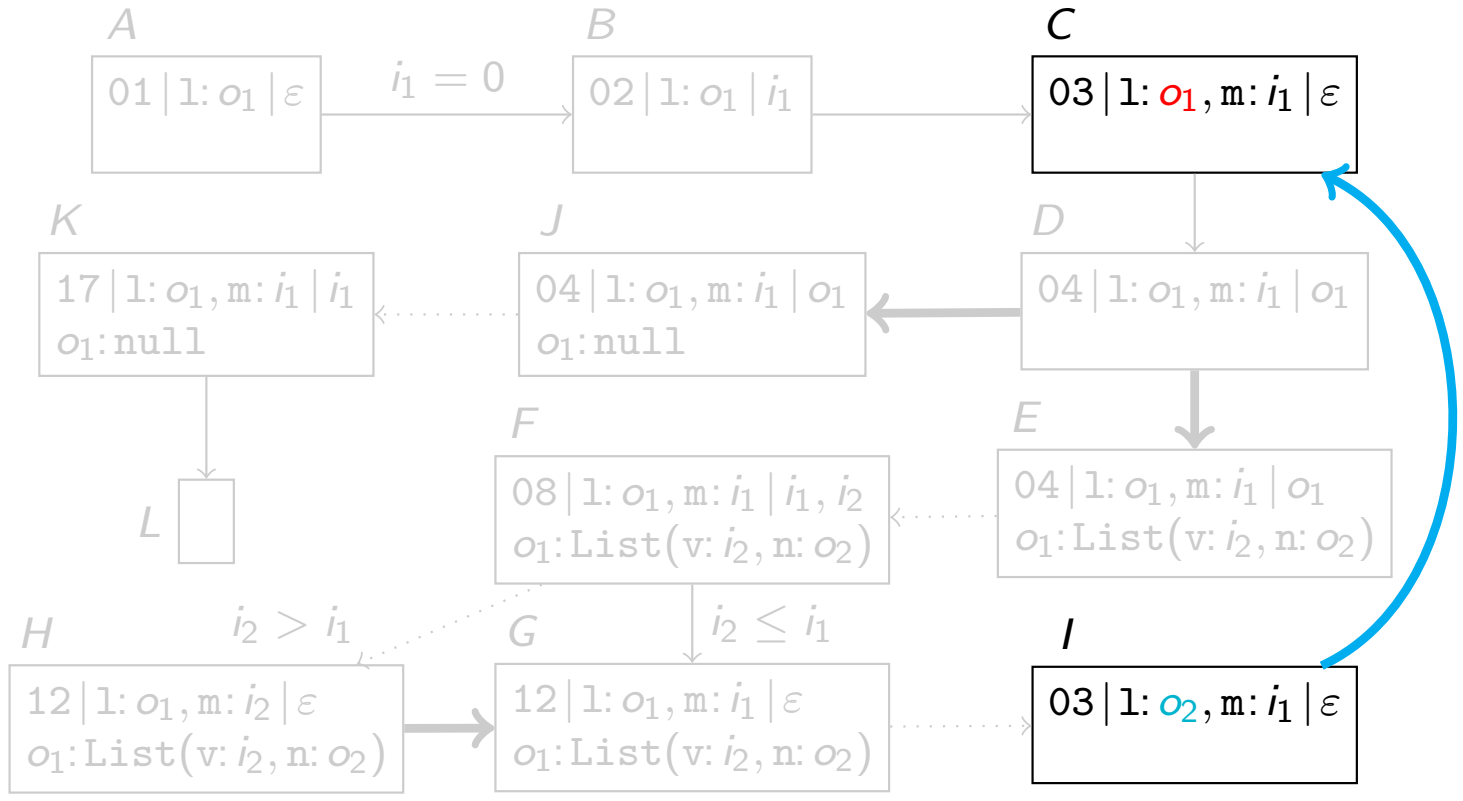
$I \xrightarrow{0} C$ **if** $\|o_2\| \geq 0$

weight: 0

condition:

$\|o\| = 0$	if	$o : \texttt{null}$
$\|o\| \geq 1$	if	$o : \texttt{List}(\ldots)$
$\|o\| \geq 0$	otherwise	
$x' = y$	if x in general state corresponds to y in special state	

Transform Generalization Edges



ITS over variables

$i_1,\ i_2,\ \|o_1\|\ ,\ \|o_2\|$

$i_1',\ i_2',\ \|o_1\|',\ \|o_2\|'$

$$H \xrightarrow{0} G \quad \textbf{if} \quad \|o_1\| \geq 1 \ \wedge\ \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_1\| \ \wedge\ i_1' = i_2 \ \wedge \ldots$$

