

Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode

M. Brockschmidt, T. Ströder, C. Otto, J. Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

2011-10-19, Karlsruhe Institute of Technology

Automated Non-Termination Analysis

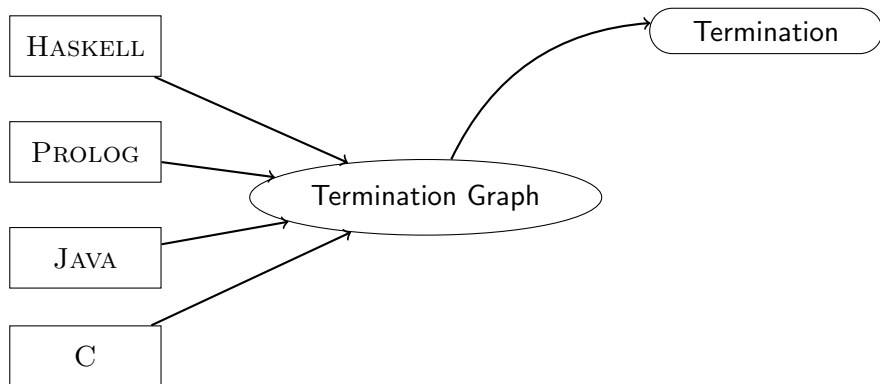
- Logic programs:
De Schreye '90 ..., Payet & Mesnard '06, ...
- TRSs and SRSs:
Giesl et. al '05, Payet '06, ...
- C:
Gupta et. al '08, ...
- JBC:
Velroyen '08, Payet & Spoto '09

Static program analysis

- Programming languages *complex* \curvearrowright Simpler representation needed

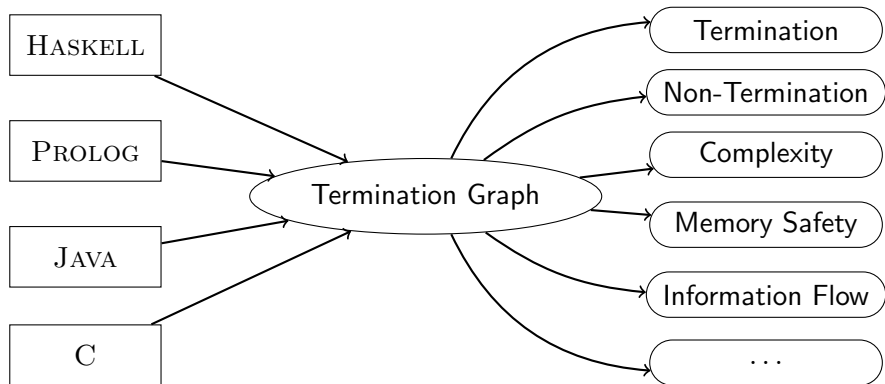
Static program analysis

- Programming languages *complex* \leadsto Simpler representation needed
- Termination Graphs: Simple, all information



Static program analysis

- Programming languages *complex* \leadsto Simpler representation needed
- Termination Graphs: Simple, all information
- **Useful for properties other than termination.**



The example

```
class Loop {  
  void main(String[] a){  
    int i = 0;  
    int j = a.length;  
    while (i < j) {  
      i += a[i].length();}}}
```

- 1 Adds up lengths.

The example

```
class Loop {  
  void main(String[] a){  
    int i = 0;  
    int j = a.length;  
    while (i < j) {  
      i += a[i].length();}}}
```

- 1 Adds up lengths.
- 2 May not terminate.

The example

```
class Loop {  
    void main(String[] a){  
        int i = 0;  
        int j = a.length;  
        while (i < j) {  
            i += a[i].length();}}}
```

- 1 Adds up lengths.
- 2 May not terminate.
- 3 May throw `NullPointerException`

The example

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

```
class Loop {
  void main(String[] a){
    int i = 0;
    int j = a.length;
    while (i < j) {
      i += a[i].length();}}}
```

- 1 Adds up lengths.
- 2 May not terminate.
- 3 May throw NullPointerException

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a_1		ε
a_1 : String[]		i_1		$i_1: [\geq 0]$

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a ₁		ε
a ₁ :String[]		i ₁		i ₁ : [≥0]

stack frame:

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00	a: a ₁	ε
a ₁ :String[]	i ₁	i ₁ : [≥0]

stack frame:

- Next program instruction

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00	a: a ₁	ε
a ₁ :String[]		i ₁ i ₁ : [≥0]

stack frame:

- Next program instruction
- Local variables
- Operand stack

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a ₁		ε
a ₁ : String[]		i ₁		i ₁ : [≥0]

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a_1		ε
		$a_1: \text{String}[]$	i_1	$i_1: [\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array
content unknown, length is i_1

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return

length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a_1		ε
a_1 :String[]		i_1		i_1 : $[\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array
content unknown, length is i_1
- at i_1 is a non-negative integer

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return
length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a_1		ε
a_1 :String[]		i_1		$i_1: [\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array
content unknown, length is i_1
- at i_1 is a non-negative integer
- Known String object:
 $\text{String}(\text{count} = i_3, \dots)$

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return
length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a_1		ε
a_1 : String[]		i_1		$i_1: [\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array
content unknown, length is i_1
- at i_1 is a non-negative integer
- Known String object:
String(count = i_3, \dots)
- Unknown String object:
String(?)

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0      #load 0 to stack
01: istore_1     #store to i
02: aload_0      #load a to stack
03: arraylength  #get array length
04: istore_2     #store to j
05: iload_1      #load i to stack
06: iload_2      #load j to stack
07: if_icmpge 22 #jump to end if i >= j
10: iload_1      #load i to stack
11: aload_0      #load a to stack
12: iload_1      #load i to stack
13: aaload       #load a[i]
14: invokevirtual length #call length()
17: iadd         #add length and i
18: istore_1     #store to i
19: goto 05
22: return
length():
00: aload_0      #load this to stack
01: getfield count #load count field
04: ireturn      #return it
```

00		a: a ₁		ε
a ₁ : String[]	i ₁	i ₁ : [≥0]		

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array
content unknown, length is i_1
- at i_1 is a non-negative integer
- Known String object:
String(count = i_3, \dots)
- Unknown String object:
String(?)

Only explicit sharing

```
main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
length():
00: aload_0
01: getfield count
04: ireturn
```

00 $a: a_1 \mid \epsilon$ $a_1: \text{String}[] \quad i_1 \quad i_1: [\geq 0]$	A
---	---

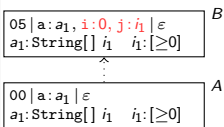
State A:

- What can happen when evaluating main?
- a_1 : Unknown array of String objects
- i_1 : a_1 's unknown length

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



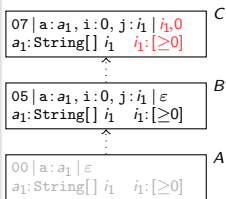
State B:

- Stored 0 to i
- Stored arraylength i_1 to j
- *Evaluations* from A to B

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



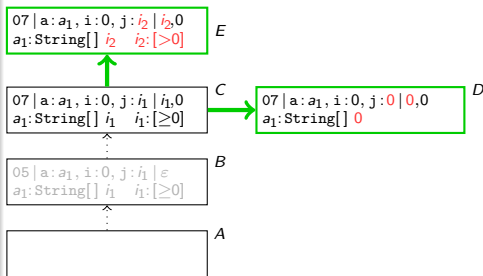
State C:

- Load i (0) and j (i_1) to operand stack
- `if_icmpge` cannot be evaluated

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



State C, D, E:

- Load i (0) and j (i_1) to operand stack
- `if_icmpge` cannot be evaluated yet:

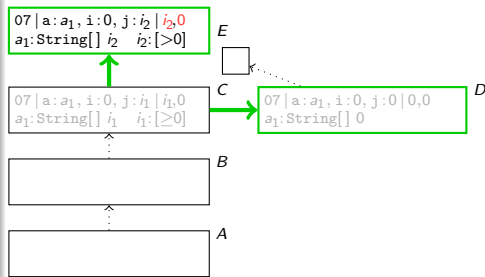
⇒ *Refine* information:

- In D , consider case $i_1 = 0$
- In E , consider case $i_1 > 0$


```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



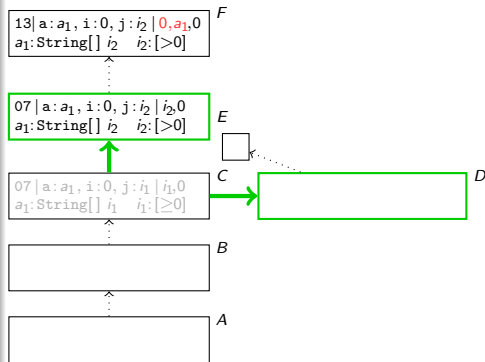
States D, E, F:

