Lecture 9
Proving program correctness

Armando Solar-Lezama and Xiaokang Qiu
Consider the following program

```java
... 
if(x > y){
    t = x - y;
    while(t > 0){
        x = x - 1;
        y = y + 1;
        t = t - 1;
    }
}
```

I claim that for any values of x and y

- the loop will terminate
- when it does, if x > y, the values of x and y will be swapped

How could I prove this?
Axiomatic Semantics

A system for proving properties about programs

Key idea:
- we can define the semantics of a construct by describing its effect on assertions about the program state

Two components
- A language for stating assertions
  - can be First Order Logic (FOL) or a specialized logic such as separation logic.
  - many specialized languages developed over the years
    - Z, Larch, JML, Spec#
- Deductive rules for establishing the truth of such assertions
The basics

Hoare triple
- If the precondition holds before stmt and stmt terminates, postcondition will hold afterwards

This is a partial correctness assertion
- we sometimes use the notation
  \[[A] \text{ stmt } [B]\]

  to denote a total correctness assertion
  - that means you also have to prove termination
What do assertions mean?

We first need to introduce a language

For today we will be using Winskel’s IMP

\[ e := n \mid x \mid e_1 + e_2 \]

\[ c := x := e \mid c_1 ; c_2 \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c \]

Big Step Semantics have two kinds of judgments

\[ \langle e, \sigma \rangle \rightarrow n \]

expressions result in values

\[ \langle c, \sigma \rangle \rightarrow \sigma' \]

commands change the state
What do assertions mean?

The language of assertions
- \( A := \text{true} | \text{false} | e_1 = e_2 | e_1 \geq e_2 | A_1 \text{ and } A_2 | \text{not } A | \forall x . A \)

Notation \( \sigma \models A \) means that the assertion holds on state \( \sigma \)
- This is defined inductively over the structure of \( A \).
- Ex.
  \[ \sigma \models A \text{ and } B \iff \sigma \models A \text{ and } \sigma \models B \]

Partial Correctness can then be defined in terms of OS
\( \{A\} \rightarrow \{B\} \iff \forall \sigma \forall \sigma' (\sigma \models A \land \langle c, \sigma \rangle \rightarrow \sigma') \Rightarrow \sigma' \models B \)
Derivation Rules

Derivation rules for each language construct

\[ \vdash \{ A[x \rightarrow e] \} x := e \{ A \} \quad \vdash \{ A \land b \} c_1 \{ B \} \quad \vdash \{ A \land \neg b \} c_2 \{ B \} \]

\[ \vdash \{ A \} \text{if } b \text{ then } c_1 \text{ else } c_2 \{ B \} \quad \vdash \{ A \} \text{while } b \text{ do } c \{ A \land \neg b \} \quad \vdash \{ A \} c_1 \{ C \} \quad \vdash \{ C \} c_2 \{ B \} \]

\[ \vdash \{ A \} ; c_2 \{ B \} \]

Can be combined together with the rule of consequence

\[ \vdash A' \Rightarrow A \vdash \{ A \} c \{ B \} \vdash B \Rightarrow B' \]

\[ \vdash \{ A' \} c \{ B' \} \]
Running Example

\{x = x_{old} \quad y = y_{old} \quad x > y\}
if(x > y) {
    t = x - y;
    while(t > 0) {
        x = x - 1;
        y = y + 1;
        t = t - 1;
    }
}
Soundness and Completeness

What does it mean for our deduction rules to be sound?

• You will never be able to prove anything that is not true
• truth is defined in terms of our original definition of \( \{A\} \subseteq \{B\} \)

\[
\forall \sigma \forall \sigma' (\sigma \models A \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' \models B
\]

• we can prove this, but it’s tricky

What does it mean for them to be complete?

• If a statement is true, we should be able to prove it via deduction

So are they complete?

• yes and no
  • They are complete relative to the logic
  • but there are no complete and consistent logics for elementary arithmetic (Gödel)
Weakest Preconditions

\[ P = wpc(c, A) \]

Weakest predicate \( P \) such that \( \models \{ P \} \ c \ \{ A \} \)

- \( P \) weaker than \( Q \) iff \( Q \Rightarrow P \)

\[
\begin{align*}
wpc(\text{skip} \ \{ Q \}) &= Q \\
wpc(x = e\{Q\}) &= Q[e/x] \\
wpc(C1; C2\{Q\}) &= wpc(C1\{wpc(C2\{Q\})\}) \\
wpc(\text{if } B \text{ then } C1 \text{ else } C2\{Q\}) &= (B \text{ and } wpc(C1\{Q\})) \text{ or (not } B \text{ and } wpc(C2\{Q\}))
\end{align*}
\]
Weakest Precondition

While-loop is tricky

• Let \( W = wpc(\text{while } e \text{ do } c, B) \)
• then,

\[
W = e \Rightarrow wpc(c, W) \land \neg e \Rightarrow B
\]
Verification Condition

Stronger than the weakest precondition

Can be computed by using an invariant

\[ VC(\text{while}_e \text{ do } c, B) = I \land \forall x_1, \ldots, x_n I \Rightarrow (e \Rightarrow VC(c, I) \land \neg e \Rightarrow B) \]

- Where \( x_i \) are variables modified in \( c \).