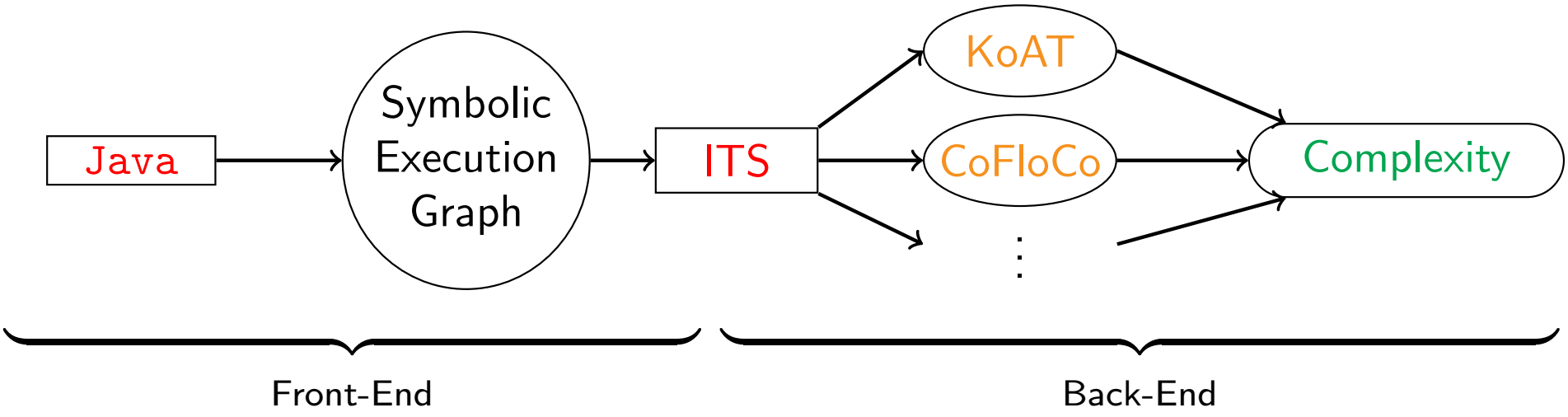
$$I \xrightarrow{0} C \quad \textbf{if} \quad \|o_2\| \geq 0 \ \wedge\ \|o_1\|' = \|o_2\| \ \wedge\ i_1' = i_1$$

weight: 0

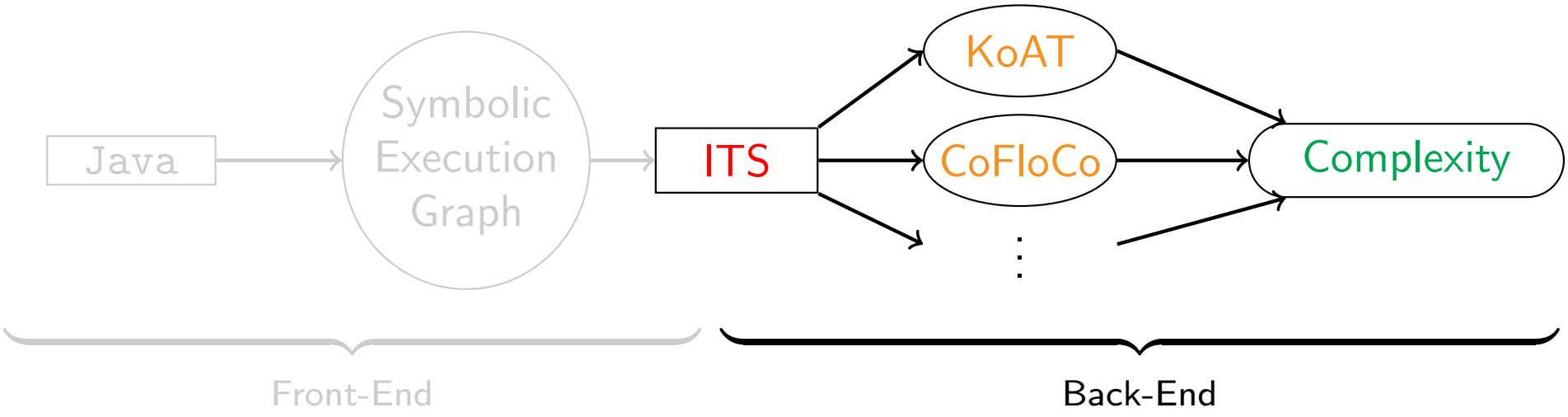
condition:

$\|o\| = 0$	if o : `null`
$\|o\| \geq 1$	if o : `List(...)`
$\|o\| \geq 0$	otherwise
$x' = y$	if x in general state corresponds to y in special state

