§ 1 Valid Partial Correctness Assertions

In the last lecture, we introduced the concept of a partial correctness assertion,

\{A\}p\{B\}

where A is a precondition or assertion, p is a program, and B is a postcondition. This PCA means that if program P is started in any state which satisfies A, then when and if p halts, the halting state satisfies B.

Formally, we say \(\models \{A\}p\{B\}\), (\(\{A\}p\{B\}\) is valid), if for all \(\sigma, \tau \in \Sigma\) and for all interpretations, \(I\),

\(\sigma \models^I A \land (\sigma, \tau) \in R[p] \rightarrow \tau \models^I B\)

In 1969, Hoare introduced the following proof system for deriving valid PCA’s:

**Assignment Axiom:**

\[
\frac{}{\{A[t/x]\} x := t \{A\}}
\]

Example:

\(\{1 + 2 = 3\} x := 1 + 2 \{x = 3\}\)

**Composition Rule:**

\[
\frac{\{A\}p\{B\}, \{B\}q\{C\}}{\{A\}p; q\{\{C\}}
\]

**Conditional Rule:**

\[
\frac{\{A \land b\}p\{C\}, \{A \land \bar{b}\}q\{C\}}{\{A\} if\ b\ then\ p\ else\ q\{C\}}
\]

**While Rule:**

\[
\frac{\{b \land A\}p\{A\}}{\{A\} while\ b\ do\ p\{A \land \bar{b}\}}
\]

**Weakening Rule:**

\[
\frac{A \rightarrow A', \{A'\}p\{B'\}, B' \rightarrow B}{\{A\}p\{B\}}
\]

**Definition:** We say \(\vdash \{A\}p\{B\}\), (\(\{A\}p\{B\}\) is derivable), if there is a proof tree for \(\{A\}p\{B\}\) using the proof system defined above and the theory of the domain of computation (in our case, the theory of the natural numbers).

**Claim:** The proof system defined above is both sound (\(\vdash \{A\}p\{B\} \Rightarrow \models \{A\}p\{B\}\) ), and complete (\(\models \{A\}p\{B\} \Rightarrow \vdash \{A\}p\{B\}\) ) for the theory of the natural numbers.

The proof of soundness is straightforward, while the proof of completeness is attributed to Cook.
2 An Application of Hoare’s Proof System in Program Verification

Consider the following program, \( p \):

\[
\text{while } (y \neq 0) \{
  z := x \mod y; \\
  x := y; \\
  y := z;
\}
\]

We wish to verify that this correctly computes the GCD of \( x \) and \( y \), assuming \( x \) and \( y \) are not both zero. That is,

\[
\vdash \{ x = i \wedge y = j \wedge \neg (i = 0 \wedge j = 0) \} \ p \ { x = \text{gcd}(i, j) }\]

**Proof:** By the weakening rule, it suffices to show:

\[
\{ \text{gcd}(x, y) = \text{gcd}(i, j) \wedge \neg (x = 0 \wedge y = 0) \} \ p \ { x = \text{gcd}(i, j) }\]

The following PCAs are easily verified with a dash of number theory:

\[
\{ \text{gcd}(x, y) = \text{gcd}(i, j) \wedge \neg (y = 0) \} \ z := x \mod y \ \{ \text{gcd}(y, z) = \text{gcd}(i, j) \}
\]

\[
\{ \text{gcd}(y, z) = \text{gcd}(i, j) \} \ x := y \ \{ \text{gcd}(x, z) = \text{gcd}(i, j) \}
\]

\[
\{ \text{gcd}(x, z) = \text{gcd}(i, j) \} \ y := z \ \{ \text{gcd}(x, y) = \text{gcd}(i, j) \}
\]

So, repeatedly applying the composition rule, we have:

\[
\{ \text{gcd}(x, y) = \text{gcd}(i, j) \wedge \neg (y = 0) \} \ z := x \mod y; \ x := y; \ y := z; \ \{ \text{gcd}(x, y) = \text{gcd}(i, j) \}
\]

Finally, the while rule yields:

\[
\{ \text{gcd}(x, y) = \text{gcd}(i, j) \wedge \neg (y = 0) \} \ p \ \{ \text{gcd}(x, y) = \text{gcd}(i, j) \wedge y = 0 \}
\]

Since \( \text{gcd}(x, 0) = x \), the weakening rule now allows us to make the desired conclusion.

3 Relational Semantics

As we saw in the previous lecture, programs can be interpreted as sets of pairs, each pair consisting of an input state and an output state. If \( \Sigma \) is the set of possible states and \( p \) is a program, then \( R[p] \subseteq \Sigma \times \Sigma \) is a binary relation which represents the meaning of \( p \) in relational semantics. We can use either of the following (equivalent) definitions for \( R[p] \):

\[
R[p] \overset{df}{=} \{(\sigma, \tau) | \tau = C[p][\sigma]\}
\]

\[
R[p] \overset{df}{=} \{(\sigma, \tau) | \langle p, \sigma \rangle \rightarrow \tau\}
\]

We also defined \( R \) on boolean values \( b \) as

\[
R[b] \overset{df}{=} \{(\sigma, \sigma) | \sigma \models b\}
\]

We can now define some basic operations on these relations.
\[ R \circ S \overset{\text{def}}{=} \{ (\sigma, \rho) \mid \exists \tau \text{ such that } (\sigma, \tau) \in R \text{ and } (\tau, \rho) \in S \} \]

\[ R \cup S \overset{\text{def}}{=} \{ (\sigma, \rho) \mid (\sigma, \rho) \in R \text{ or } (\sigma, \rho) \in S \} \]

\[ R^* \overset{\text{def}}{=} \bigcup_{n \geq 0} R^n \]

where \( R^n \) is defined inductively as

\[
R^0 = \{ (\sigma, \sigma) \mid \sigma \in \Sigma \}
\]

\[
R^{n+1} = R \circ R^n
\]

Using these operations on relations, we can define the meaning of three operations on programs: composition, non-deterministic choice, and iteration.

\[ R[p; q] \overset{\text{def}}{=} R[p] \circ R[q] \]

\[ R[p + q] \overset{\text{def}}{=} R[p] \cup R[q] \]

\[ R[p^*] \overset{\text{def}}{=} R[p]^* \]

Note that we can now give simple interpretations to our language constructs, including while. For example,

\[ \mathcal{R}[\text{if } b \text{ then } p \text{ else } q] = \mathcal{R}[((b; p) + (\overline{b}; q))] \]

\[ \mathcal{R}[\text{while } b \text{ do } p] = \mathcal{R}[(b; p)^*; \overline{b}] \]

So what we have now is a set of regular operators. Those equations which are true as regular expressions, such as \((p + q)^* = (p^*q)^*p^* \) and \( p(qp)^* = (pq)^*p \), are exactly those expressions which are true for our binary relations.