- D jumps to end of method

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
length():
00: aload_0
01: getfield count
04: ireturn

```



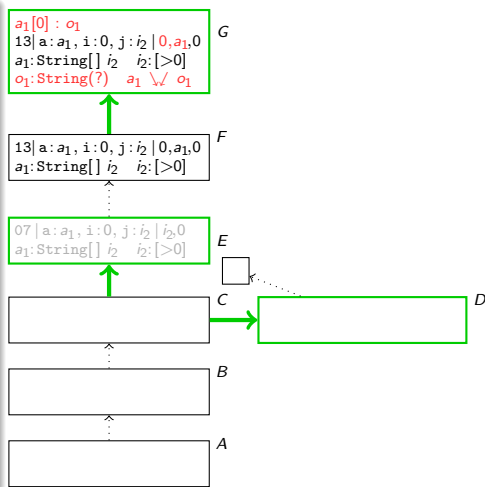
States *D, E, F*:

- *D* jumps to end of method
- *E* evaluates to *F*
- Loads array *a* (a_1) and index *i* (0) to stack
- Cannot evaluate *aa*load: $a_1[0]$ not known

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



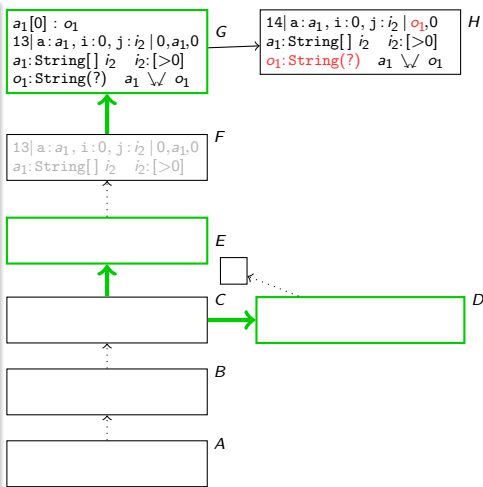
State G:

- *Refinement* of F
- o_1 created: null or unknown String
- $a_1 \surd o_1$: They *share*
- o_1 is value at $a_1[i_2]$

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



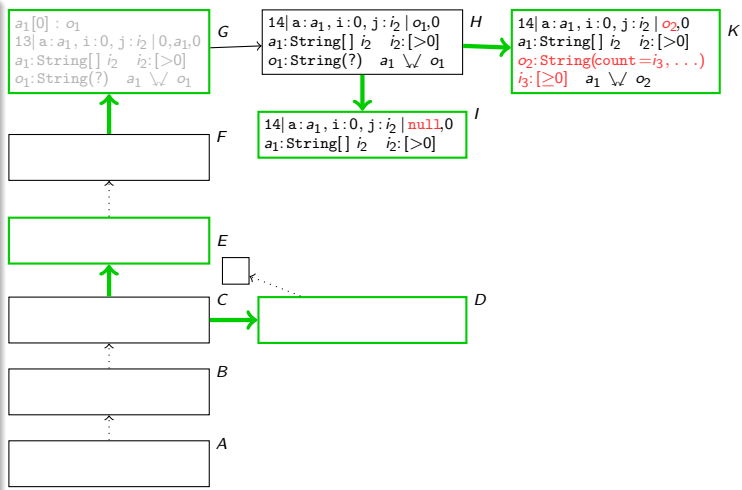
States *H* to *K*:

- *H* evaluated from *G*, loaded o_1 to stack
- invoke may throw `NullPointerException`

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



States H to K:

- *H* evaluated from *G*, loaded o_1 to stack
- invoke may throw `NullPointerException`

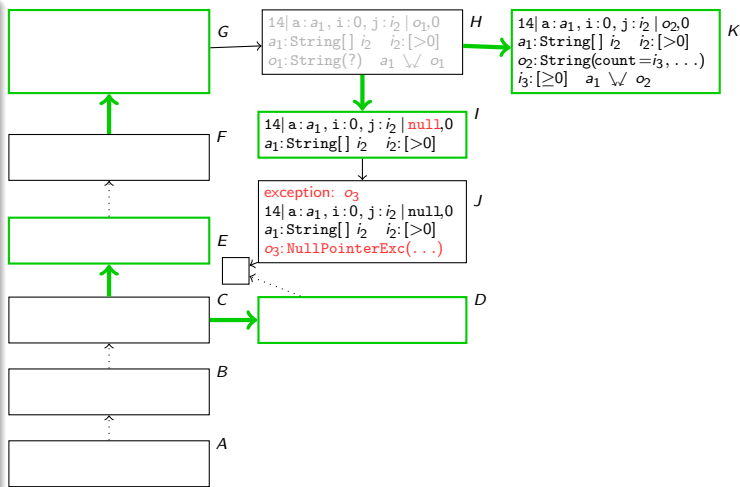
⇒ **Refinement:**

- *I*: Case o_1 is null
- *K*: Case o_1 is some object with fields

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



States *H* to *K*:

- *H* evaluated from *G*, loaded o_1 to stack
- invoke may throw `NullPointerException`

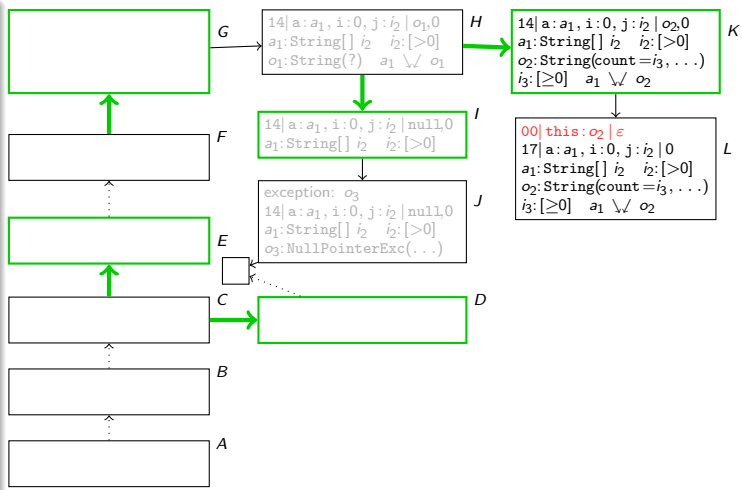
⇒ *Refinement*:

- *I*: Case o_1 is null (leads to NPE)
- *K*: Case o_1 is some object with fields

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



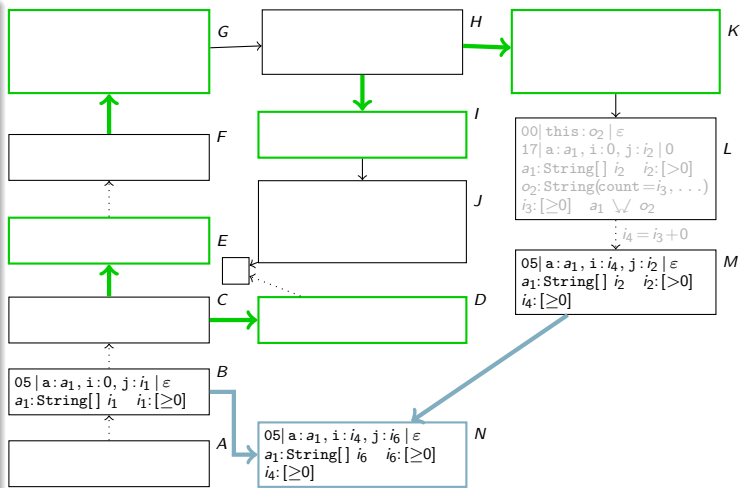
State L, M, N:

- Evaluation to L: New stack frame on top


```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
   length():
00: aload_0
01: getfield count
04: ireturn

```



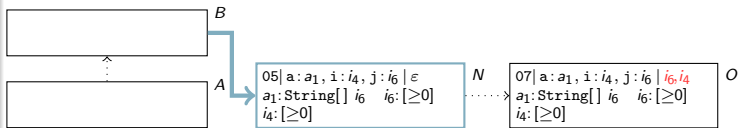
State L, M, N:

- Evaluation to L: New stack frame on top
- Evaluation to M: Retrieve length, add to i
- B and M similar: Merge states, get N
- N represents both B and M (*instances* of N)

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



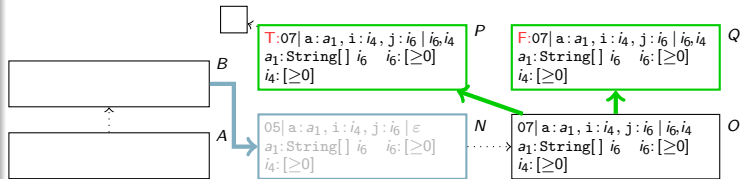
States O to S :

- Evaluate to O : Load i, j to stack

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



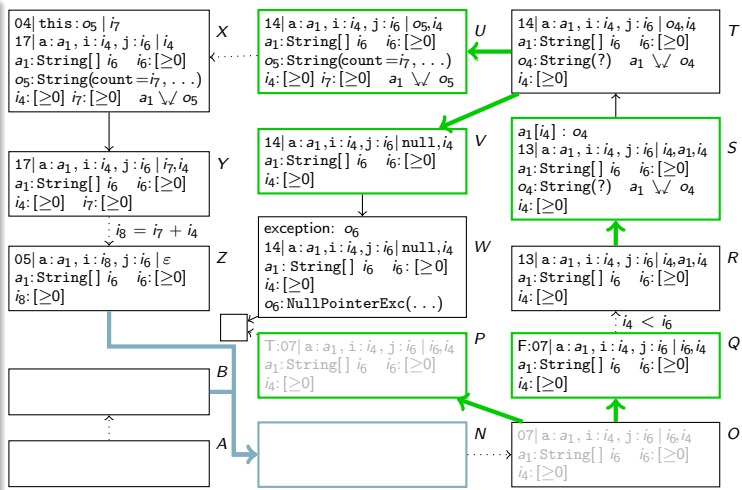
States O to S :

- Evaluate to O : Load i, j to stack
- Refine O to P, Q : Decide result of `if_icmpge`

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
    length():
00: aload_0
01: getfield count
04: ireturn