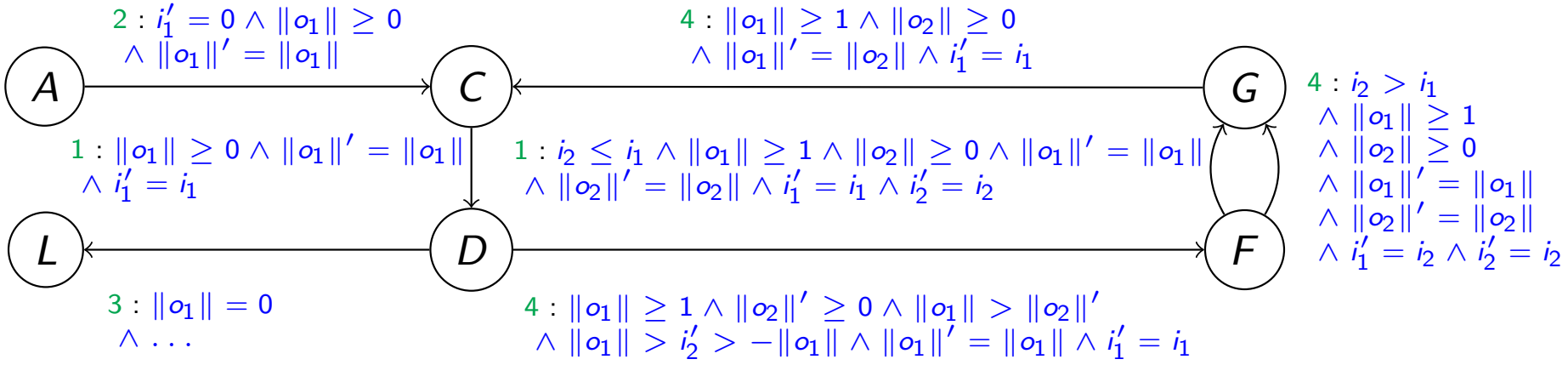
AProVE for Complexity Analysis of Java



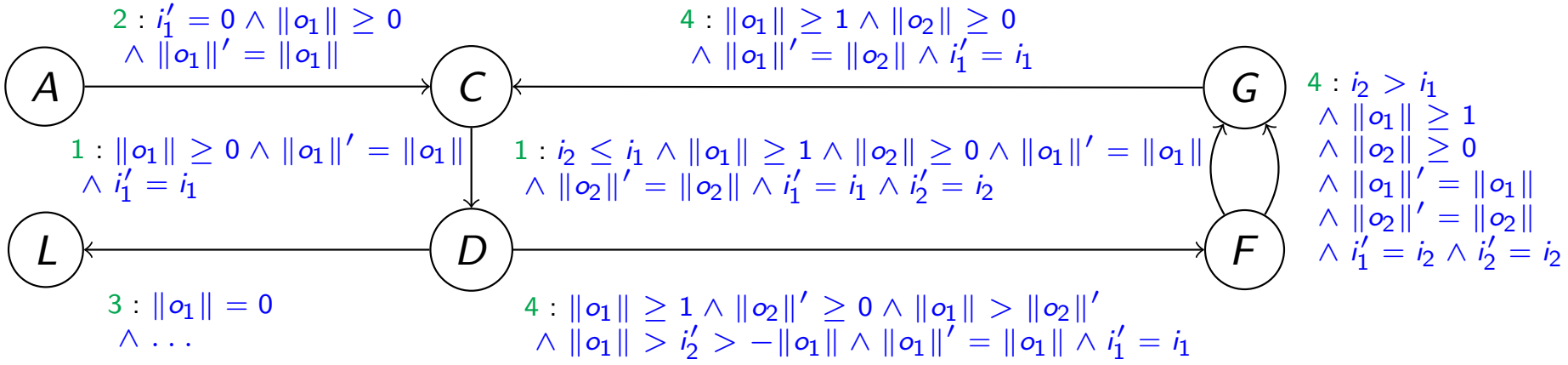
AProVE for Complexity Analysis of Java



Complexity of Integer Transition Systems



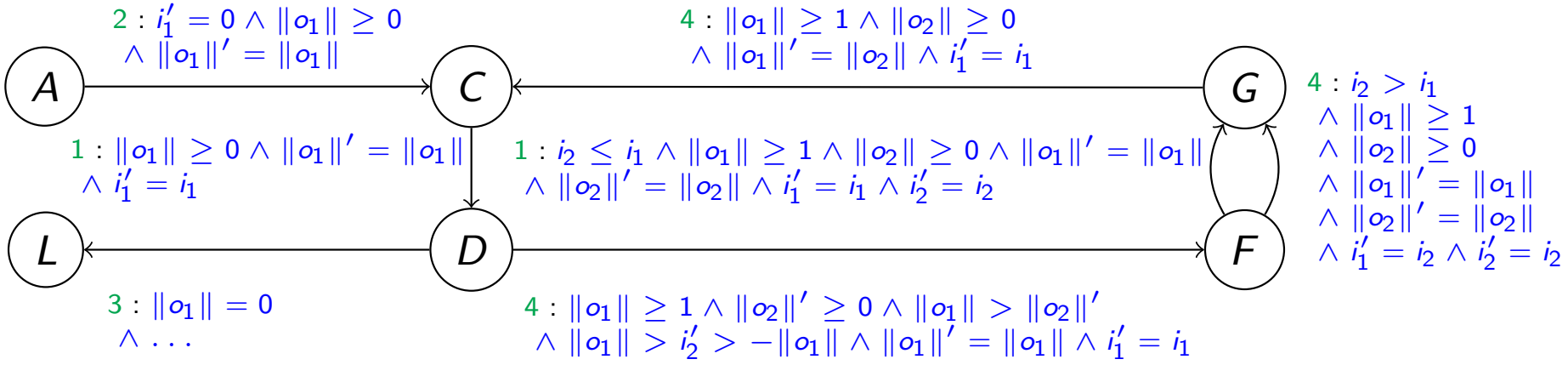
Complexity of Integer Transition Systems



