Hoare Logic

Programs section, **Lecture 14**

Pavel Panchekha

CS 6110, U of Utah

20 February 2020

Symbolic Evaluation

Program expression → Logical formula

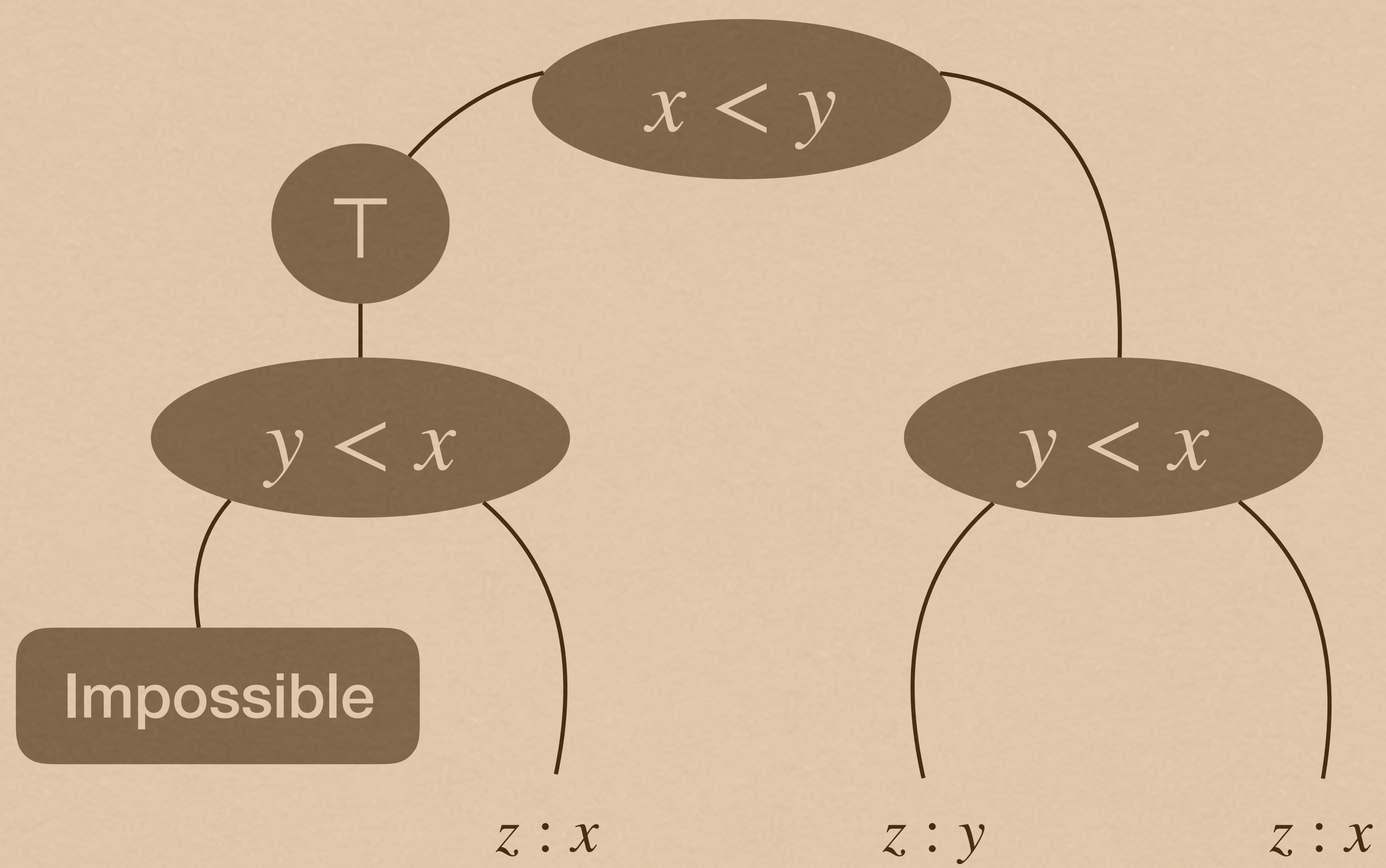
Convert program expressions to equivalent formulas

