Scribe: Xinyang Zhang, Xiangyang Lan Lecturer: Radu Rugina

1 Finishing Denotational Semantics

We express the denotational semantics of the while loop $\mathcal{C}[\![w]\!]$ using the higher order function Γ which maps functions into functions, i.e., $\Gamma \in (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$. Given $\gamma : \Sigma \to \Sigma_\perp$, $\Gamma(\gamma)$ is:

$$\begin{array}{llll} \Gamma(\gamma) = \lambda\sigma. & if & \mathcal{B}[\![b]\!]\sigma & then \\ & & if & \mathcal{C}[\![c]\!]\sigma = \perp \quad then \quad \perp \\ & & else & \gamma(\mathcal{C}[\![c]\!]\sigma) \\ & else & \sigma & \end{array}$$

We can rewrite Γ using domain constructions:

$$\begin{array}{rcl} \Gamma(\gamma) & = & \lambda\sigma. \quad if \;\; \mathcal{B}[\![b]\!]\sigma \;\; then \;\; \gamma^\star(\mathcal{C}[\![c]\!]\sigma) \;\; else \;\; \sigma \\ & = & \lambda\sigma. \quad cond\ (\mathcal{B}[\![b]\!]\sigma, \gamma^\star(\mathcal{C}[\![c]\!]\sigma), \sigma) \end{array}$$

So we get:

$$\Gamma(\gamma) = cond \circ \langle \mathcal{B}[\![b]\!], \gamma^\star \circ \mathcal{C}[\![c]\!], id \rangle$$

And $\mathcal{C}[\![w]\!] = fix(\Gamma)$.

Claim: if $\mathcal{B}[\![b]\!]$, $\mathcal{C}[\![c]\!]$ are continuous, then:

1. γ *continuous* $\Rightarrow$ $\Gamma(\gamma)$ *continuous* $\quad (\Gamma : [\Sigma \to \Sigma_\perp] \to [\Sigma \to \Sigma_\perp])$
2. Γ *is continuous*

Proof: (1) easy, using composition of continuous functions in $\Sigma \to \Sigma_\perp$.
For (2), we can express Γ as follows:

$$\Gamma = \lambda\gamma.\ \ cond \circ \langle \mathcal{B}[\![b]\!], \gamma^\star \circ \mathcal{C}[\![c]\!], id \rangle$$

Since $\lambda\gamma.\mathcal{B}[\![b]\!]$ and $\lambda\gamma.id$ are constant functions, they are continuous.
Also $\lambda\gamma.\gamma^\star$ is continuous and $\lambda f.(f \circ \mathcal{C}[\![c]\!])$ is continuous(where f is continuous), so their composition is also continuous:

$$\lambda\gamma.(\gamma^\star \circ \mathcal{C}[\![c]\!]) = (\lambda f.f \circ \mathcal{C}[\![c]\!]) \circ (\lambda\gamma.\gamma^\star)$$

That means $\lambda\gamma.(\gamma^\star \circ \mathcal{C}[\![c]\!])$ is continuous too. Since

$$\Gamma = \lambda\gamma.\ cond \circ \langle \mathcal{B}[\![b]\!], \gamma^\star \circ \mathcal{C}[\![c]\!], id \rangle$$

Γ is continuous, and $\mathcal{C}[\![w]\!] = fix(\Gamma)$ is well formed.

1.1 Non-Termination

We know that the command **while true do skip** doesn't terminate. In large-step operational semantics, this results building a tree of infinite height; in small-step operational semantics, this results in an infinite sequence of transitions. Let's how is non-termination expressed in denotational semantics.

$$\begin{array}{rl} & \mathcal{C}[\![\textbf{while true do skip}]\!] \\ = & fix(\lambda\gamma.\ cond \circ \langle \mathcal{B}[\![\textbf{true}]\!], \gamma^\star \circ \mathcal{C}[\![\textbf{skip}]\!], id \rangle) \\ = & fix(\lambda\gamma.\ \gamma^\star \circ \mathcal{C}[\![\textbf{skip}]\!]) \\ = & fix(\lambda\gamma.\ \gamma) \\ = & \perp_{\Sigma \to \Sigma_\perp} \\ = & \lambda\sigma.\perp_\Sigma \end{array}$$

2 Equivalence of Semantics

$$\forall a, \sigma, n \quad \langle a,\, \sigma\rangle \to n \Leftrightarrow \mathcal{A}[\![a]\!]\sigma = n \quad (1)$$
$$\forall b, \sigma, t \quad \langle b,\, \sigma\rangle \to t \Leftrightarrow \mathcal{B}[\![b]\!]\sigma = t \quad (2)$$
$$\forall c, \sigma, \sigma' \quad \langle c,\, \sigma\rangle \to \sigma' \Leftrightarrow \mathcal{C}[\![a]\!]\sigma = \sigma' \quad (3)$$

(1) and (2) can be proved with structure induction. In order to prove (3), we need to use some induction on derivations. See Winskel for proofs. With (1)-(3) proved, we can conclude that denotational semantics is equivalent to operational semantics.