Complexity Bound b for location s:

$$b \quad \geq \quad \text{cost of any ITS evaluation starting with } s$$

Complexity of Integer Transition Systems

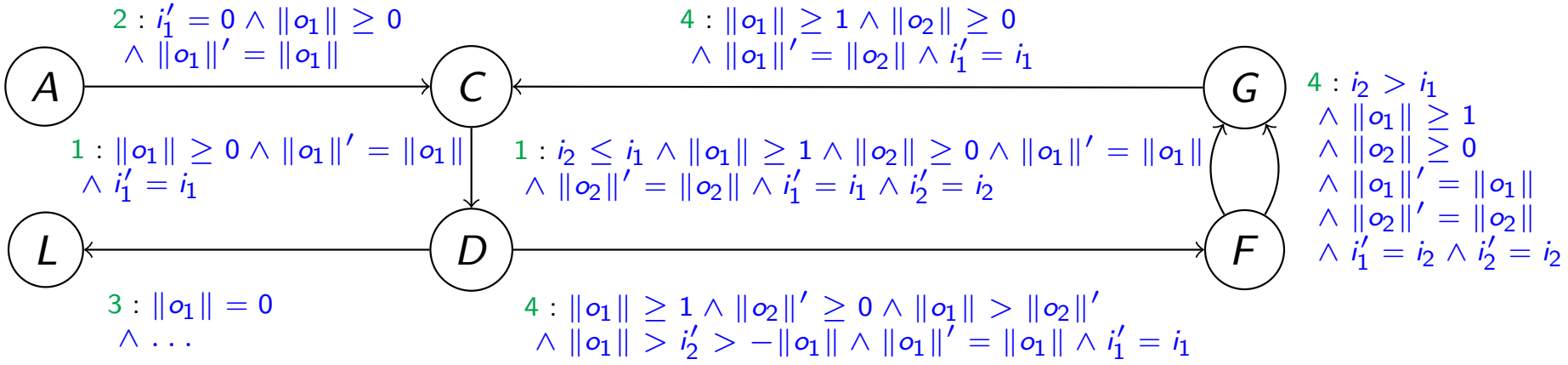


Complexity Bound b for location s:

$$b \quad \geq \quad \text{cost of any ITS evaluation starting with } s$$

for all instantiations of variables in b and s

Complexity of Integer Transition Systems



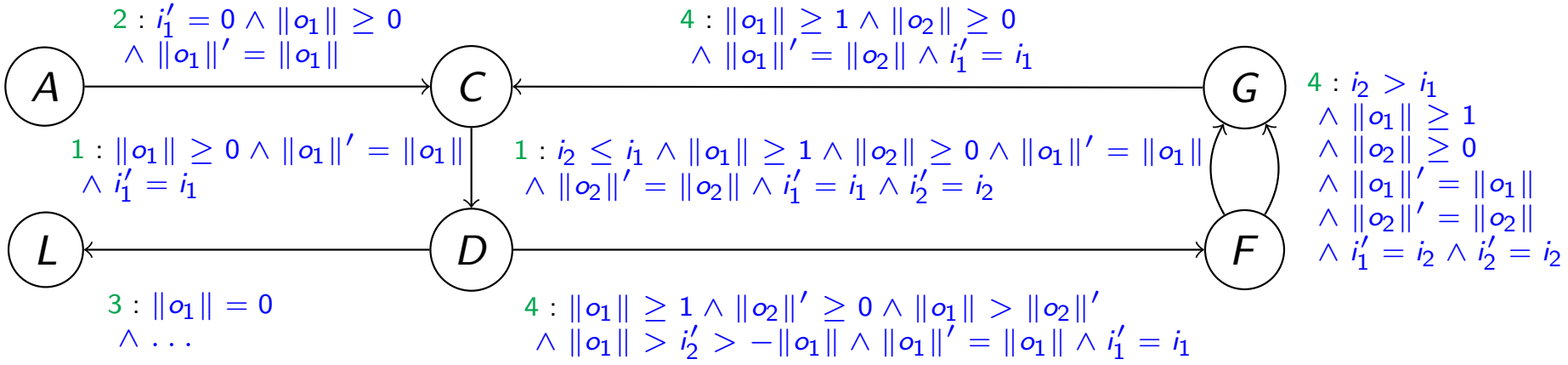
$13 \cdot \|o_1\| + 6$ is complexity bound for Location A

