Symbolic Execution

Programs section, **Lecture 12**

Pavel Panchekha

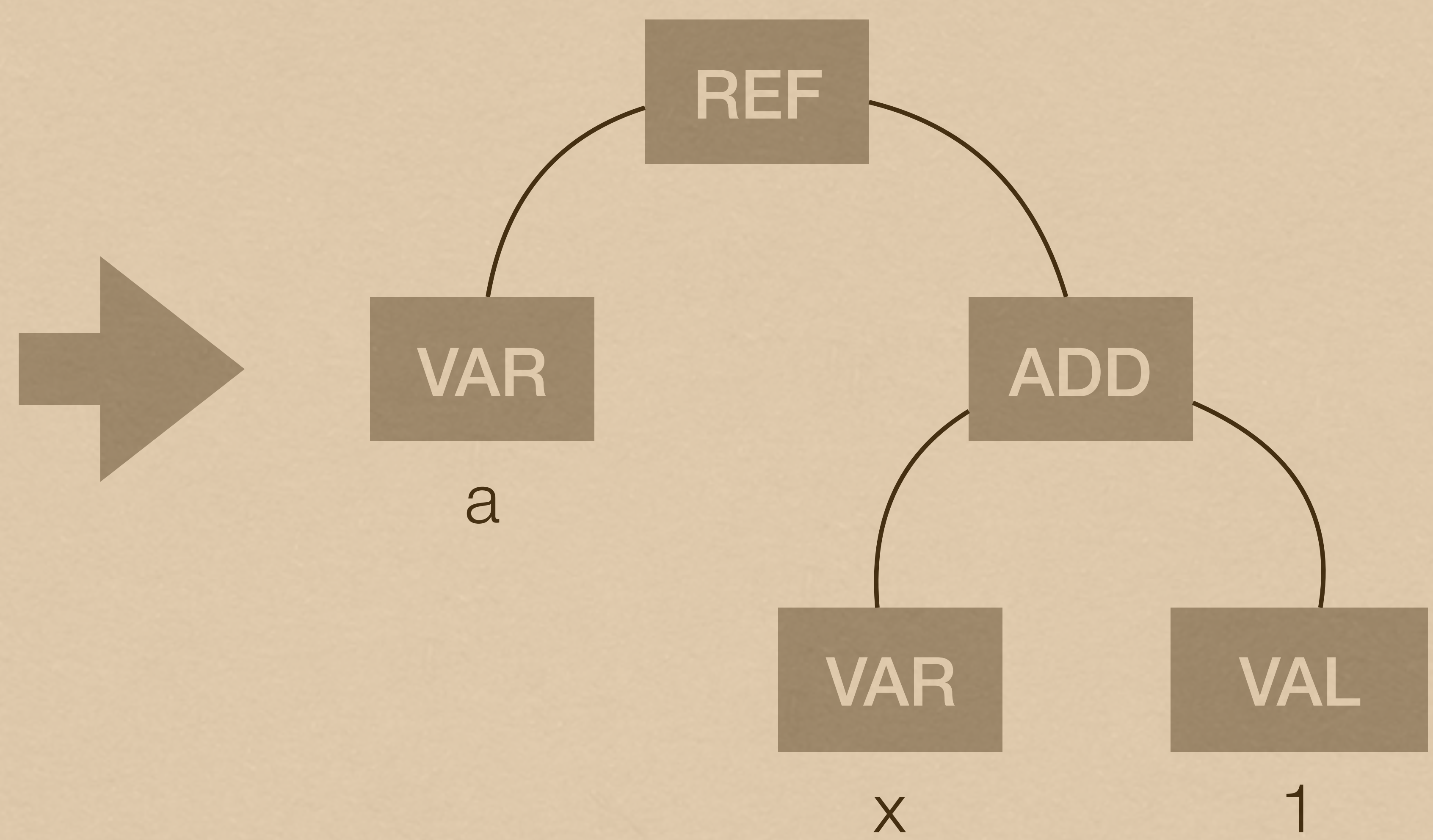
CS 6110, U of Utah

13 February 2020

Syntax

$$e, e_1, e_2 := e_1 + e_2 \mid e_1[e_2] \mid -e_1 \mid \cdots$$

$a[i + 1]$



Expression Semantics

$$\underbrace{\Gamma}_{\text{Environment}} \vdash \underbrace{e}_{\text{Expression}} \Downarrow \underbrace{v}_{\text{Value}}$$

```
def eval(env, expr):
  match expr:
    Add(expr1, expr2):
      eval(env, expr1) + eval(env, expr2)
    ...
```

Statement Semantics

$$\underbrace{\Gamma}_{\text{Environment}}, \underbrace{s}_{\text{Statement}} \rightsquigarrow \underbrace{\Gamma', s'}_{\text{New State}}$$

```
def run(stmt):
  env = initial_env()
  while stmt != HALT:
    env, stmt = step(env, stmt)
```

Class Progress

Logical reasoning

Program logics

Static analysis

Expressions

Statements

Procedures

Loops

Expressions

What does an expression **compute**?

Symbolic values and environments

Executing statements without running expressions

Evaluating statements with path conditions

Feasibility of program executions

Test generation and symbolic/concrete execution

Symbolic Expressions

Running expressions without running them

Evaluating Expressions

```
x = (2 << (2 << 8))
return x - x
```

0

```
x = (2 << (2 << n))
return x - x
```

0

```
y = x
return x - y
```

0

Symbolic Evaluation

Program expression → Logical formula

Convert program expressions to equivalent formulas

Program variables become logical variables

```
def sym_eval(expr): // Returns a z3 formula
  match expr:
    Add(expr1, expr2):
      return z3.Add(sym_eval(expr1), …)
    ...
```

Environments

Run statements, convert expressions

```
env : Map<Variable, Formula>
```

Step semantics don't change

But expression evaluation now symbolic

{ x: $3n+1$, y: $n-1$ } , (x -= y; s)

$\rightsquigarrow$

{ x: $2n+2$, y: $n-1$ } , s

Example

Execute; what is returned; what is the environment?

```
def do_something(x: Int, y: Int):
  z = x + y
  w = z - x
  return [1, 4, 8][w % 3]
```

$$[1,4,8][((x+y)-x) \textbf{ mod } 3]$$

{ x: x, y: y, z: $x+y$, w: $(x+y)-x$ }

Challenges

Logic **may not match** expressions in language

$$\texttt{s = arr[1:7]} \longrightarrow \underline{\textbf{slice}}(a, 1, 7)$$

$$\forall k, 0 \leq k < j - i \rightarrow \textbf{slice}(a, i, j)[k] = a[k + i]$$

Need **logical form** for each primitive operation

Summaries of library functions also helpful

$$\forall a, \forall x, \textbf{indexof}(a, x) \neq -1 \rightarrow a[\textbf{indexof}(a, x)] = x$$

Path Conditions

Executing statements to track control flow

Conditionals

What does the following code **return**?

```
def min(x: Int, y: Int):
  z = x
  if (x < y):
    z = x
  if (y < x):
    z = y
  return z
```

How do we handle **conditionals, symbolically**?

