Scribe: Yejin Choi, Bryan Silverthorn Lecturer: Andrew Myers

1 Introduction

In this lecture, we will learn techniques for reasoning about the behavior of programs when applied to sets of states. In particular, we would like to reason about the correctness of programs. Operational and denotational semantics are in some sense too fine-grained, while practical type systems make only the very coarse guarantee that a program will never become stuck. Today we will introduce the axiomatic semantics, which consider programs as predicate transformers: given statements about the set of possible states, we can determine how executing the program affects those statements.

2 Assertions

An *assertion* is a predicate that characterizes the set of possible states before and after the execution of a command.

A *partial correctness assertion* is an assertion of the form

$$\{A\}\ c\ \{B\}$$

This assertion specifies that, if we start in a state satisfying A and run c, *and c terminates*, we will be in a state satisfying B. Note that the condition on the termination of c is what makes this assertion partial. Here A is called a *pre-condition* and B is called a *post-condition*.

A *total correctness assertion* is given as

$$[A]\ c\ [B] \iff \{A\}\ c\ \{B\} \wedge \forall \sigma \models A\ c \Downarrow$$

What is an assertion? It is simply a statement in some *assertion language* that we specify. For IMP, we can define the syntax for conditions as

$$\mathsf{Assn} \ni A \ ::= \mathsf{true} \mid \mathsf{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid \neg A \mid \forall i.A \mid \exists i.A$$

Recall the syntax for artithmatic expressions in IMP, defined as

$$\mathsf{Aexp} \ni a \ ::= n \mid x \mid i \mid a_1 + a_2 \mid a_1 \times a_2 \mid$$

with the addition of $i \in \mathsf{IVar}$, a set of fixed variables associated with assertion quantifiers. This assertion language is quite powerful; if we had arrays in IMP, we could write assertions such as

$$\forall i. \forall j. i \leq j \Rightarrow a[i] \leq a[j]$$

which asserts that "the array a is sorted."

Now we can give the semantics of this assertion language, where the "meaning" of an assertion is the set of satisfying states:

$$\sigma \in [\![A]\!]_I \Leftrightarrow \sigma \models_I A$$

under some environment for fixed vars:

$$I : \mathsf{IVar} \to \mathbb{Z}$$

allowing us to define the semantics below:

$$\begin{aligned}
[\![\mathsf{true}]\!]_I &= \Sigma_\perp \\
[\![\mathsf{false}]\!]_I &= \{\perp\} \\
[\![A \wedge B]\!]_I &= [\![A]\!]_I \cap [\![B]\!]_I \\
[\![A \vee B]\!]_I &= [\![A]\!]_I \cup [\![B]\!]_I \\
[\![\neg A]\!] &= (\Sigma_\perp \setminus [\![A]\!]) \cup \{\perp\} \\
[\![\forall i.A]\!]_I &= \bigcap_{n\in\mathbb{Z}} [\![A]\!]_{I[i\mapsto n]} \\
[\![\exists i.A]\!]_I &= \bigcup_{n\in\mathbb{Z}} [\![A]\!]_{I[i\mapsto n]} \\
[\![a_1 = a_2]\!]_I &= \{\sigma | \mathcal{A}[\![\hat{I}(a_1)]\!]\sigma = \mathcal{A}[\![\hat{I}(a_2)]\!]\sigma\}
\end{aligned}$$

where $\hat{I}(a)$ replaces all the occurances of free variables in a with numbers that I maps to. Notice that in order to compare the values of arithematic expressions, we borrowed the reasoning from the denotational logic.