Complexity Bound b for location s:

$$b \quad \geq \quad \text{cost of any ITS evaluation starting with } s$$

for all instantiations of variables in b and s

Complexity of Integer Transition Systems



$13 \cdot \|o_1\| + 6$ is complexity bound for Location A

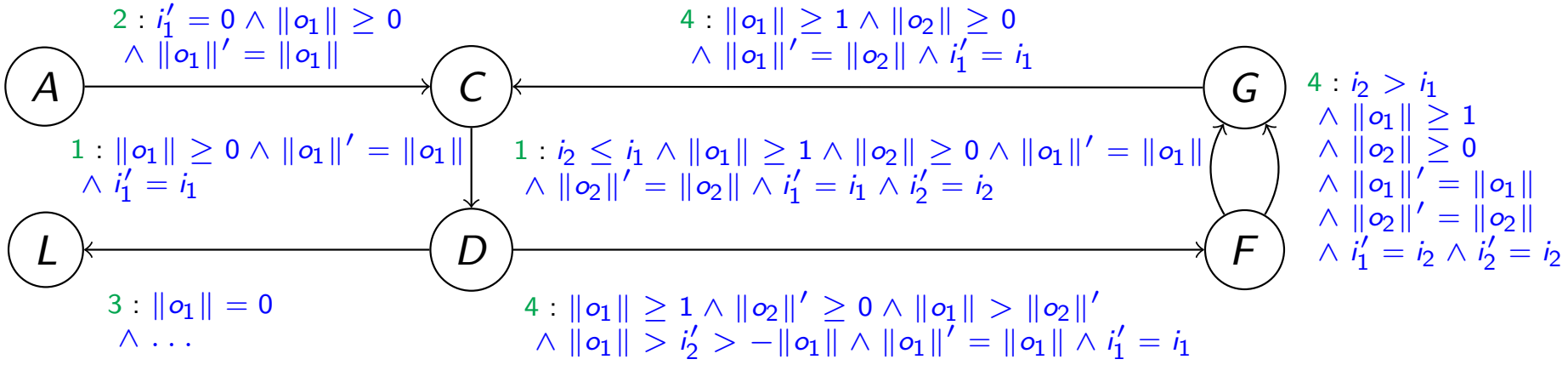
- linear bound determined by KoAT, CoFloCo, ...

Complexity Bound b for location s:

$$b \quad \geq \quad \text{cost of any ITS evaluation starting with } s$$

for all instantiations of variables in b and s

Complexity of Integer Transition Systems



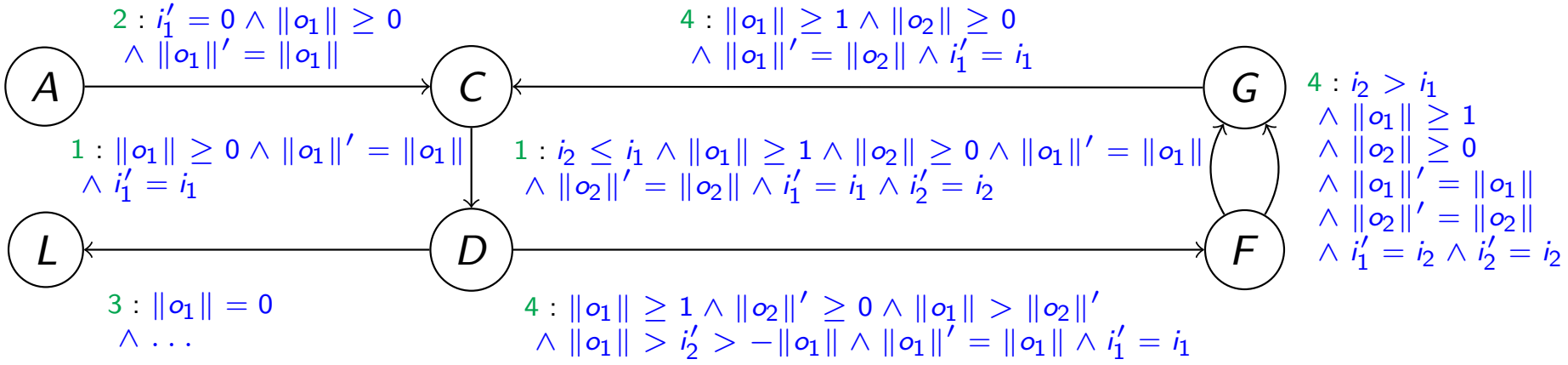
$13 \cdot \|o_1\| + 6$ is complexity bound for Location A

- linear bound determined by KoAT, CoFloCo, ...