Path Conditions

Symbolic

$$\frac{\Gamma \vdash e \Downarrow \top}{\Gamma, \textbf{if}\ (e)\ \{s_t\}\ \textbf{else}\ \{s_f\} \rightsquigarrow \Gamma, s_t} \qquad \frac{\Gamma \vdash e \Downarrow \bot}{\Gamma, \textbf{if}\ (e)\ \{s_t\}\ \textbf{else}\ \{s_f\} \rightsquigarrow \Gamma, s_f}$$

Symbolic evaluation **doesn't** produce true or false

Try both ways, **record** symbolic conditions used

```
step(env, stmt) → List<(Env, Stmt, Formula)>
```

$$\Gamma, \textbf{if}\ (e)\ \{s_t\}\ \textbf{else}\ \{s_f\}, p \rightsquigarrow \begin{bmatrix} \Gamma, s_t, p \wedge e(\Gamma) \\ \Gamma, s_f, p \wedge \neg e(\Gamma) \end{bmatrix}$$

Path Conditions

```
if (x < y):

  z = x

if (y < x):

  z = y
```

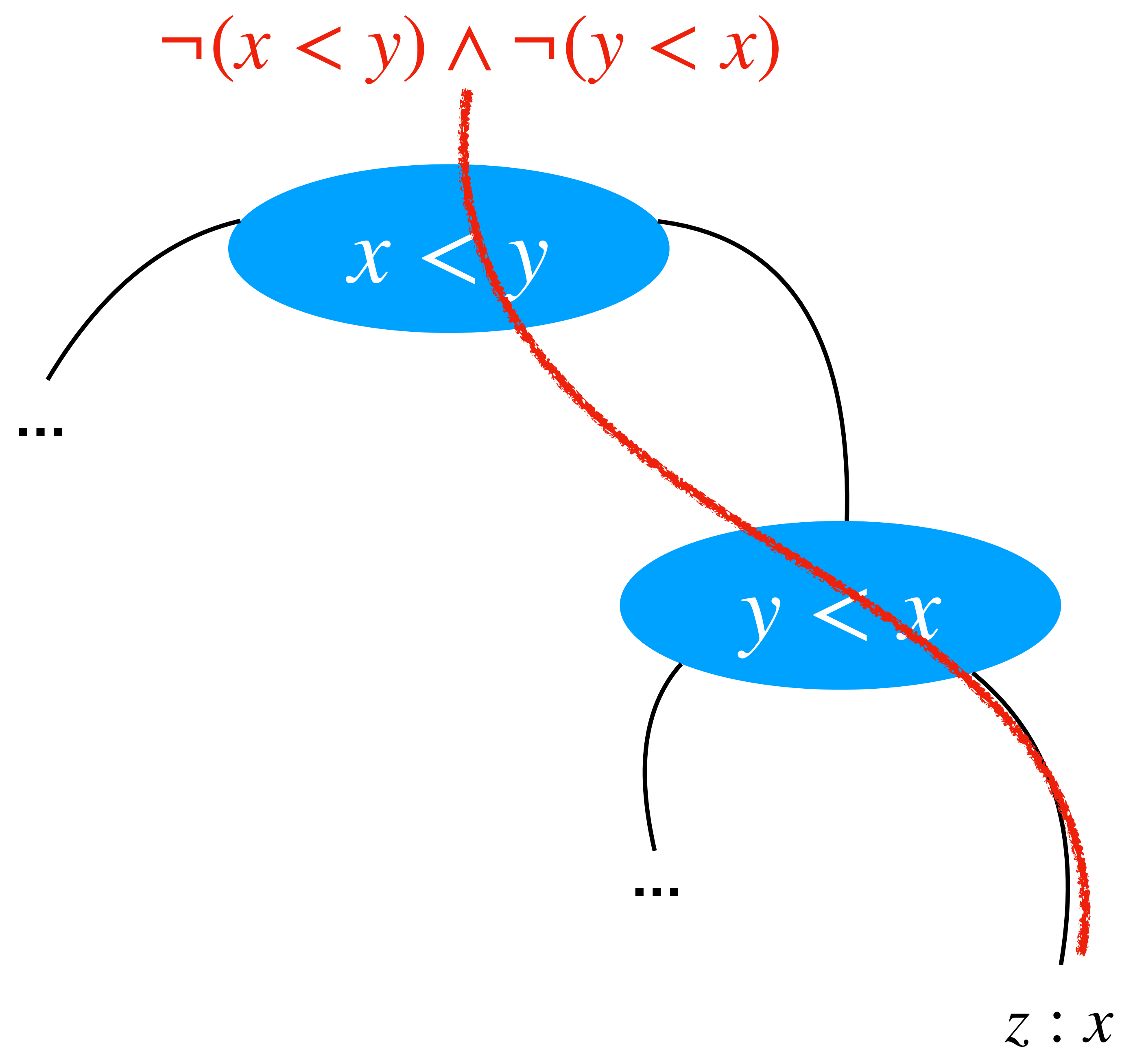