```



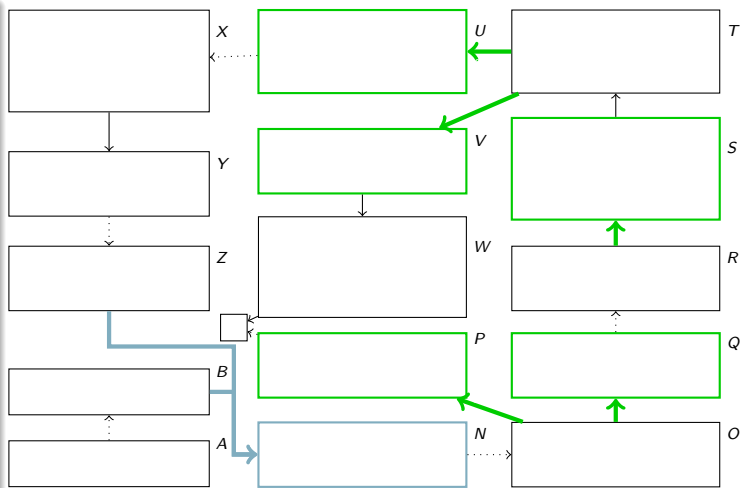
States O to S:

- Evaluate to O: Load i, j to stack
- Refine O to P, Q: Decide result of if_icmpge
- Rest as before

```

main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
  length():
00: aload_0
01: getfield count
04: ireturn

```



States O to S:

- Evaluate to O: Load i, j to stack
- Refine O to P, Q: Decide result of if_icmpge
- Rest as before
- All leaves program ends \Rightarrow construction finished

Termination graph construction

- 1 Choose (abstract) start state

Termination graph construction

- 1 Choose (abstract) start state
 - 2 Perform
 - symbolic evaluation
 - information **refinement**
 - **instantiation** (may need to *merge* states)
- until all leaves are program ends

Termination graph construction

- 1 Choose (abstract) start state
- 2 Perform
 - symbolic evaluation
 - information **refinement**
 - **instantiation** (may need to *merge* states)until all leaves are program ends
- 3 Recursion and modularity supported (cf. RTA'11)

- Termination graphs are *overapproximations* of all runs

Witness generation

- Termination graphs are *overapproximations* of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s ?

Witness generation

- Termination graphs are *overapproximations* of all runs
 - Let state e occur in graph starting in s
 - e really occurring in runs from s ?
- ⇒ Generate a witness run

Witness generation

- Termination graphs are *overapproximations* of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s ?

⇒ Generate a witness run

Definition

w *witness* for $e \Leftrightarrow$ all runs from w lead to e

Witness generation

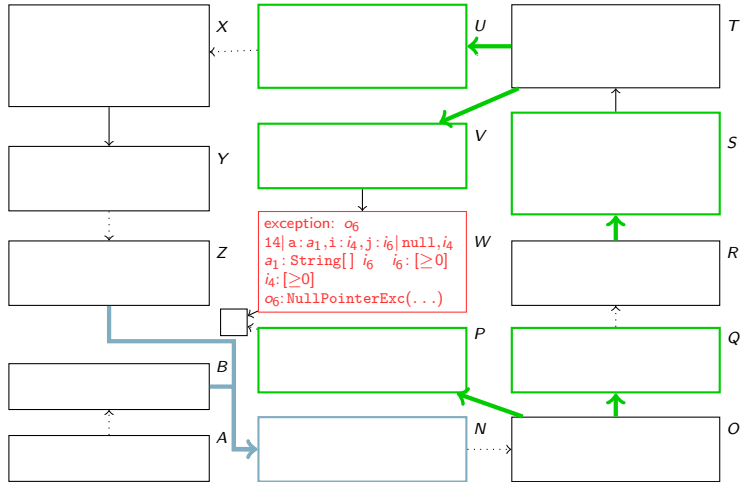
- Termination graphs are *overapproximations* of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s ?

⇒ Generate a witness run

Definition

w *witness* for $e \Leftrightarrow$ all runs from w lead to e

Re-use graph: Traverse edges backwards



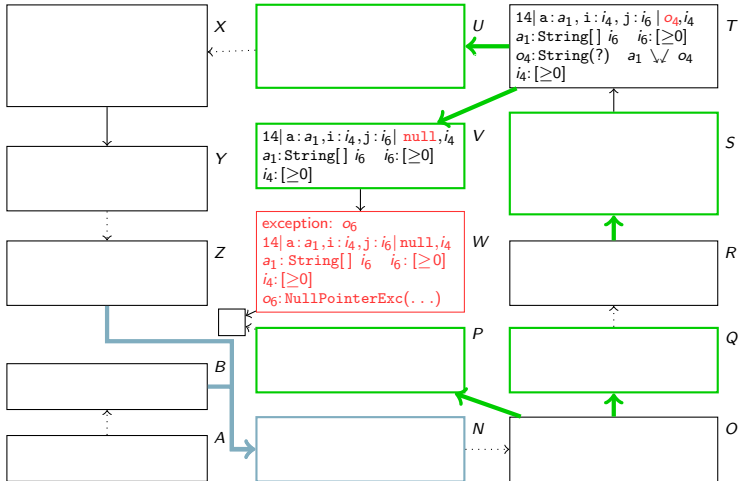
- W : program ends with a NullPointerException

⇒ Find witness

Witness for W

```
exception:  $\sigma_6$   
14 | a:  $a_1, i: i_4, j: i_6$  | null,  $i_4$   
 $a_1: \text{String}[]$   $i_6$   $i_6: [\geq 0]$   
 $i_4: [\geq 0]$   
 $\sigma_6: \text{NullPointerException}(\dots)$ 
```

```
14 | a:  $a_1, i: i_4, j: i_6$  | null,  $i_4$   
 $a_1: \text{String}[]$   $i_6$   $i_6: [\geq 0]$   
 $i_4: [\geq 0]$ 
```



- V evaluated to W

⇒ Reverse evaluation step

- T refined to V

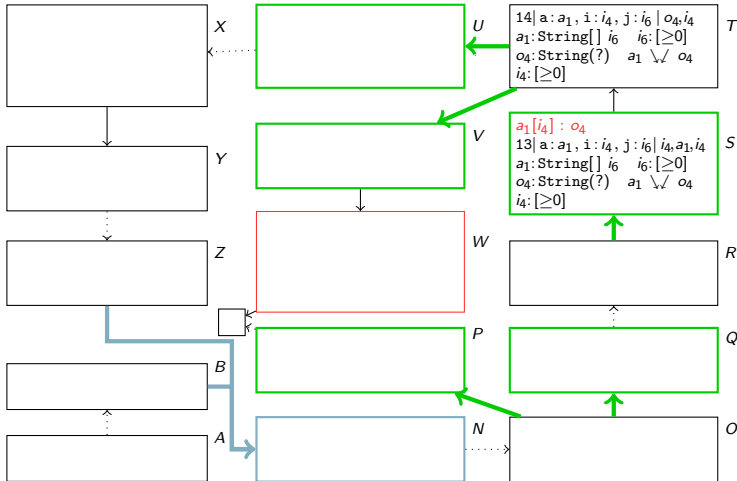
Witness for W

exception: o_6
 $14 \mid a: a_1, i: i_4, j: i_6 \mid \text{null}, i_4$
 $a_1: \text{String}[] \ i_6 \ i_6: [\geq 0]$
 $i_4: [\geq 0]$
 $o_6: \text{NullPointerException}(\dots)$

$14 \mid a: a_1, i: i_4, j: i_6 \mid \text{null}, i_4$
 $a_1: \text{String}[] \ i_6 \ i_6: [\geq 0]$
 $i_4: [\geq 0]$

$14 \mid a: a_1, i: i_4, j: i_6 \mid \text{null}, i_4$
 $a_1: \text{String}[] \ i_6 \ i_6: [\geq 0]$
 $i_4: [\geq 0]$