Soundness Theorem

Complexity bound for location s in ITS is also complexity bound for state s in symbolic execution graph

Complexity of Integer Transition Systems



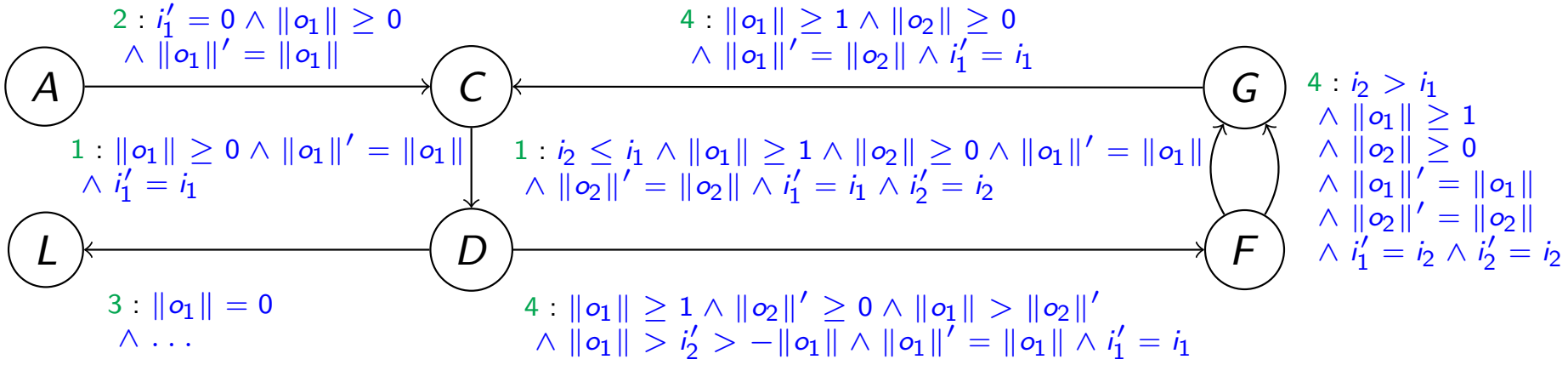
$13 \cdot \|o_1\| + 6$ is complexity bound for State A

- linear bound determined by KoAT, CoFloCo, ...

Soundness Theorem

Complexity bound for location s in ITS is also complexity bound for state s in symbolic execution graph

Complexity of Integer Transition Systems



$13 \cdot \|o_1\| + 6$ is complexity bound for State A

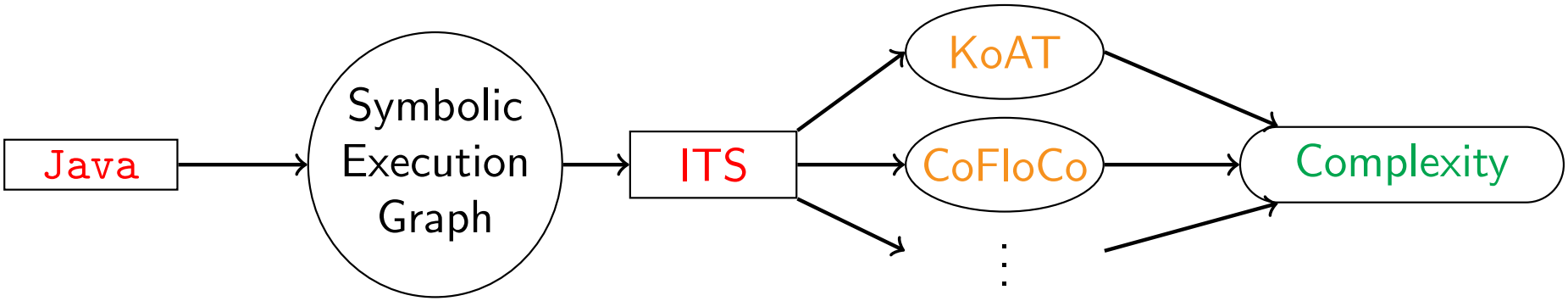
- linear bound determined by KoAT, CoFloCo, ...
- $13 \cdot \|\mathtt{l}\| + 6$ is upper bound for runtime of `max`

```
static int max(List l) {
  int m = 0;
  while (l != null) {
    if (l.v > m) {
      m = l.v;
    }
    l = l.n;
  }
  return m;
}
```

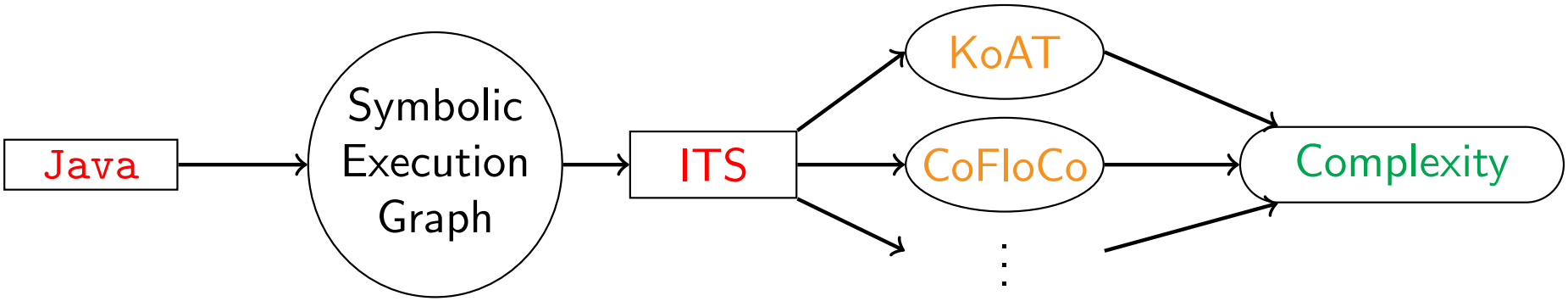
Soundness Theorem

Complexity bound for location s in ITS is also complexity bound for state s in symbolic execution graph