$$\{x : x, y : y\}, \top$$

$$\{x : x, y : y\}, x < y \qquad \{x : x, y : y\}, \neg(x < y)$$

$$\{x : x, y : y, z : x\}, x < y$$

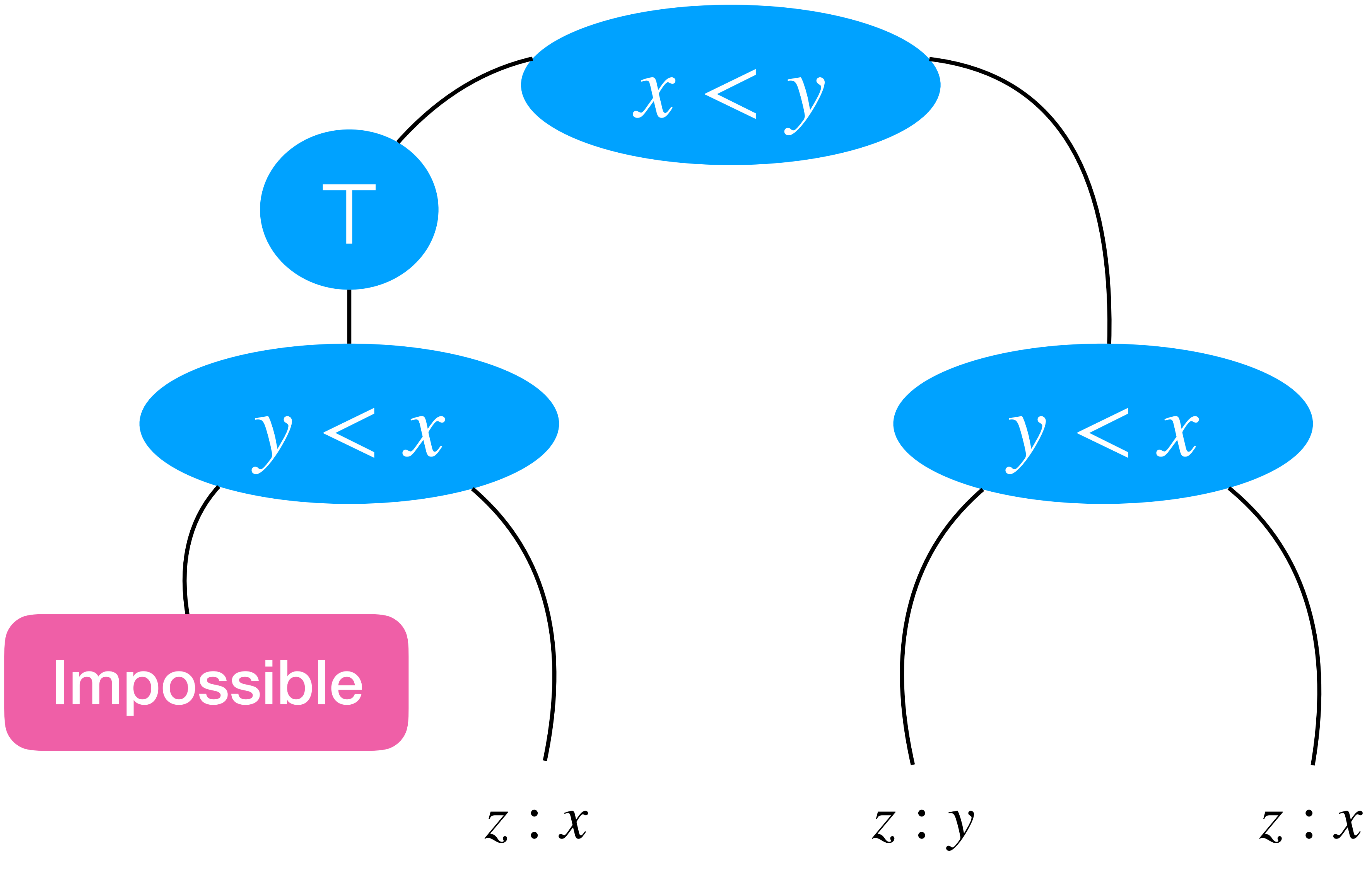
Path Conditions

```
if (x < y):
  z = x
if (y < x):
  z = y
```



Path Conditions

```
if (x < y):
  z = x
if (y < x):
  z = y
```



Feasibility

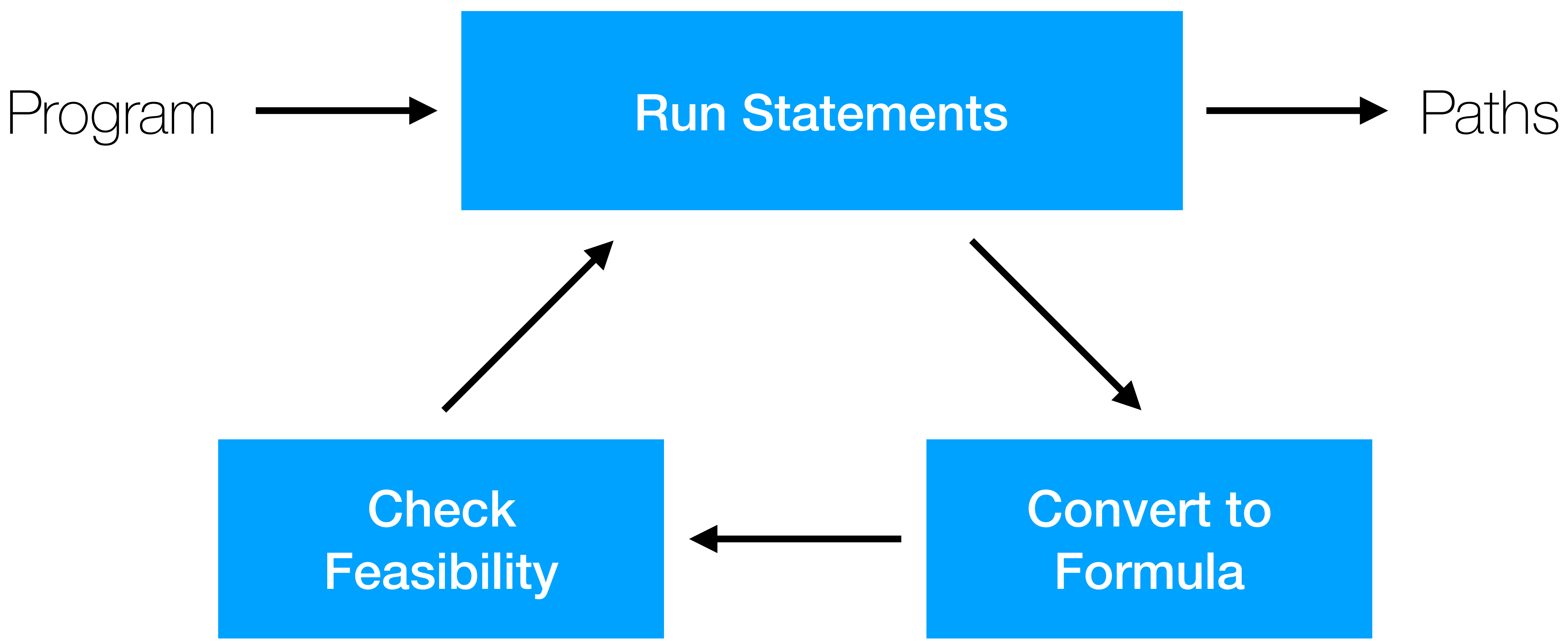
Path condition may be **impossible** to meet

- **No program execution** could take that program path
- Important to **discard** those paths; avoid exponential growth

To symbolically execute "Γ, if (e) $\{s_t\}$ else $\{s_f\}$, p":

- Check $p \land e(\Gamma)$; if so execute Γ, s_t, p'
- Check $p \land \neg e(\Gamma)$; if so execute Γ, s_f, p'