- Program variables become logical variables

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

Path Conditions

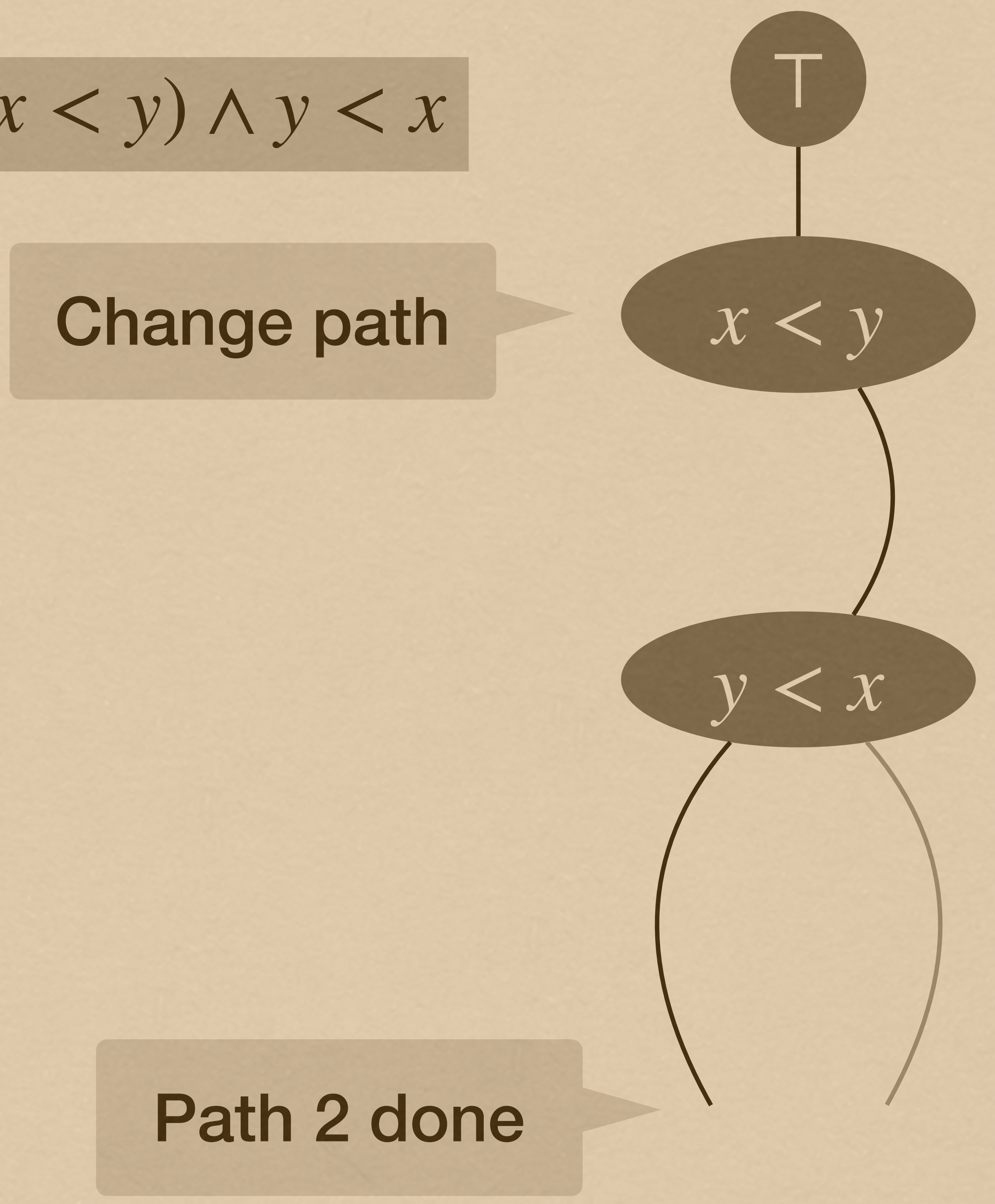
```
if (x < y):
  z = x
if (y < x):
  z = y
```