AProVE for Complexity Analysis of Java

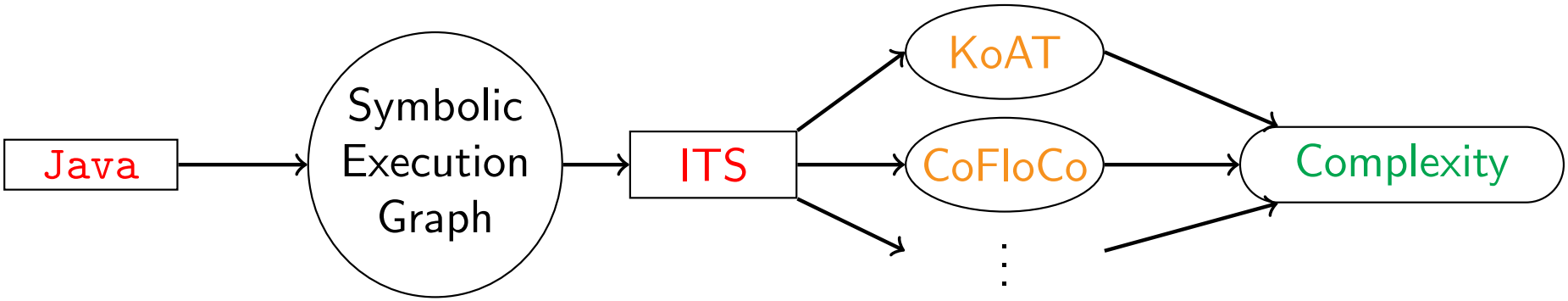


AProVE for Complexity Analysis of Java



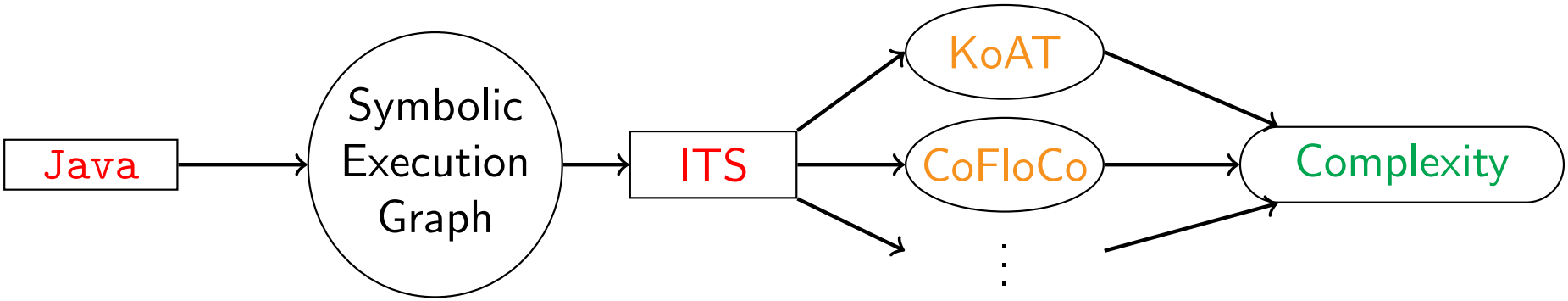
- runtime complexity analysis for heap-manipulating Java programs

AProVE for Complexity Analysis of Java



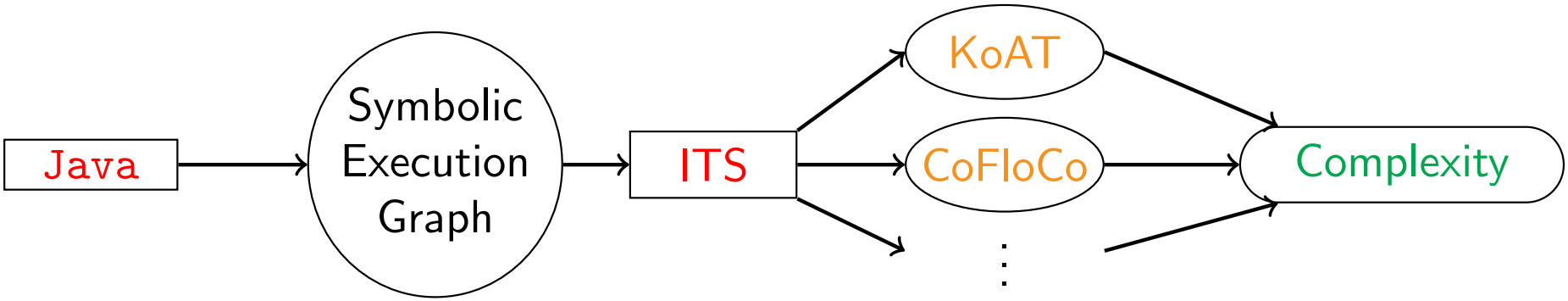
- runtime complexity analysis for heap-manipulating Java programs
- adaption for space or size analysis