Architecture



Exercise

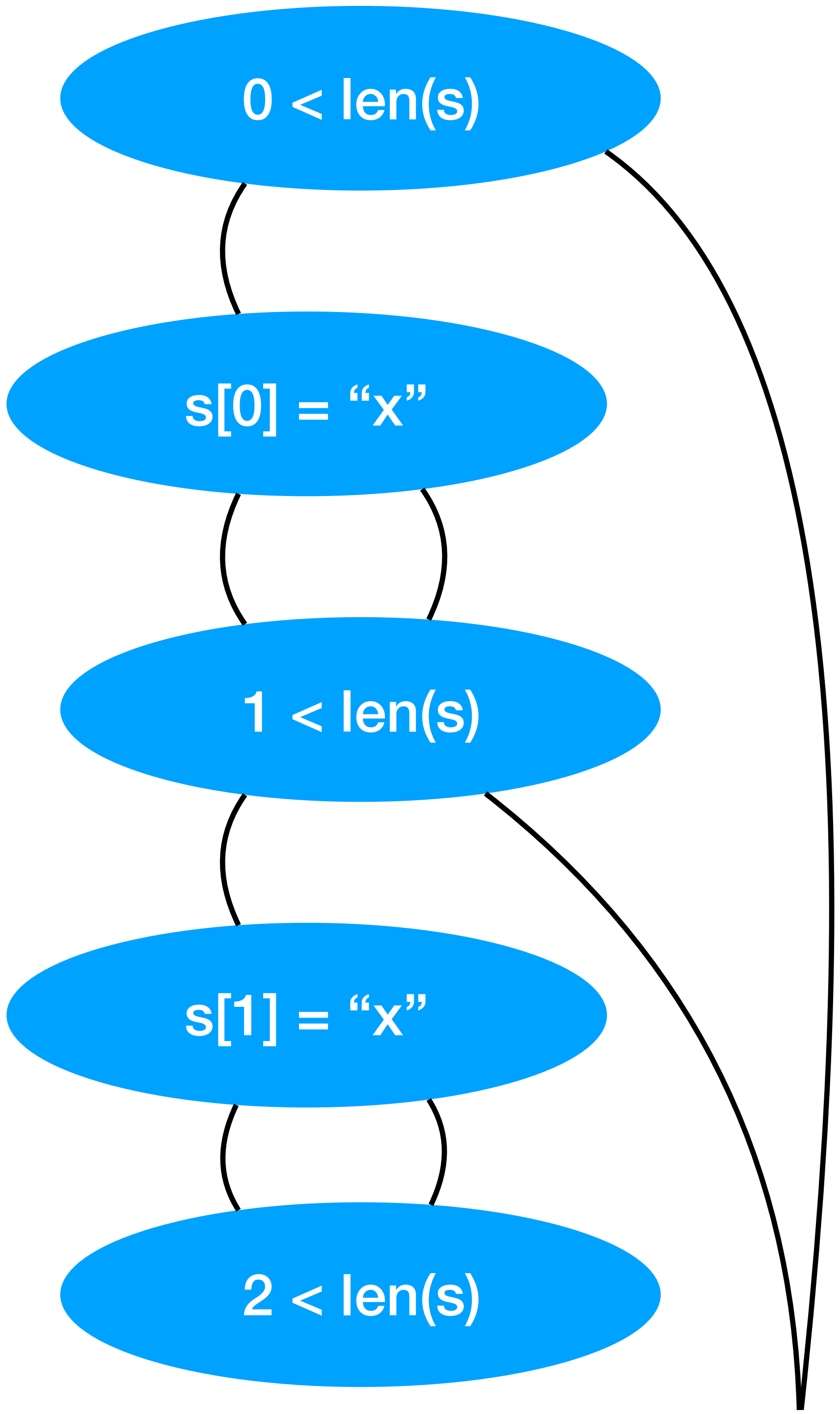
```
def which(x, y, z, target):
  i = 0
  a = [x, y, z]
  while i < 3:
    if a[i] == target:
      return i
    i = i + 1
  return -1
```

Draw the path tree. **How many paths** are there?

What is the **path condition** and **return value** for each

Merge vs Split

```
def has_x(s: String):
  i = 0
  out = False
  while i < len(s):
    if s[i] == “x”:
      out = True
    i = i + 1
  return out
```



Merge vs Split

Length k means 2^k paths

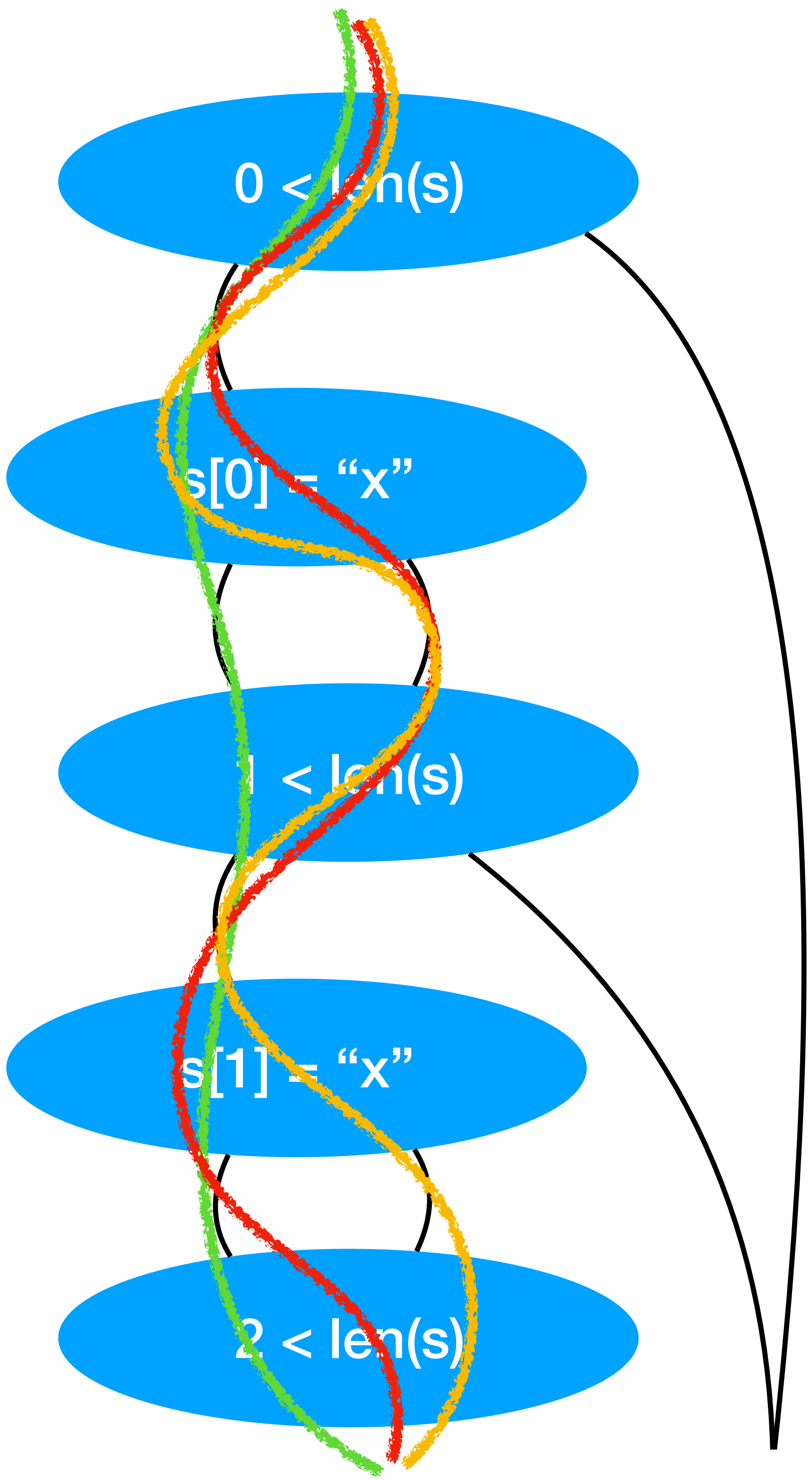
But how different are they?

out = $\top$ **out =** $\top$ **out =** $\top$

Many paths, **same** environment

Merge paths; $k + 1$ left (here)

If $\forall s, \Gamma_1 = \Gamma_2$, keep one with path $p_1 \vee p_2$



Course Updates

Proposal Presentations

Presentations

Project proposal presentations on Monday

- 8 minutes for groups, 4 minutes for solo
- **Upload slides in PDF format** for fast switching

For groups:

- Motivation
- Goal
- Approach

For solo:

- Syntax
- Design choices
- Example code

Presentation Tips

Practice! It takes 8/4 minutes. If stuck, change slides!