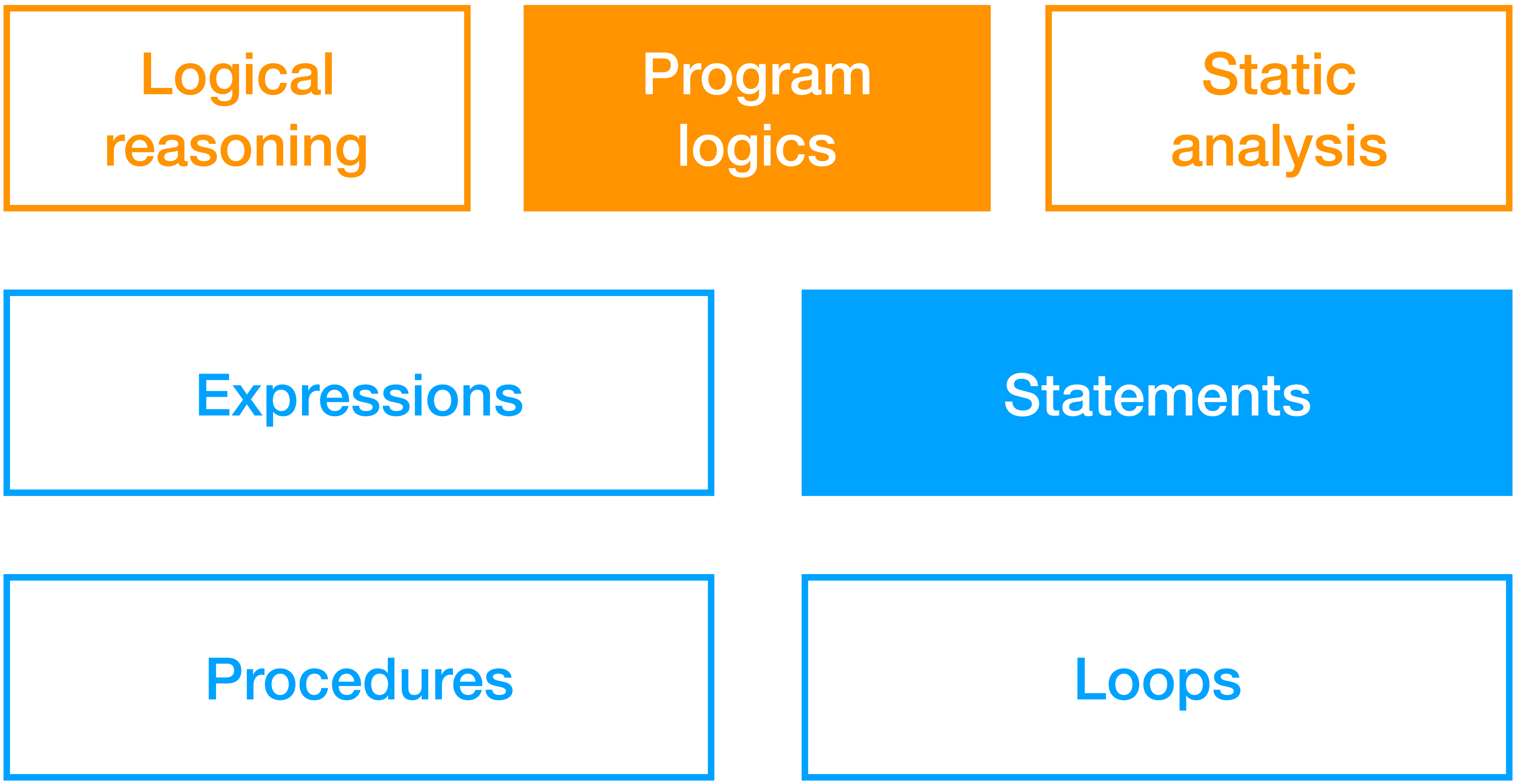
Generating Inputs

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

Solution: $x = 1,\ y = 0$



Class Progress



Statements

How to statements **modify the environment**?

- Pre- and post-conditions for statements

Generating **weakest preconditions**

- Backwards reasoning about program behavior

Verification conditions for programs

- Logical tests of program correctness

Hoare Logic

Pre- and post-conditions

Problems

What is **missing from symbolic evaluation**?

Effects	Printing text, file writes, network calls
Unbounded	Loops, data structures, recursion
Abstraction	Less detailed formulas, what not how

Move **beyond symbolic evaluation** to address these

Solutions

What is **missing from symbolic evaluation**?

Effects	Describe states, not computations
Unbounded	Describe before/after, not in-between
Abstraction	Describe properties, not values

Move **beyond symbolic evaluation** to address these

Statement

$$\{P\}\ \mathbf{S}\ \{Q\}$$

Pre-condition

Post-condition

Hoare Logic

Behavior of a statement given by a **triple**:

$$\{P\}\ \mathbf{s}\ \{Q\}$$

If **P** true of a state, and **s** executed, then **Q** true after

P and **Q** are **logical properties** of the state and effects

Need not (but can) exactly represent the state

Example

$\{ \; a \leq 5 \wedge 1 \leq b \; \}$

```
if (b < a) {
  min = b;
} else {
  min = a;
}
```

$\{ \; min \leq 5 \; \}$

Computes the smaller of `a` and `b`

Which must be less than `a`

Which is less than `5`

Exercise

Which of these triples is **true**?

$\{\ \top\ \}$ `if (b < 0) { b = -b }` $\{\ b \geq 0\ \}$

$\{\ \bot\ \}$ `b = 4` $\{\ b = -3\ \}$

$\{\ b > c\ \}$ `b *= b; c *= c` $\{\ b > c\ \}$

Statements

Let’s list some **common statements** across languages

```
pass            x = e          s ; t

if (e) {s} else {t}

while (e) {s}

f(e, e, ...)
```

Next two lectures

How does each statement’s pre-/post-conditions work?