3 Hoare Logic

How do we show that some assertion is true? We introduce Hoare logic, a proof system for Partial Correctness Assertions (PCAs). We define the proof system by induction on the structure of the commands. Notice that statements look similiar to those of the big step semantics, except that we use assertions in the place of states.

$$\frac{}{\{A\}\ \mathsf{skip}\ \{A\}}$$

$$\frac{}{\{B\{a/x\}\}\ x := a\ \{B\}}$$

$$\frac{\{A\}\ c_1\ \{C\} \quad \{C\}\ c_2\ \{B\}}{\{A\}\ c_1; c_2\ \{B\}}$$

$$\frac{\{A \wedge b\}\ c_1\ \{B\} \quad \{A \wedge \neg b\}\ c_2\ \{B\}}{\{A\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \{B\}}$$

$$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{A \wedge \neg b\}}$$

The rules above are straightforward and purely syntax-directed. We need a final rule, however, which eliminates this simplicity: the "rule of consequence" below

$$\frac{\models A \Rightarrow A' \quad \{A'\}\ c\ \{B'\} \quad \models B' \Rightarrow B}{\{A\}\ c\ \{B\}}$$

How do we know that this set of inference rules is reasonable? That is, is it sound? Is it complete? Soundness and completeness for Hoare logic with proof rules above can be captured as follows;

- Soundness:

$$\vdash \{A\}\ c\ \{B\} \Rightarrow \underbrace{\forall I.\sigma \models_I A \Rightarrow \mathcal{C}[\![c]\!]\sigma \models_I B}_{\models\{A\}\ c\ \{B\}}$$

- Completeness:

$$\models \{A\}\ c\ \{B\} \Rightarrow \vdash \{A\}\ c\ \{B\}$$

A complete Hoare logic is not achievable (blame Gödel), but we can get *relative completeness*. That is, the completeness of Hoare logic is relative to our ability to reason about implications in the rule of consequence—if we had an oracle to take care of the implications, which we know is not possible to obtain, then we would have completeness.

3.1 Soundness

Let's start by sketching out a proof of soundness, by induction on $\vdash \{A\}\ c\ \{B\}$

- case $\{A\}$ skip $\{A\}$

$$\sigma \models_I A \stackrel{?}{\Rightarrow} \underbrace{\mathcal{C}[\![\mathsf{skip}]\!]\sigma}_{\sigma} \models_I A$$

  This trivially holds, since the denotation of $\mathcal{C}[\![\mathsf{skip}]\!]\sigma$ is just σ.

- case $\{B\{a/x\}\}\ x := a\ \{B\}$

$$\sigma \models_I B\{a/x\} \stackrel{?}{\Rightarrow} \underbrace{\mathcal{C}[\![x := a]\!]\sigma}_{\sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]} \models_I B$$

  We have $\mathcal{C}[\![x := a]\!]\sigma$ equals to $\sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$ from the denotational semantics of an assignment command. The fact that effect of substitution in $B\{a/x\}$ abides by the updated state can be shown by a substitution lemma, proved by induction on B and the structure of arithmetic expressions. Hence this holds.

- skipping to case while

$$\sigma \models_I A \stackrel{?}{\Rightarrow} \mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]\sigma \models_I \{A \wedge \neg b\}$$

  Recall that the denotation of while command was

$$\begin{aligned}
\mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!] &= fix(\underbrace{\lambda f.\lambda\sigma.\mathsf{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathsf{then}\ f^*(\mathcal{C}[\![c]\!]\sigma)\ \mathsf{else}\ \sigma}_{F}) \\
&= fix\ F \\
&= \bigsqcup_{n\in\omega} F^n(\bot) \\
[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]\sigma &= (fix\ F)\sigma \\
&= \big(\bigsqcup_{n\in\omega} F^n(\bot)\big)\sigma \\
&= \bigsqcup_{n\in\omega} \big((F^n(\bot))\sigma\big)
\end{aligned}$$

  The chain of states is either $\bot \sqsubseteq \bot \sqsubseteq \bot \sqsubseteq \bot \sqsubseteq \cdots$ or $\bot \sqsubseteq \cdots \sqsubseteq \bot \sqsubseteq \sigma' \sqsubseteq \sigma' \sqsubseteq \sigma' \sqsubseteq \cdots$ because $\Sigma = \mathsf{Var} \to \mathbb{Z}$ is a discrete CPO. We will show that in any case,

$$\forall n. F^n(\bot)\sigma \models_I \{A \wedge \neg b\}$$

  holds, by induction on n

  - base $n = 0$: $\bot \models_I A \wedge \neg b$; anything trivially holds at bottom.
  - induction : to be continued.