- Slides aren't set in stone! **Practice**, then change slides.
- If presenting together, **practice together**

Examples and demos > explanations or summaries

Use Keynote, Powerpoint, or Google Docs

- Beamer, etc encourage bad presentation habits

Happy to give **personalized feedback** on request!

Assignment 3

Analyze GNU echo using symbolic execution tool KLEE

1. Find all **valid flags**
2. Find **hidden bug**
3. Convert path condition to regular expression

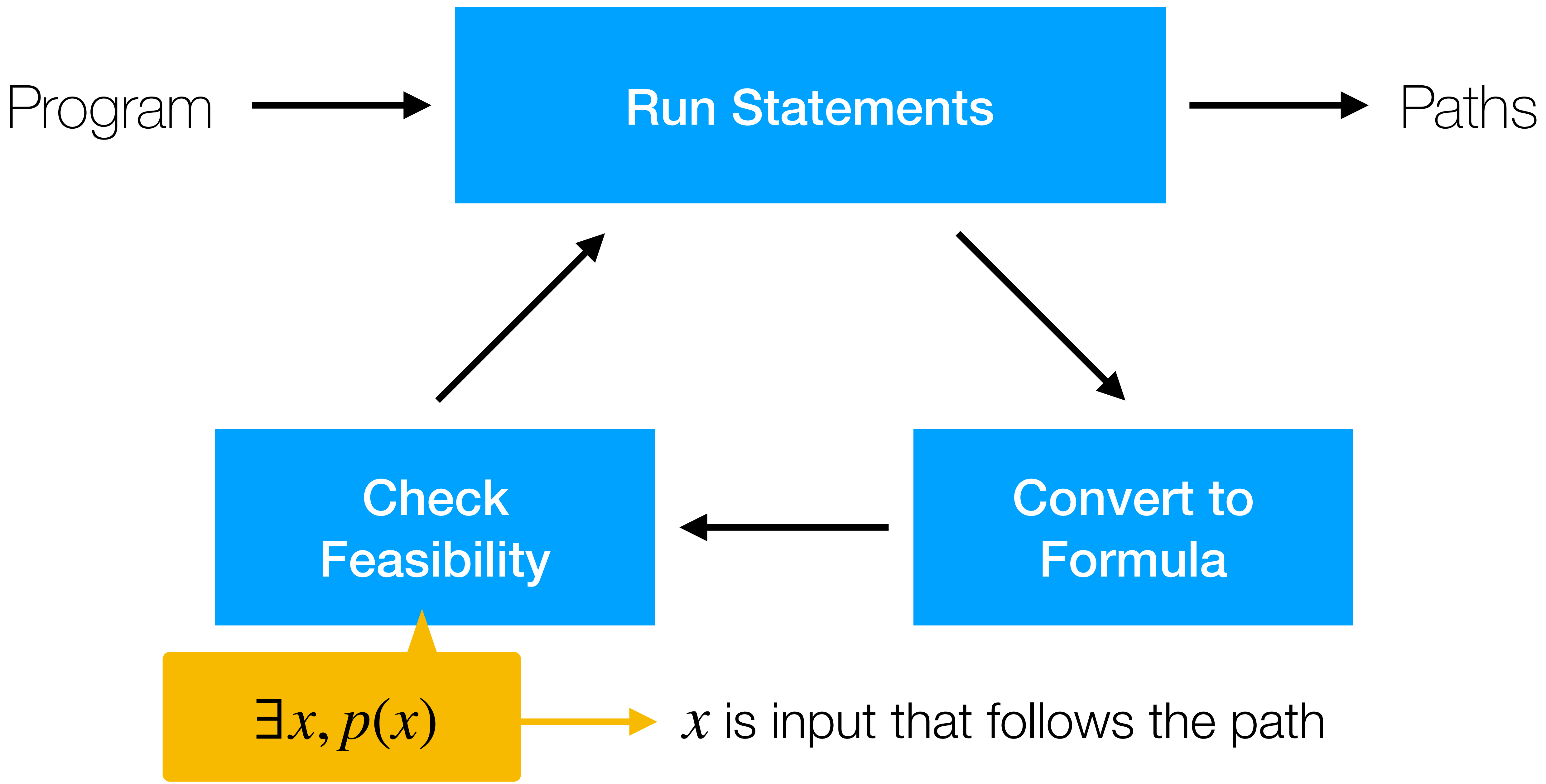
Recitation **Friday 13:00 MEB 3485** (as usual)