Simple Statements

In what cases is **{P} pass {Q}** true?

$$\texttt{\{P\} pass \{Q\}} \leftrightarrow (P \rightarrow Q)$$

In what cases is **{P} s ; t {Q}** true?

$$\texttt{\{P\} s; t \{Q\}} \leftrightarrow \exists T, \begin{array}{l} \texttt{\{P\} s \{T\}} \\ \texttt{\{T\} t \{Q\}} \end{array}$$

Conditionals

In what cases is **{P} if (e) { s } else { t } {Q}** true?

- If **e**, same as **{P} s {Q}**
- If not **e**, same as **{P} t {Q}**

Triples already include the “if precondition” idea:

$$\{P \land e\}\ s\ \{Q\} \lor \{P \land \neg e\}\ t\ \{Q\}$$

Assignment

In what cases is **{P} x = e {Q}** true?

P may or may not mention **x** and constrain prior value

Anything true of **x** after must be true of **e** before!

$$\texttt{\{P\}}\ \ \texttt{x = e}\ \ \texttt{\{Q\}} \quad \leftrightarrow (P \rightarrow Q[x := e])$$

Course Updates

The mailbag and project management

The Mailbag

"How would you implement or use the concepts taught in class?"

"What programs are and are not easy to analyze?"

"I would like to get more applied experience."

Assignments

Applied lectures

Come do research!

Project Tips

Class project is a **large, long-term** project

Proposals under-estimate difficulty of verification steps

- Schedule time for **consistent progress**
- Work on **least clear parts first**
- Get a **working prototype** early

Goal is to **find failure** to give yourself time to think.

Weakest Preconditions

Reasoning about programs in reverse

Weakest Precondition

$$\texttt{\{P\} x = e \{Q\}} \quad \leftrightarrow \boxed{(P \to Q[x := e])}$$

$$WP[\texttt{x = e}](Q) = Q[x := e]$$

Common pattern: $P \to$ something(Q)

Weakest precondition of Q

$$\texttt{\{P\} s \{Q\}} \quad \leftrightarrow (P \to \overline{WP[s](Q)})$$

Weakest Precondition

$$WP[\texttt{pass}](Q) = Q$$

$$WP[\texttt{x = e}](Q) = Q[x := e]$$

$$WP[\texttt{s; t}](Q) = WP[\texttt{s}](WP[\texttt{t}](Q))$$

$$WP[\texttt{if (e) \{ s \} else \{ t \}}](Q) = (e \rightarrow WP[\texttt{s}](Q)) \wedge (\neg e \rightarrow WP[\texttt{t}](Q))$$

Example

Code

```
if (x < 0) {
  y = -x;
} else {
  y = x;
}
```

$$WP[\textbf{Code}](y \geq 0) =$$

$$Q[y := -x] = -x \geq 0$$

$$(x < 0 \rightarrow WP[\textbf{y = -x}](y \geq 0))$$

$$\wedge$$

$$(\neg(x < 0) \rightarrow WP[\textbf{y = x}](y \geq 0))$$

$$Q[y := x] = x \geq 0$$

Example

Code

```
if (x < 0) {
  y = -x;
} else {
  y = x;
}
```

$$WP[\textbf{Code}](y \geq 0) = \top$$

$$\top$$

$$(x < 0 \rightarrow -x \geq 0)$$

$$\wedge$$

$$(x \geq 0 \rightarrow x \geq 0)$$

$$\top$$

Exercise

Code

```
t = x
x = y
y = t
```

$$WP[\textbf{Code}](y = 4) =$$

Exercise

Code

```
if (k < l) {
  k++;
  n *= 2;
}
```

$$WP[\textbf{Code}](n = 2^k) =$$

Exercise

Code

```
x = x ^ y
y = x ^ y
x = x ^ y
```

$$WP[\textbf{Code}](x \geq y) =$$

Simple Questions

Why compute weakest preconditions?

Compact description of effects of a statement

Why are they called **“weakest”** preconditions?

Because $P \rightarrow WP[s](Q)$; $WP[s](Q)$ is **weaker** than P

Why not “strongest postconditions”?

Because assignments **destroy** information (old value)

Setup for Verification

Program is a **sequence of statements**

Give **pre-/post-conditions as specification**

What about functions? In a week…

$$\{P\}\ \mathbf{s}\ \{Q\}$$

Compute **verification condition** $P^* = WP[s](Q)$

Send $P \land \neg P^*$ to the solver; UNSAT means verified

Next class:

Loops

To do:

- ☐ Course feedback
- ☐ Read Chapter 5
- ☐ Assignment 3

Statements

How to statements **modify the environment**?

Pre- and post-conditions for statements

Generating **weakest preconditions**

Backwards reasoning about program behavior

Verification conditions for programs

Logical tests of program correctness

SEMANTICS

INFINITE

CODE

INVARIANTS

IMPLY

YOURSELF

TERMINATION

HALTING

RANKING

Next class:

Loops

To do:

- ☐ Course feedback
- ☐ Read Chapter 5
- ☐ Assignment 3