$a_1[i_4]: \text{null}$
 $13 \mid a: a_1, i: i_4, j: i_6 \mid i_4, a_1, i_4$
 $a_1: \text{String}[] \ i_6 \ i_6: [\geq 0]$
 $i_4: [\geq 0]$



- V evaluated to W
- ⇒ Reverse evaluation step
- T refined to V
- ⇒ Re-use current witness
- S evaluated to T : Reverse evaluation

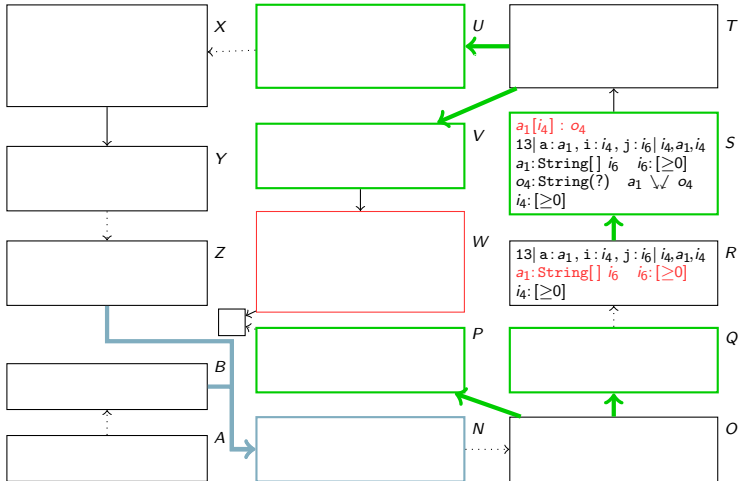
Witness for W

exception: o_6
 14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$
 $o_6: \text{NullPointerException}(\dots)$

14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$

14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$

$a_1[i_4]: \text{null}$
 13 | $a: a_1, i: i_4, j: i_6$ | i_4, a_1, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$



- R refined to S : Re-use current witness

Witness for W

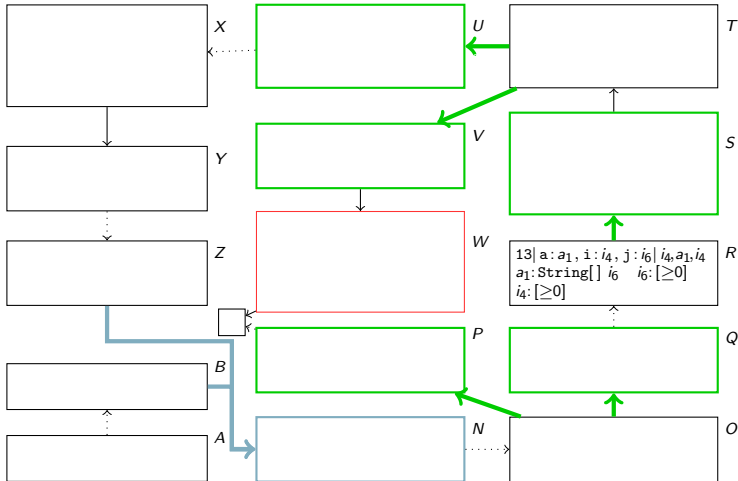
exception: o_6
 14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$
 $o_6: \text{NullPointerException}(\dots)$

14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$

14 | $a: a_1, i: i_4, j: i_6$ | null, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$

$a_1[i_4]: \text{null}$
 13 | $a: a_1, i: i_4, j: i_6$ | i_4, a_1, i_4
 $a_1: \text{String}[]$ i_6 $i_6: [\geq 0]$
 $i_4: [\geq 0]$

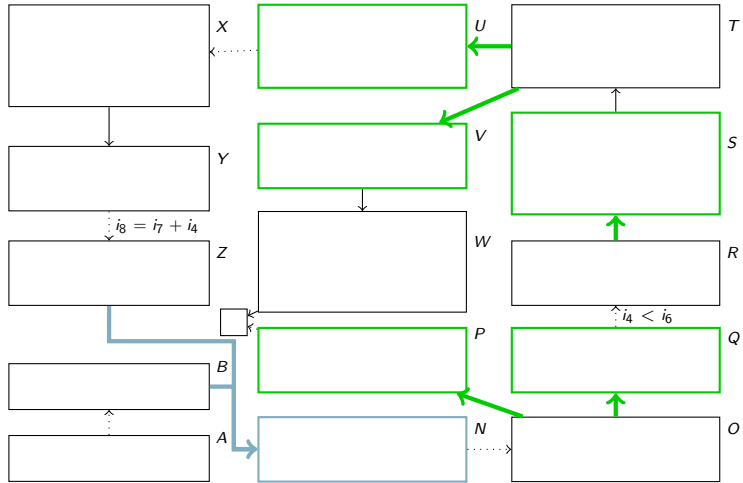
13 | $a: a_2, i: 0, j: 1$ | $Q, a_2, 0$
 $a_2: \text{String}[]$ {null}



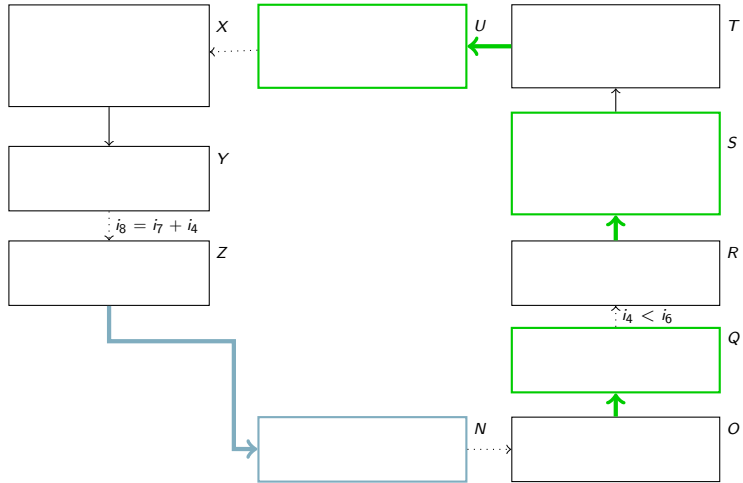
- R refined to S : Re-use current witness - **No!**

- a_1 too abstract to keep information

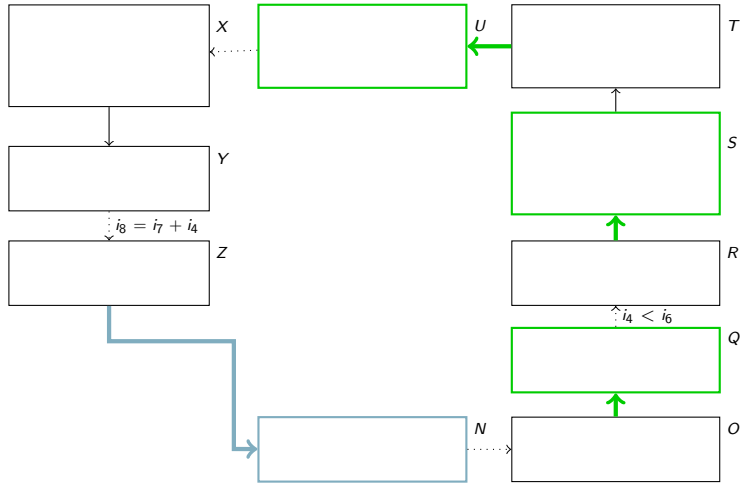
⇒ Instantiate array and index by concrete values



- Goal: Find state that is visited again and again

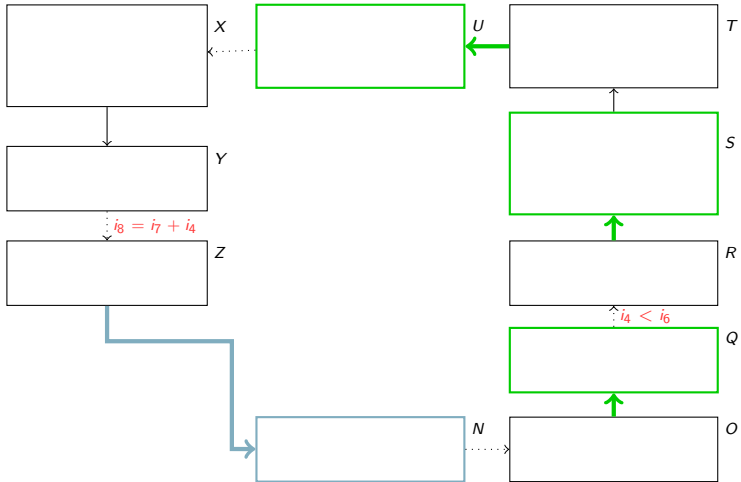


- Goal: Find state that is visited again and again
- ⇒ Only consider cycles
- Heuristic: Integers: Find values by SMT
Objects: Find values by witness generation