Installation, how to use KLEE, example output

Concrete/Symbolic

Generating inputs that test program paths

Architecture



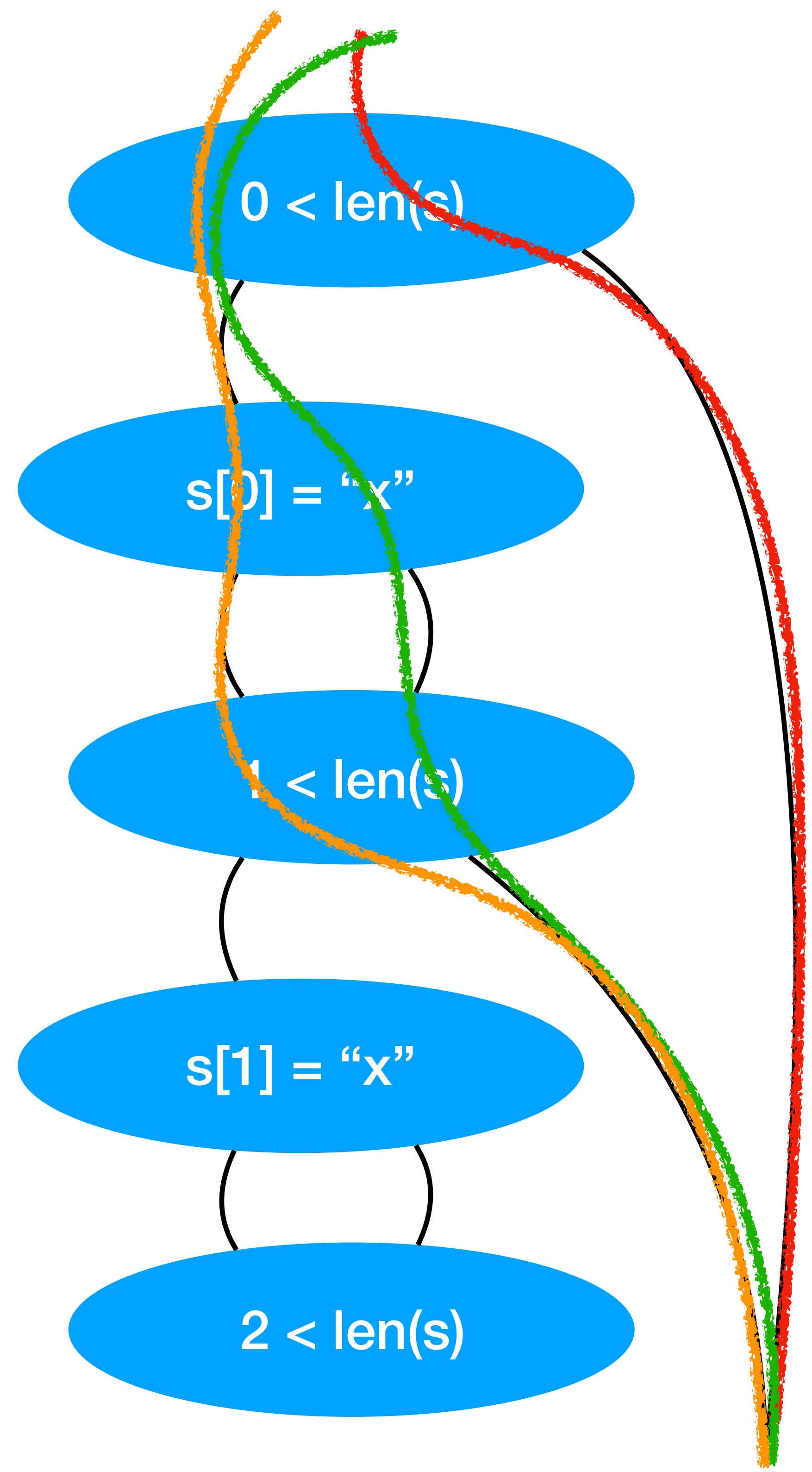
Example

Each path condition is **one input**

$\neg(0 < \textbf{len}(s))$ → **“”**

$0 < \textbf{len}(s) \wedge s[0] = x \wedge \neg(1 < \textbf{len}(s))$ → **“x”**

$0 < \textbf{len}(s) \wedge s[0] \neq x \wedge \neg(1 < \textbf{len}(s))$ → **“y”**



Each input is **distinct**

- Path conditions **mutually exclusive**
- Achieve **complete path coverage**

Concrete/symbolic

Totally symbolic inputs challenging in many cases

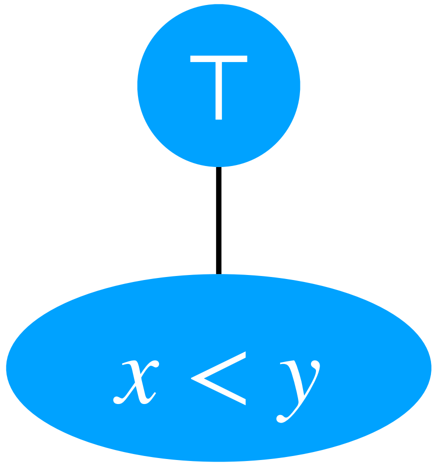
- Loops: **infinite path trees**, must be cut off
- Data structures: **infinite possibilities**, must be bounded
- System calls: **how do you print a symbolic string**?