AProVE for Complexity Analysis of Java



- runtime complexity analysis for heap-manipulating Java programs
- adaption for space or size analysis
- modular analysis by using summaries

AProVE for Complexity Analysis of Java

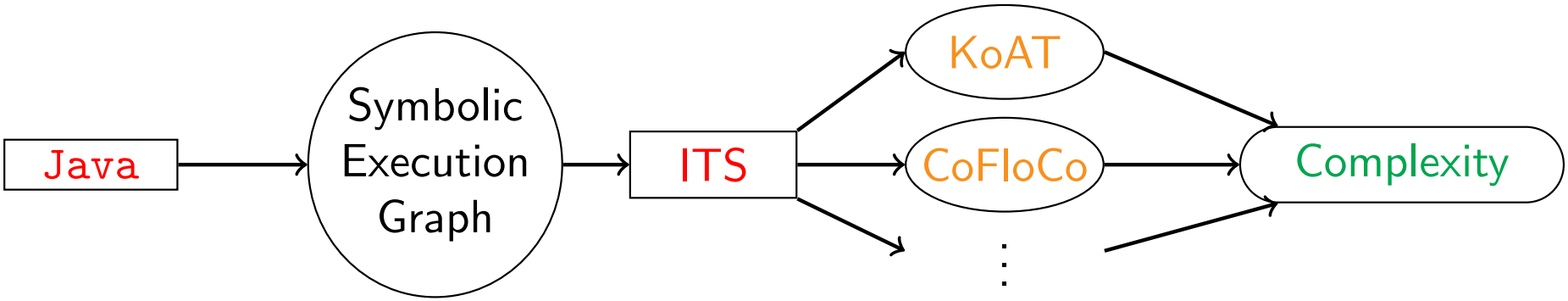


- runtime complexity analysis for heap-manipulating Java programs
- adaption for space or size analysis
- modular analysis by using summaries

Experiments on 211 programs from the TPDB

	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{>3})$	?	Success
AProVE	28	0	102	0	13	2	4	62	71 %
COSTA	10	4	45	3	5	0	1	143	32 %

AProVE for Complexity Analysis of Java



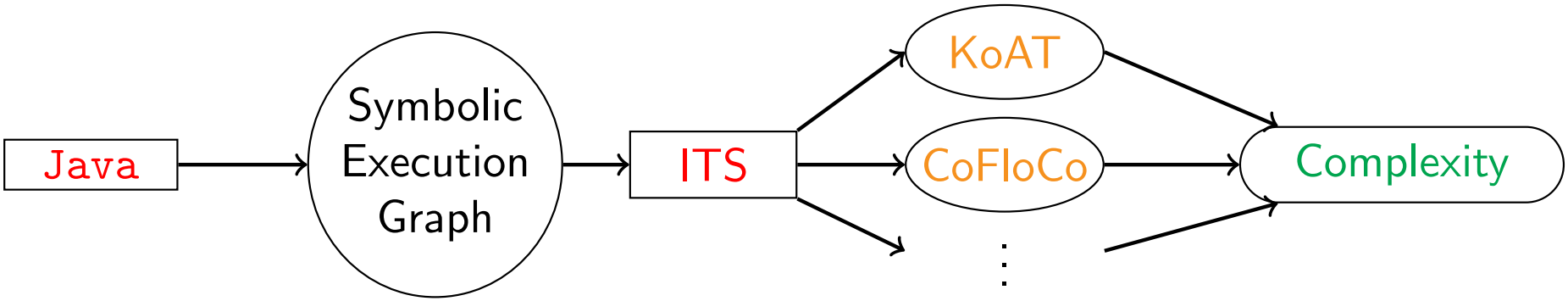
- runtime complexity analysis for heap-manipulating Java programs
- adaption for space or size analysis
- modular analysis by using summaries

Experiments on 211 programs from the TPDB

	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \cdot \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{>3})$	?	Success
AProVE	28	0	102	0	13	2	4	62	71 %
COSTA	10	4	45	3	5	0	1	143	32 %

AProVE: *size* considers path length and values
COSTA: *size* considers path length only

AProVE for Complexity Analysis of Java



- runtime complexity analysis for heap-manipulating Java programs
- adaption for space or size analysis
- modular analysis by using summaries

STAC Project "Complexity Analysis-Based Guaranteed Execution"

- DARPA program "Space/Time Analysis for Cybersecurity"
- joint project with Draper Inc. and University of Innsbruck
- AProVE and KoAT crucial to detect/prove absence of vulnerabilities for Java programs