- Goal: Find state that is visited again and again
- ⇒ Only consider cycles
- Heuristic:
 - Integers: Find values by SMT
 - Objects: Find values by witness generation
- Verification: By symbolic evaluation

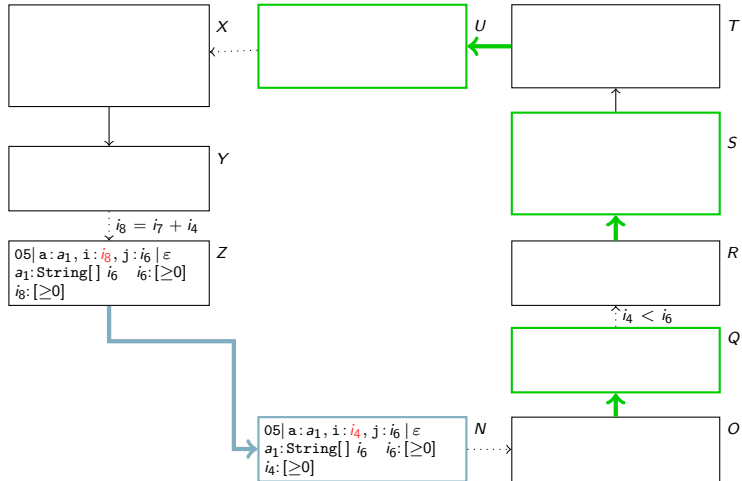
$$i_4 < i_6$$
$$\wedge i_8 = i_7 + i_4$$



Find values by SMT

- Use conditions on edges

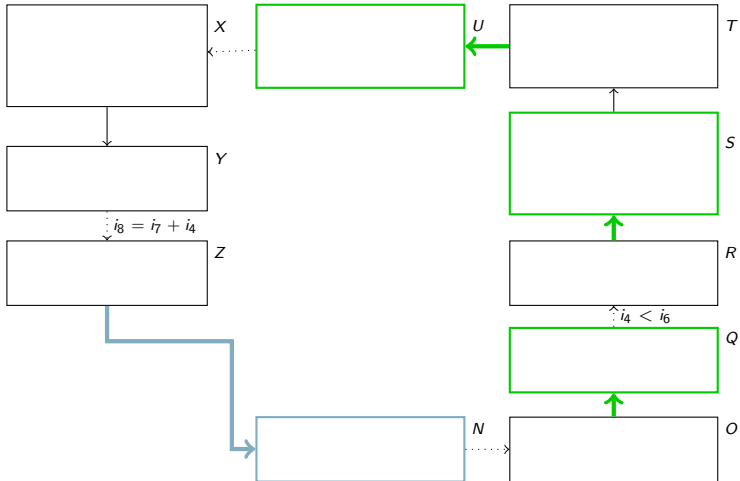
$$\begin{aligned}
 & i_4 < i_6 \\
 \wedge & i_8 = i_7 + i_4 \\
 \wedge & i_4 = i_8
 \end{aligned}$$



Find values by SMT

- Use conditions on edges
 - **Refinement**, **Instance** renamings \Rightarrow Equalities
- \Rightarrow Last instance edge \sim values stay the same

$$\begin{aligned}
 & i_4 < i_6 \\
 \wedge & i_8 = i_7 + i_4 \\
 \wedge & i_4 = i_8
 \end{aligned}$$



Find values by SMT

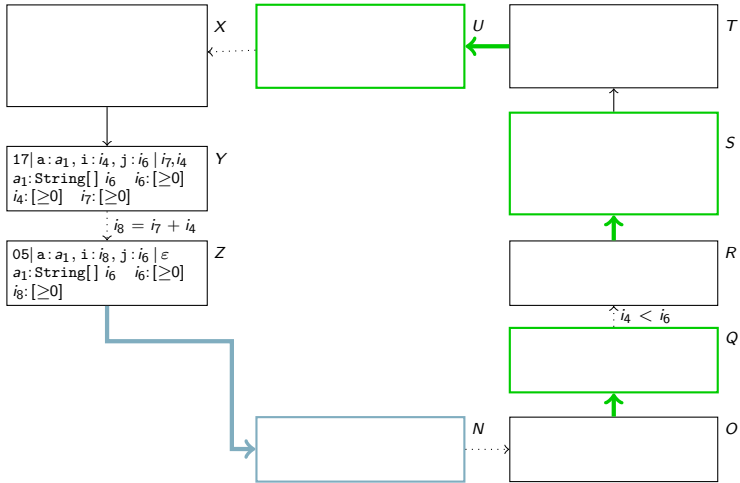
- Use conditions on edges
 - **Refinement**, **Instance** renamings \Rightarrow Equalities
- \Rightarrow Last instance edge \sim values stay the same

SMT Solution:

$$\begin{aligned}
 i_4 &= i_7 = i_8 = 0 \\
 i_6 &= 1
 \end{aligned}$$

Witness

```
05| a: a1, i: 0, j: 1 | ε  
a1: String[] 1
```



Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation

SMT Solution:

$$i_4 = i_7 = i_8 = 0$$

$$i_6 = 1$$

Witness

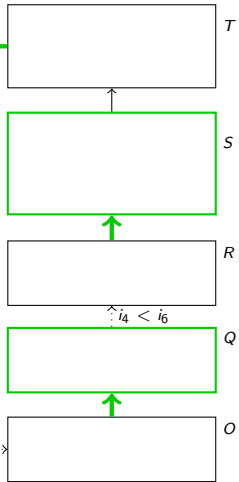
```
05| a: a1, i: 0, j: 1 | ε  
a1: String[] 1
```

```
17| a: a1, i: 0, j: 1 | 0, 0  
a1: String[] 1
```

```
04| this: o5 | i7  
17| a: a1, i: i4, j: i6 | i4  
a1: String[] i6 i6: [≥ 0]  
o5: String(count = i7, ...) |  
i4: [≥ 0] i7: [≥ 0] a1 √ o5
```

```
17| a: a1, i: i4, j: i6 | i7, i4  
a1: String[] i6 i6: [≥ 0]  
i4: [≥ 0] i7: [≥ 0]
```

$$i_8 = i_7 + i_4$$



Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible

SMT Solution:

$$i_4 = i_7 = i_8 = 0$$

$$i_6 = 1$$

Witness

```
05 | a: a1, i: 0, j: 1 | ε  
a1: String[] 1
```

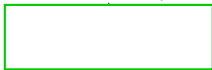
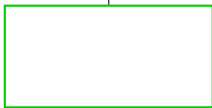
```
17 | a: a1, i: 0, j: 1 | Q0  
a1: String[] 1
```

```
04 | this: o5 | 0  
17 | a: a1, i: 0, j: 1 | 0  
a1: String[] 1 a1 √ o5  
o5: String(count=0, ...)
```

```
04 | this: o5 | i7  
17 | a: a1, i: i4, j: i6 | i4  
a1: String[] i6 i6: [≥0]  
o5: String(count=i7, ...)  
i4: [≥0] i7: [≥0] a1 √ o5
```

```
17 | a: a1, i: i4, j: i6 | i7, i4  
a1: String[] i6 i6: [≥0]  
i4: [≥0] i7: [≥0]
```

$$i_8 = i_7 + i_4$$



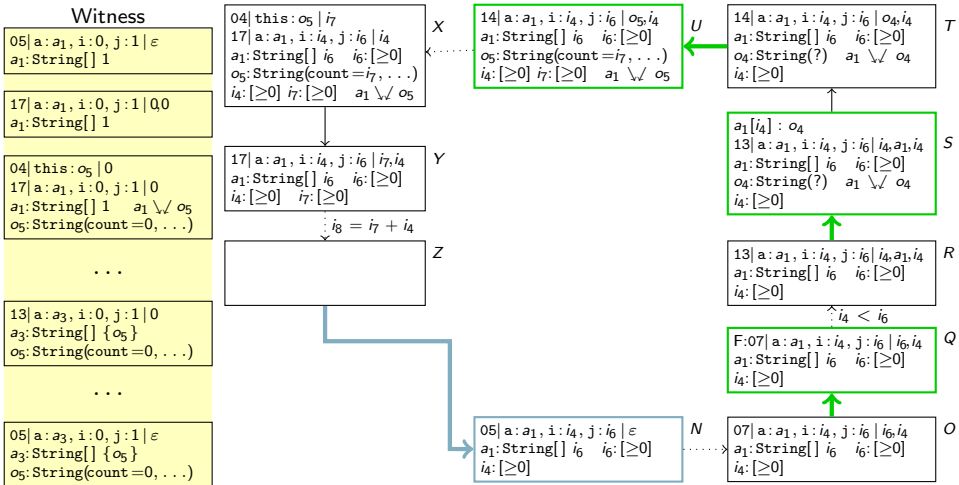
Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible

SMT Solution:

$$i_4 = i_7 = i_8 = 0$$

$$i_6 = 1$$



Find values by witness generation

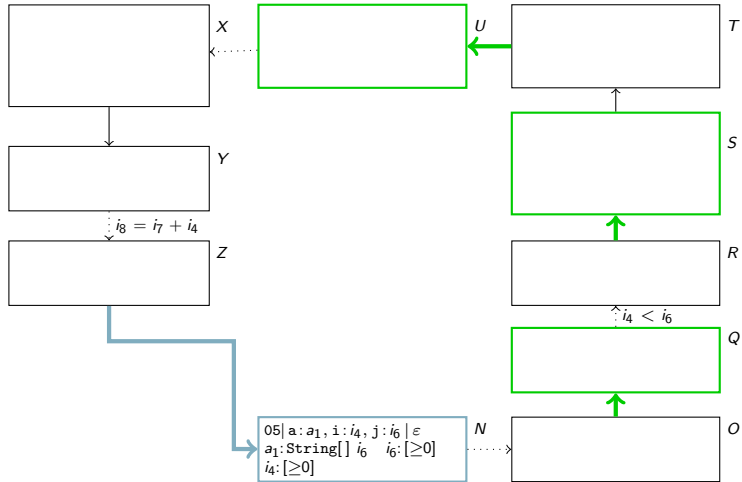
- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible
- Continue until state at loop start

SMT Solution:

$$i_4 = i_7 = i_8 = 0$$

$$i_6 = 1$$

Witness



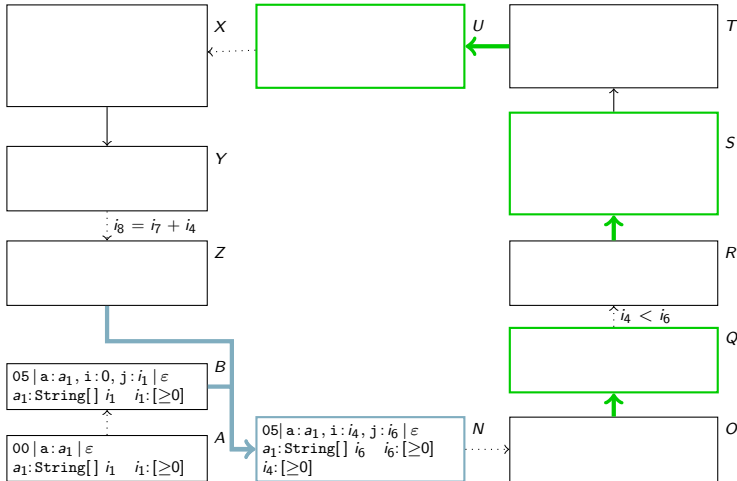
```
05 | a: a3, i: 0, j: 1 | ε  
a3: String[] {o5}  
o5: String(count=0, ...)
```

```
05 | a: a1, i: i4, j: i6 | ε  
a1: String[] i6 i6: [≥0]  
i4: [≥0]
```

Verification

- Check that object values stay the same:
⇒ Perform symbolic evaluation.

Witness



Verification

- Check that object values stay the same:
- ⇒ Perform symbolic evaluation.

Witness

- Find start state witness


```
static void nonLoop(  
int x, int y) {  
    if (y >= 0) {  
        1 while(x >= y) {  
            int z = x - y;  
            2 if (z > 0) {  
                3 x--;  
            } else {  
                4 x = 2*x + 1;  
                y++; }  
        }  
    }  
}
```

- nonLoop does not terminate for $x = 1, y = 1$:

$(1, 1) \rightarrow (3, 2) \rightarrow (2, 2) \rightarrow (5, 3) \rightarrow (4, 3) \rightarrow (3, 3) \rightarrow \dots$

```
static void nonLoop(  
int x, int y) {  
    if (y >= 0) {  
        1 while(x >= y) {  
            int z = x - y;  
            2 if (z > 0) {  
                3 x--;  
            } else {  
                4 x = 2*x + 1;  
                y++; }  
        }  
    }  
}
```

- nonLoop does not terminate for $x = 1, y = 1$:
- Program run is *non-periodic*:

$(1, 1) \xrightarrow{4} (3, 2) \xrightarrow{3} (2, 2) \xrightarrow{4} (5, 3) \xrightarrow{3} (4, 3) \xrightarrow{3} (3, 3) \xrightarrow{4} \dots$

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
    2 if (z > 0) {
    3 x--;
      } else {
    4 x = 2*x + 1;
      y++; }}}}

```

↓

1	$x: i_1, y: i_2$	A
	$i_1: \mathbb{Z} \quad i_2: [\geq 0]$	

- nonLoop does not terminate for $x = 1, y = 1$:
- Program run is *non-periodic*:

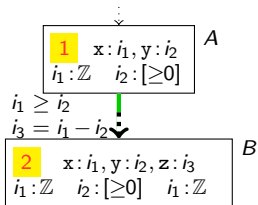
$(1, 1) \xrightarrow{4} (3, 2) \xrightarrow{3} (2, 2) \xrightarrow{4} (5, 3) \xrightarrow{3} (4, 3) \xrightarrow{3} (3, 3) \xrightarrow{4} \dots$

- Always: Generate Termination Graph


```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
          y++; }}}}

```

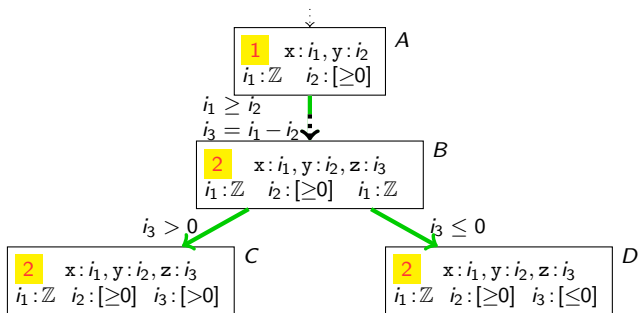


- A to B : Refine + Evaluation

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
          y++; }}}}

```

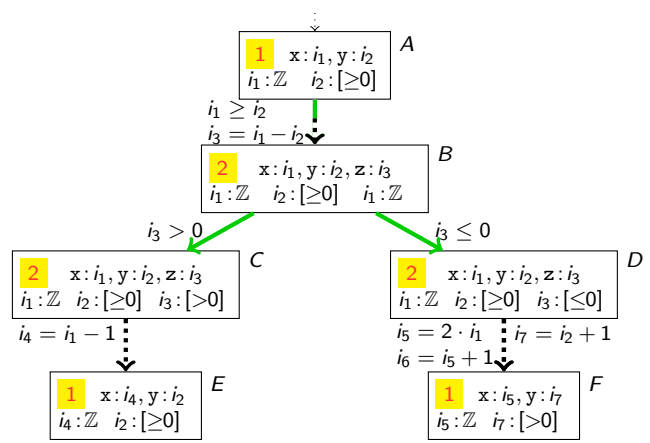


- A to B: Refine + Evaluation
- B to C, D: Refine z/i_3

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }

```

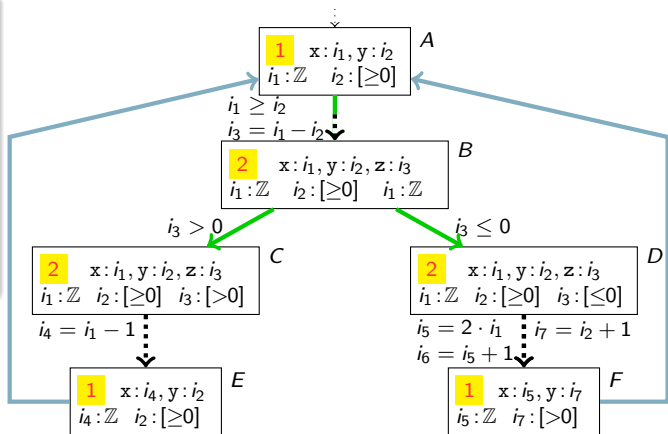


- A to B: Refine + Evaluation
- B to C, D: Refine z/i₃
- C to E, D to F: Evaluation

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } }
  }
}

```

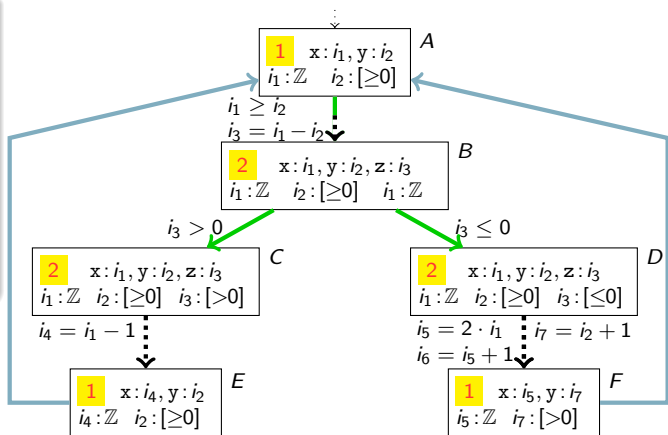


- A to B: Refine + Evaluation
- B to C, D: Refine z/i_3
- C to E, D to F: Evaluation
- E, F represented by A: Instantiation