Solution: **solve for inputs** while symbolically executing

- Input is **representative** for current path condition

Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, \top$



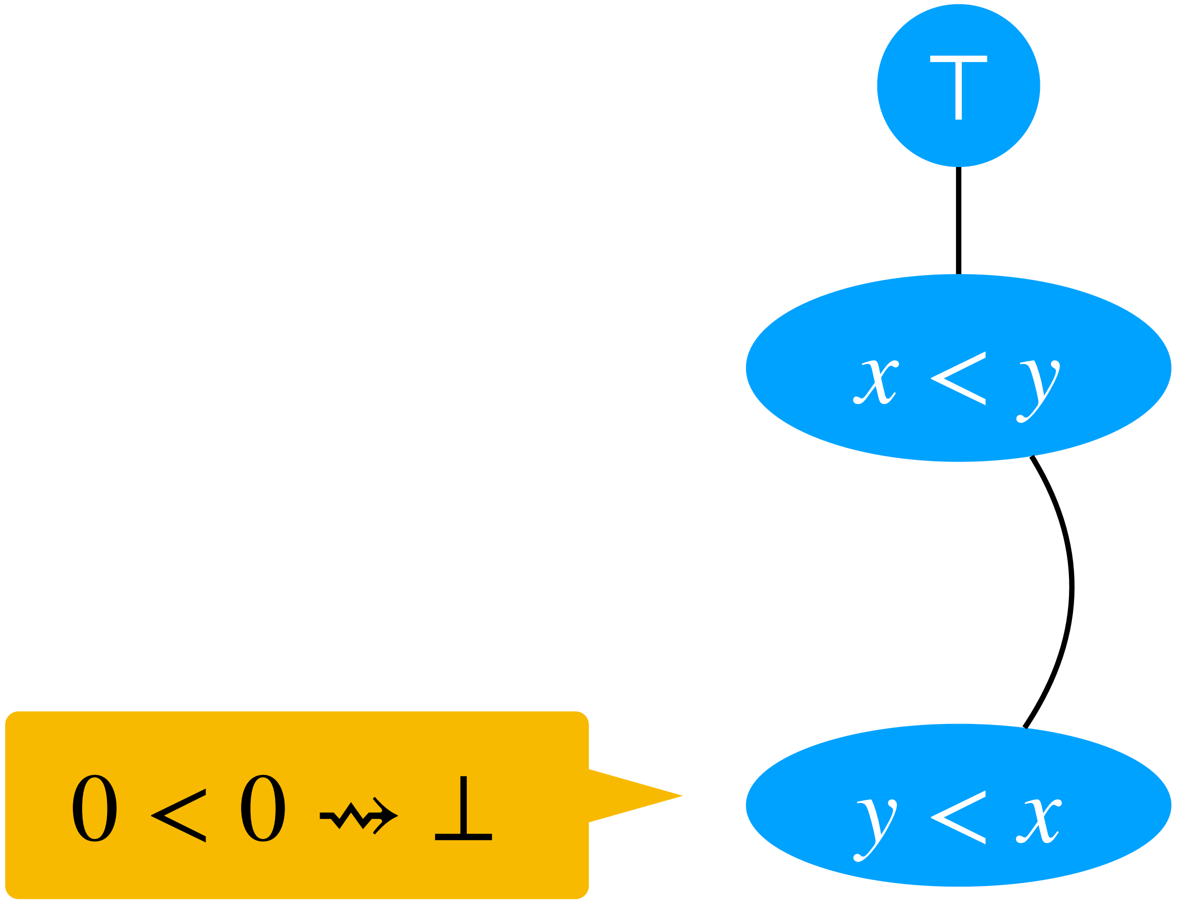
$0 < 0 \rightsquigarrow \bot$

Solution: $x = 0,\ y = 0$

Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, \top$

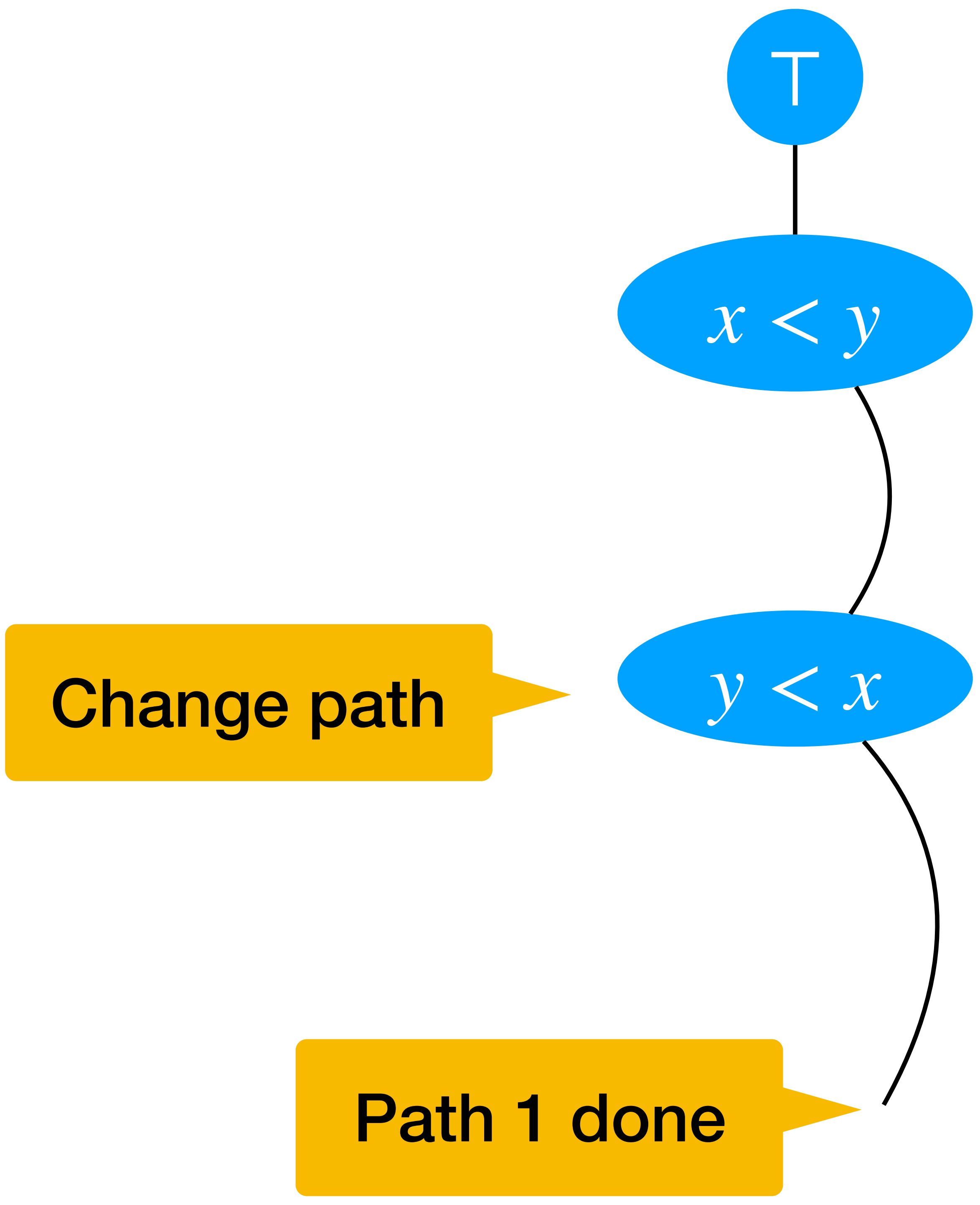
Solution: $x = 0,\ y = 0$



Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, \top$

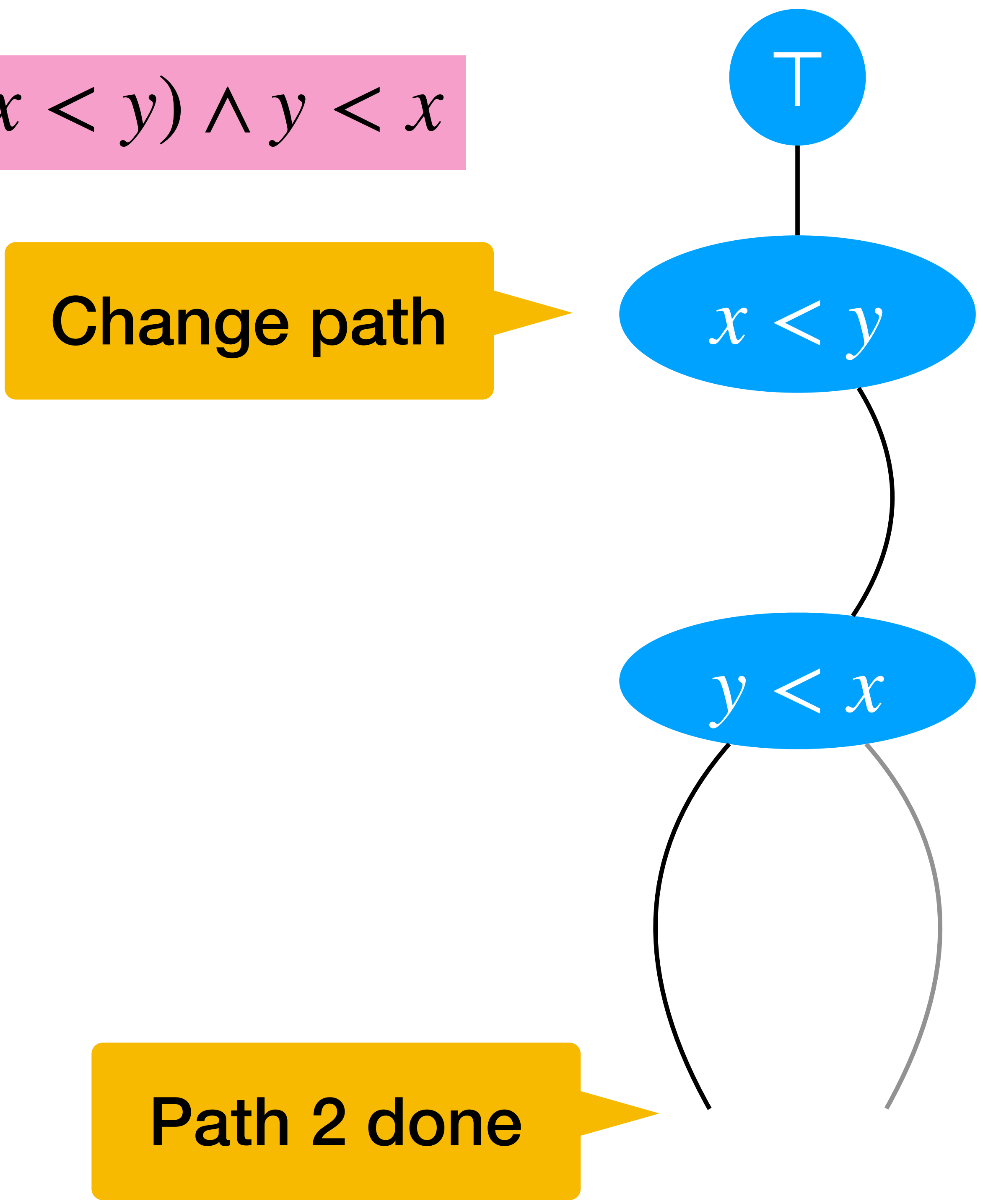
Solution: $x = 0, y = 0$



Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, \neg(x < y) \wedge y < x$

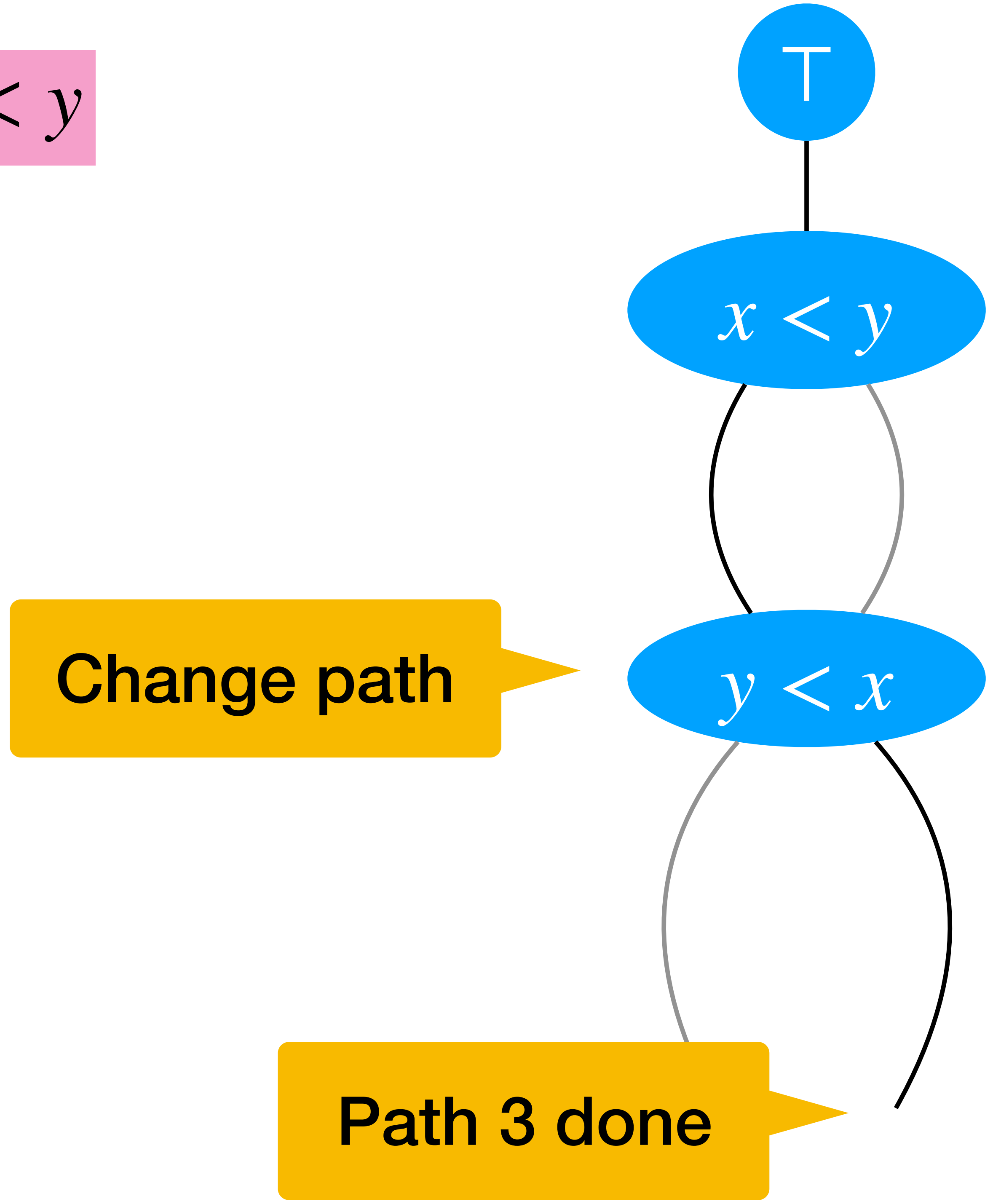
Solution: $x = 1, y = 0$



Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, x < y$

