Scribe: Reba Schuller and Tom Wexler Lecturer: Dexter Kozen

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,

$$(\sigma \models^I A \ \wedge \ (\sigma, \tau) \in \mathcal{R}[\![\mathsf{p}]\!]) \Rightarrow \tau \models^{\mathcal{I}} \mathcal{B}$$

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

Assignment Axiom:

$$\overline{\{A[t/x]\}x := t\{A\}}$$

Example:

$$\{1 + 2 = 3\}\ x := 1 + 2\ \{x = 3\}$$

Composition Rule:

$$\frac{\{A\}p\{B\}, \{B\}q\{C\}}{\{A\}\mathsf{p}; \mathsf{q}\{\{C\}}$$

Conditional Rule:

$$\frac{\{A \wedge b\}p\{C\}, \{A \wedge \bar{b}\}q\{C\}}{\{A\}\mathbf{if}\, b\, \mathbf{then}\, p\, \mathbf{else}\, q\{C\}}$$

While Rule:

$$\frac{\{b \wedge A\}p\{A\}}{\{A\}\mathbf{while}\, b\, \mathbf{do}\, p\{A \wedge \bar{b}\}}$$

Weakening Rule:

$$\frac{A \to A', \{A'\}p\{B'\}, B' \to B}{\{A\}p\{B\}}$$

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:

```
while (y != 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,

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

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

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

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

$$\{gcd(x, y) = gcd(i, j) \wedge \neg(y = 0)\}\ z := x(mod\ \mathrm{y})\ \{gcd(y, z) = gcd(i, j)\}$$

$$\{gcd(y, z) = gcd(i, j)\}\ x := y\ \{gcd(x, z) = gcd(i, j)\ \}$$

$$\{gcd(x, z) = gcd(i, j)\}\ y := z\ \{gcd(x, y) = gcd(i, j)\}$$

So, repeatedly applying the composition rule, we have:

$$\{gcd(x, y) = gcd(i, j) \wedge \neg(y = 0)\}\ \{z := x(mody);\ x := y;\ y := z;\}\ \{gcd(x, y) = gcd(i, j)\ \}$$

Finally, the while rule yields:

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

Since $gcd(x, 0) = x$, 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 Σ is the set of possible states and p is a program, then $\mathcal{R}[\![p]\!] \subseteq \Sigma \times \Sigma$ is a binary relation which represents the meaning of p in relational semantics. We can use either of the following (equivalent) definitions for $\mathcal{R}[\![p]\!]$:

$$\begin{aligned} \mathcal{R}[\![p]\!] &\stackrel{def}{=} \{(\sigma, \tau) | \tau = \mathcal{C}[\![p]\!]\sigma\} \\ &\stackrel{def}{=} \{(\sigma, \tau) | \langle p,\ \sigma\rangle \rightarrow \tau\} \end{aligned}$$

We also defined $\mathcal{R}$ on boolean values b as

$$\mathcal{R}[\![b]\!] \stackrel{def}{=} \{(\sigma, \sigma) | \sigma \models b\}$$

We can now define some basic operations on these relations.

$$
\begin{aligned}
\mathcal{R} \circ \mathcal{S} &\stackrel{def}{=} \{(\sigma, \rho) | \exists \tau \text{ such that } (\sigma, \tau) \in \mathcal{R} \text{ and } (\tau, \rho) \in \mathcal{S}\} \\
\mathcal{R} \cup \mathcal{S} &\stackrel{def}{=} \{(\sigma, \rho) | (\sigma, \rho) \in \mathcal{R} \text{ or } (\sigma, \rho) \in \mathcal{S}\} \\
\mathcal{R}^* &\stackrel{def}{=} \bigcup_{n \geq 0} \mathcal{R}^n
\end{aligned}
$$

where $\mathcal{R}^n$ is defined inductively as

$$
\begin{aligned}
\mathcal{R}^0 &= \{(\sigma, \sigma) | \sigma \in \Sigma\} \\
\mathcal{R}^{n+1} &= \mathcal{R} \circ \mathcal{R}^n
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{R}[\![p; q]\!] &\stackrel{def}{=} \mathcal{R}[\![p]\!] \circ \mathcal{R}[\![q]\!] \\
\mathcal{R}[\![p + q]\!] &\stackrel{def}{=} \mathcal{R}[\![p]\!] \cup \mathcal{R}[\![q]\!] \\
\mathcal{R}[\![p^*]\!] &\stackrel{def}{=} R[\![p]\!]^*
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{R}[\![\text{ if } b \text{ then } p \text{ else } q]\!] &= \mathcal{R}[\![(b; p) + (\overline{b}; q)]\!] \\
\mathcal{R}[\![\text{ while } b \text{ do } p]\!] &= \mathcal{R}[\![(b; p)^*; \overline{b}]\!]
\end{aligned}
$$

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.