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }
  }
}

```

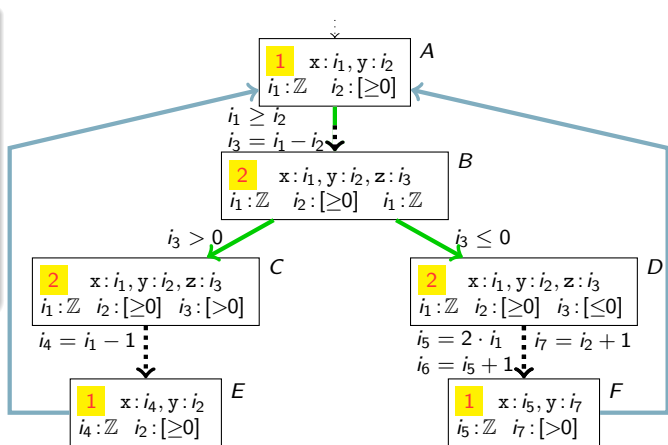


- A to B: Refine + Evaluation
- B to C, D: Refine z/i_3
- C to E, D to F: Evaluation
- E, F represented by A: Instantiation
- Prove (condition & computation) \rightarrow condition

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }

```



- Consider each path separately

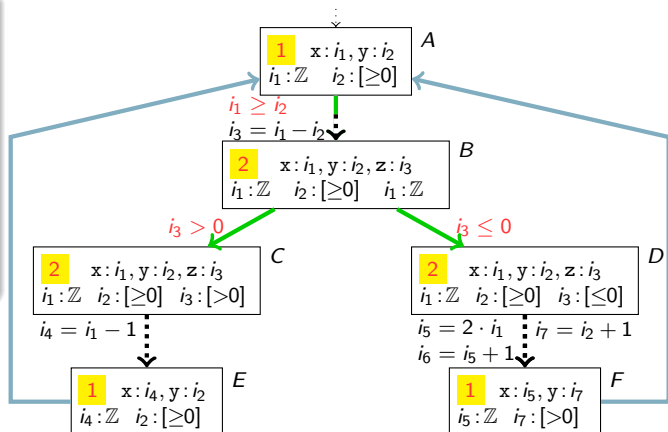
```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; }}}}

```

$\varphi_1: i_1 \geq i_2 \wedge i_3 > 0$

$\varphi_2: i_1 \geq i_2 \wedge i_3 \leq 0$

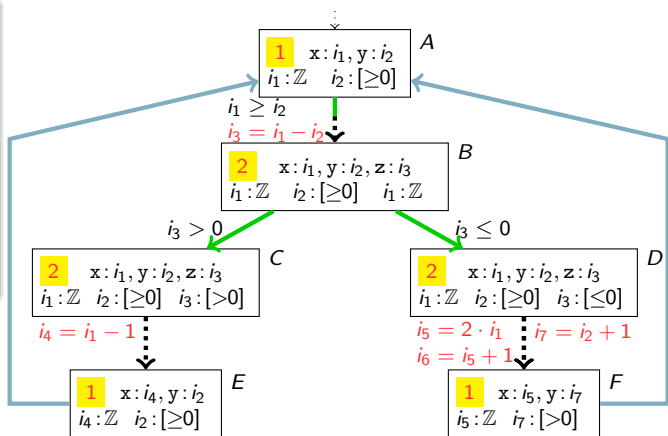


- Consider each path separately
- Condition formulas φ_i from refinements

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; }}}}

```



$$\varphi_1: i_1 \geq i_2 \wedge i_3 > 0$$

$$\varphi_2: i_1 \geq i_2 \wedge i_3 \leq 0$$

$$\psi_1: i_3 = i_1 - i_2 \wedge i_4 = i_1 - 1$$

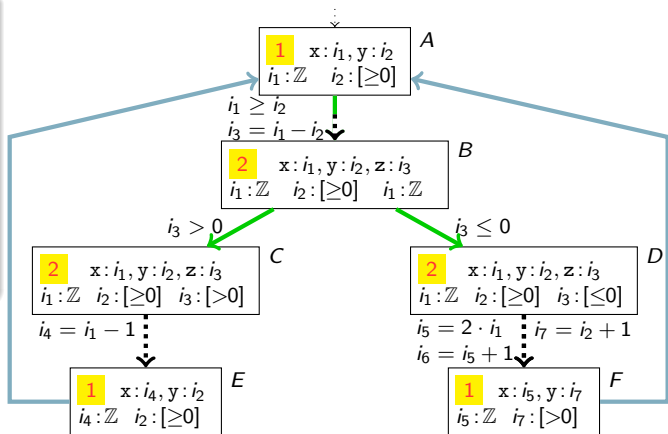
$$\psi_2: i_3 = i_1 - i_2 \wedge i_5 = 2 \cdot i_1 \\ \wedge i_6 = i_5 + 1 \wedge i_7 = i_2 + 1$$

- Consider each path separately
- Condition formulas φ_i from **refinements**
- Computation formulas ψ_i from evaluations


```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }
  }

```



$$\varphi_1^1: i_1^1 \geq i_2^1 \wedge i_3^1 > 0$$

$$\varphi_2^1: i_1^1 \geq i_2^1 \wedge i_3^1 \leq 0$$

$$\psi_1^1: i_3^1 = i_1 - i_2 \wedge i_4^1 = i_1 - 1$$

$$\psi_2^1: i_3^1 = i_1 - i_2 \wedge i_5^1 = 2 \cdot i_1^1$$

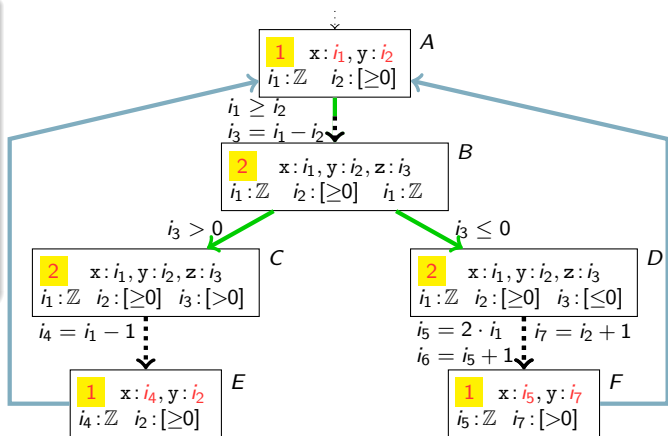
$$\wedge i_6^1 = i_5^1 + 1 \wedge i_7^1 = i_2^1 + 1$$

- Consider each path separately
- Condition formulas φ_i from **refinements**
- Computation formulas ψ_i from evaluations
- Label formulas for loop run k with k

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }
  }

```



$$\varphi_1^1: i_1^1 \geq i_2^1 \wedge i_3^1 > 0$$

$$\varphi_2^1: i_1^1 \geq i_2^1 \wedge i_3^1 \leq 0$$

$$\psi_1^1: i_3^1 = i_1 - i_2^1 \wedge i_4^1 = i_1^1 - 1$$

$$\psi_2^1: i_3^1 = i_1 - i_2^1 \wedge i_5^1 = 2 \cdot i_1^1 \\ \wedge i_6^1 = i_5^1 + 1 \wedge i_7^1 = i_2^1 + 1$$

$$\iota_1: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

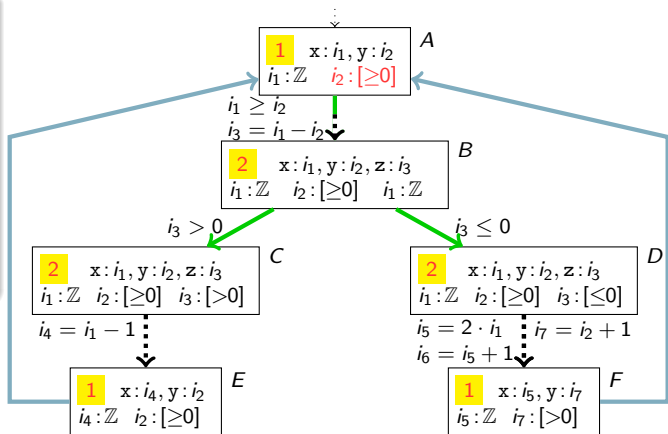
$$\iota_2: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

- Consider each path separately
- Condition formulas φ_i from **refinements**
- Computation formulas ψ_i from evaluations
- Label formulas for loop run k with k
- Connect runs using ι_j from **instantiations**

