Symbolic Evaluation

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 \( y < x \):
    
    \( z = y \)

Impossible
Generating Inputs

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

Solution: $x = 1, y = 0$
Class Progress

- Logical reasoning
- Program logics
- Static analysis

Expressions

Statements

Procedures

Loops
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\} \mathcal{S} \{Q\}

Pre-condition

Post-condition
Hoare Logic

Behavior of a statement given by a *triple*:

\[ \{ P \} \; s \; \{ Q \} \]

- 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 \land 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**?

\[
\{ \top \} \quad \text{if } (b < 0) \quad \{ b = -b \} \quad \{ b \geq 0 \}
\]

\[
\{ \bot \} \quad \{ b = 4 \} \quad \{ b = -3 \}
\]

\[
\{ b > c \} \quad \{ b \neq b; \ c \neq c \} \quad \{ b > c \}
\]
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?

\[ \{P\} \text{ pass } \{Q\} \iff (P \rightarrow Q) \]

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

\[ \{P\} \text{ s ; t } \{Q\} \iff \exists T, \{P\} s \{T\} \{T\} t \{Q\} \]
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!

\[
\{P\} \ x = e \ \{Q\} \iff (P \rightarrow Q[x := e])
\]
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

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

\{P\} \ x = e \ \{Q\} \iff (P \to Q[x := e])

\[
WP[x = e](Q) = Q[x := e]
\]

Common pattern: \(P \to \text{something}(Q)\)

Weakest precondition of \(Q\)

\{P\} \ s \ \{Q\} \iff (P \to WP[s](Q))
Weakest Precondition

\[ WP[\text{pass}](Q) = Q \]

\[ WP[x = e](Q) = Q[x := e] \]

\[ WP[s; t](Q) = WP[s](WP[t](Q)) \]

\[ WP[\text{if (e) } \{ s \} \text{ else } \{ t \}](Q) = (e \to WP[s](Q)) \land (\neg e \to WP[t](Q)) \]
Example

\[
WP[\text{Code}](y \geq 0) = \\
Q[y := -x] = -x \geq 0
\]
\[
(x < 0 \rightarrow WP[y = -x](y \geq 0))
\]
\[
\land
\]
\[
(\neg(x < 0) \rightarrow WP[y = x](y \geq 0))
\]
\[
Q[y := x] = x \geq 0
\]
Example

Code

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

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

\[
\begin{align*}
\top & \\downarrow \\
(x < 0 \rightarrow -x \geq 0) & \\land \\
(x \geq 0 \rightarrow x \geq 0) & \\downarrow \\
\top
\end{align*}
\]
Exercise

<table>
<thead>
<tr>
<th>Code</th>
<th>WP[Code](y = 4) =</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t = x )</td>
<td></td>
</tr>
<tr>
<td>( x = y )</td>
<td></td>
</tr>
<tr>
<td>( y = t )</td>
<td></td>
</tr>
</tbody>
</table>
Exercise

Code

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

\[ WP[\text{Code}] (n = 2^k) = \]
Exercise

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

Code

\[
\begin{align*}
x &= x \wedge y \\
y &= x \wedge y \\
x &= x \wedge y
\end{align*}
\]
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

\{P\} \ s \ \{Q\}

{P} \ s \ \{Q\}

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

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

What about functions? In a week...
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
INVARIANTS

IMPLY

YOURSELF
Next class:

Loops

To do:
- Course feedback
- Read Chapter 5
- Assignment 3