Solution: $x = 0, y = 1$

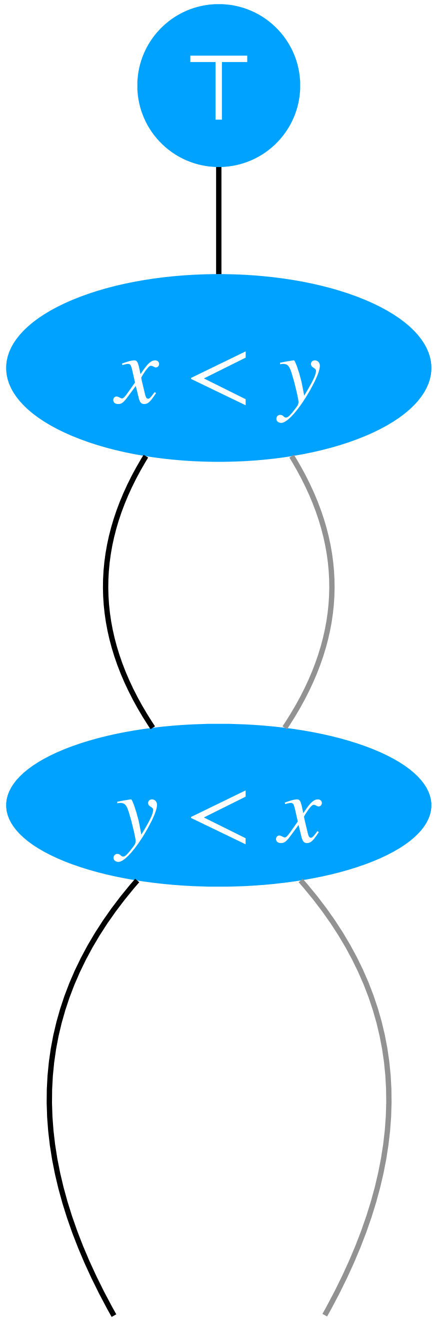


Generating Inputs

Query: $\exists x : \mathbb{Z}, \exists y : \mathbb{Z}, x < y \wedge y < x$

Solution: UNSAT

Path 4 impossible



Next class:

Hoare Logic

To do:

- ☐ Course feedback
- ☐ Proposal presentation
- ☐ Assignment 3 released

Expressions

What does an expression **compute**?

Symbolic values and environments

Executing statements without running expressions

Symbolic evaluation of expressions

Feasibility of program executions

Path conditions, concolic execution, and test generation

PRE/POST

ONE STEP

AT A TIME

ANNOTATIONS SPECIFICATION FOR PROGRAM

PROPAGATION

WEAKEST

PRECONDITION

Next class:

Hoare Logic

To do:

- ☐ Course feedback
- ☐ Proposal presentation
- ☐ Assignment 3 released