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; }}}}

```



$$\varphi_1^1: i_1^1 \geq i_2^1 \wedge i_3^1 > 0$$

$$\varphi_2^1: i_1^1 \geq i_2^1 \wedge i_3^1 \leq 0$$

$$\psi_1^1: i_3^1 = i_1 - i_2 \wedge i_4^1 = i_1 - 1$$

$$\psi_2^1: i_3^1 = i_1 - i_2 \wedge i_5^1 = 2 \cdot i_1^1$$

$$\wedge i_6^1 = i_5^1 + 1 \wedge i_7^1 = i_2^1 + 1$$

$$\iota_1: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

$$\iota_2: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

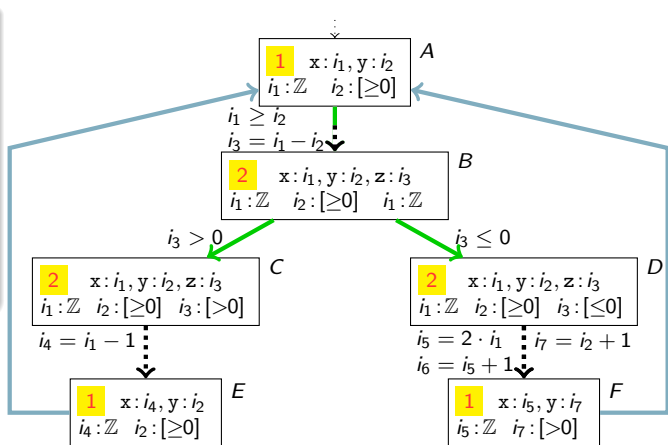
$$\mu: i_2^1 \geq 0$$

- Consider each path separately
- Condition formulas φ_i from **refinements**
- Computation formulas ψ_i from evaluations
- Label formulas for loop run k with k
- Connect runs using ι_j from **instantiations**
- Add invariant μ from loop start

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } } }
  }

```



- All conditions (φ_i^2) false \leadsto loop is left

$$\varphi_1^1: i_1^1 \geq i_2^1 \wedge i_3^1 > 0$$

$$\varphi_2^1: i_1^1 \geq i_2^1 \wedge i_3^1 \leq 0$$

$$\psi_1^1: i_3^1 = i_1 - i_2^1 \wedge i_4^1 = i_1^1 - 1$$

$$\psi_2^1: i_3^1 = i_1 - i_2^1 \wedge i_5^1 = 2 \cdot i_1^1 \\ \wedge i_6^1 = i_5^1 + 1 \wedge i_7^1 = i_2^1 + 1$$

$$\iota_1: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

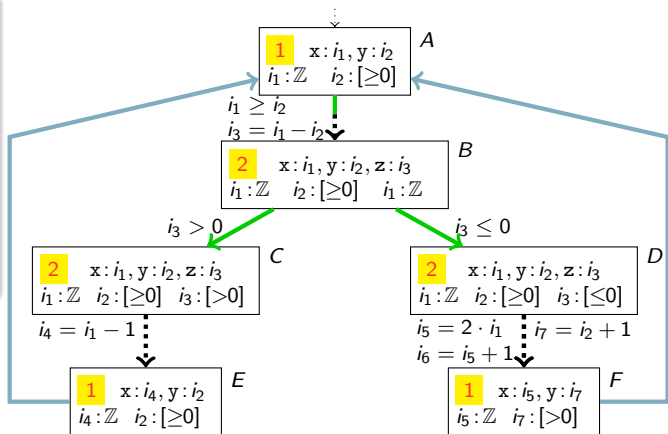
$$\iota_2: i_4^1 = i_1^2 \wedge i_2^1 = i_2^2$$

$$\mu: i_2^1 \geq 0$$

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; } } }
  }
}

```



$$\begin{aligned}
\varphi_1^1: & i_1^1 \geq i_2^1 \wedge i_3^1 > 0 \\
\varphi_2^1: & i_1^1 \geq i_2^1 \wedge i_3^1 \leq 0 \\
\psi_1^1: & i_3^1 = i_1 - i_2 \wedge i_4^1 = i_1 - 1 \\
\psi_2^1: & i_3^1 = i_1 - i_2 \wedge i_5^1 = 2 \cdot i_1 \\
& \wedge i_6^1 = i_5 + 1 \wedge i_7^1 = i_2 + 1 \\
\iota_1: & i_4^1 = i_2^1 \wedge i_2^1 = i_2^2 \\
\iota_2: & i_4^1 = i_1^1 \wedge i_2^1 = i_2^2 \\
\mu: & i_2^1 \geq 0
\end{aligned}$$

- All conditions (φ_j^2) false \leadsto loop is left
- Put together:

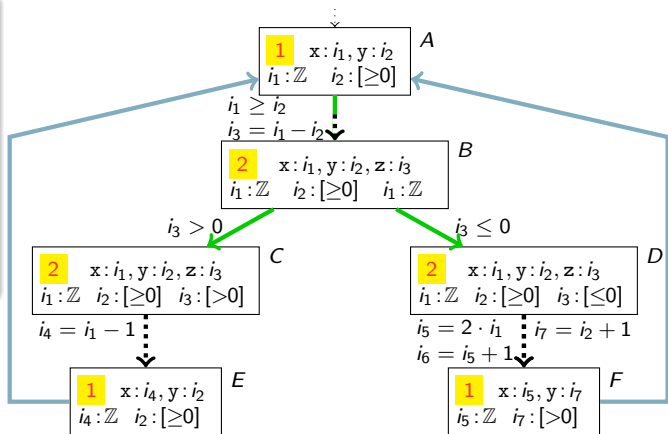
$$\underbrace{\mu}_{\text{invariants}} \wedge \underbrace{\left(\bigvee_{j=1}^n (\varphi_j^1 \wedge \psi_j^1 \wedge \iota_j) \right)}_{\text{first run through the loop}} \wedge \underbrace{\left(\bigwedge_{j=1}^n (\neg \varphi_j^2 \wedge \psi_j^2) \right)}_{\text{second run, leaving the loop}}$$

- SMT Solver says UNSAT \leadsto loop never left

```

static void nonLoop(
int x, int y) {
  if (y >= 0) {
    1 while(x >= y) {
      int z = x - y;
      2 if (z > 0) {
        3 x--;
      } else {
        4 x = 2*x + 1;
        y++; }}}}

```



$$\begin{aligned}
\varphi_1^1: i_1^1 &\geq i_2^1 \wedge i_3^1 > 0 \\
\varphi_2^1: i_1^1 &\geq i_2^1 \wedge i_3^1 \leq 0 \\
\psi_1^1: i_3^1 &= i_1 - i_2^1 \wedge i_4^1 = i_1^1 - 1 \\
\psi_2^1: i_3^1 &= i_1 - i_2^1 \wedge i_5^1 = 2 \cdot i_1^1 \\
&\wedge i_6^1 = i_5^1 + 1 \wedge i_7^1 = i_2^1 + 1 \\
\iota_1: i_4^1 &= i_2^1 \wedge i_2^1 = i_2^2 \\
\iota_2: i_4^1 &= i_1^2 \wedge i_2^1 = i_2^2 \\
\mu: i_2^1 &\geq 0
\end{aligned}$$

- All conditions (φ_j^2) false \leadsto loop is left
- Put together:

$$\underbrace{\mu}_{\text{invariants}} \wedge \underbrace{\left(\bigvee_{j=1}^n (\varphi_j^1 \wedge \psi_j^1 \wedge \iota_j)\right)}_{\text{first run through the loop}} \wedge \underbrace{\left(\bigwedge_{j=1}^n (\neg\varphi_j^2 \wedge \psi_j^2)\right)}_{\text{second run, leaving the loop}}$$

- SMT Solver says UNSAT \leadsto loop never left
- \Rightarrow Prove that one loop run occurs (SMT, witness)

Already done:

- Prove correctness
- Termination analysis

Already done:

- Prove correctness
- Termination analysis

Extensions:

- Complexity analysis
- Use other SMT theories
- Use source code annotations
- ...

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
 - 268 from the *Termination Problem Data Base*
 - 55 from the evaluation of Invel
 - 2 examples from our paper

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
 - 268 from the *Termination Problem Data Base*
 - 55 from the evaluation of Invel
 - 2 examples from our paper

	Invel Ex.					Other Ex.				
	Y	N	F	T	R	Y	N	F	T	R
AProVE-No	1	51	0	3	5	204	30	12	24	11
AProVE	1	0	5	49	54	204	0	27	39	15
Julia	1	0	54	0	2	166	22	82	0	4
Invel	0	42	13	0	?					

Yes, **N**o, **F**ailed proof, **T**imeout, **R**untime (average)

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
 - 268 from the *Termination Problem Data Base*
 - 55 from the evaluation of Invel
 - 2 examples from our paper

	Invel Ex.					Other Ex.				
	Y	N	F	T	R	Y	N	F	T	R
AProVE-No	1	51	0	3	5	204	30	12	24	11
AProVE	1	0	5	49	54	204	0	27	39	15
Julia	1	0	54	0	2	166	22	82	0	4
Invel	0	42	13	0	?					

Yes, **N**o, **F**ailed proof, **T**imeout, **R**untime (average)

<http://aprove.informatik.rwth-aachen.de/eval/JBC-Nonterm>