3 Axiomatic Semantics

Operational semantics describe how programs execute, and Denotational semantics express the program as mathematical function. Now we present Axiomatic semantics, which express the effects of the computation in the program using precondition and postcondition assertions.

3.1 Example

$$\begin{array}{l} P := \{x = i\}; \\ \textbf{while}\ \neg(x = 0)\ \textbf{do} \\ \quad (y := y + 2 * x; x := x - 1); \\ Q := \{y = i * (i - 1)\} \end{array}$$

3.2 Assertions

There are two kinds of assertions:

- Partial Correctness Assertions:
$$\{P\}c\{Q\}$$
this kind of assertions tells us that if P holds before the program c executes and the program c terminates, then Q holds after the program terminates. But the assertion does not tell anything when the program does not terminate.

- Total Correctness Assertions:
$$[P]C[Q]$$
This kind of assertions tells us that if P holds before the program c executes, then the program c must terminate, and Q holds after the program terminates. In the previous example, we can change the partial correctness assertion to the total correctness assertion as following:
$$[x = i \wedge i \geq 0]C[y = i * (i - 1)]$$

3.3 Syntax of Assertion

The syntax of assertions is an extension of the syntax for boolean expressions. We inductively define the syntactic set of assertion **Assn**, using the following grammar:

$$\begin{array}{rcl} A & ::= & \textbf{true} \mid \textbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid \\ & & A_0 \Rightarrow A_1 \mid \forall i.A \mid \exists i.A \end{array}$$

A simpler syntax of assertions is:

$$A \quad ::= \quad \textbf{false} \mid a_0 \leq a_1 \mid A_0 \Rightarrow A_1 \mid \forall i.A$$

Since we can construct everything else using only these operators and constants:

$$\begin{array}{rcl}
\textbf{true} & \equiv & \textbf{false} \rightarrow \textbf{false} \\
\neg A & \equiv & A \Rightarrow \textbf{false} \\
a_0 = a_1 & \equiv & a_0 \leq a_1 \wedge a_1 \leq a_0 \\
A_0 \vee A_1 & \equiv & \neg A_0 \Rightarrow A_1 \\
A_0 \wedge A_1 & \equiv & \neg(\neg A_0 \vee \neg A_1) \\
\exists i, A & \equiv & \neg \forall i, \neg A
\end{array}$$

Since quantification introduces new integer variables i, which are not variables in the program, we define a new syntactic set **Intvar**:

$$\textbf{Intvar} \quad ::= \quad \text{integer variables in assertions } (i,\ j, \text{ etc})$$

We must also extend the syntactic domain of arithmetic expressions *in assertions* to include these integer variables. This new domain domain is **Aexpv**. Given an assertion A, the arithmetic expressions in A are elements of **Aexpv**. The expressions $a \in \textbf{Aexpv}$ are inductively defined as follows:

$$a \quad ::= \quad n \ \mid \ X \ \mid \ i \ \mid \ a_0 \ op \ a_1 \qquad (op \in \{+, -, \times\})$$

3.4 Semantics of assertions

The semantics of the assertion syntax defined above are as follows:

First we need to define interpretation function

$$I : \textbf{Intvar} \rightarrow \mathbb{Z}$$

Then, the evaluation function of an assertion expression is $\mathcal{A}v[\![a]\!] : (interpretation) \rightarrow \Sigma \rightarrow \mathbb{Z}$, defined as follows:

$$\begin{array}{rcl}
\mathcal{A}v[\![n]\!]I\sigma & = & n \\
\mathcal{A}v[\![x]\!]I\sigma & = & \sigma(x) \\
\mathcal{A}v[\![i]\!]I\sigma & = & I(i) \\
\mathcal{A}v[\![a_0 \ op \ a_1]\!]I\sigma & = & \mathcal{A}v[\![a_0]\!]I\sigma \ \ op \ \ \mathcal{A}v[\![a_1]\!]I\sigma \qquad (op \in \{+, -, \times\})
\end{array}$$

Now we introduce the relation $\sigma \models^I A$, which means that state σ satisfies A under interpretation I. This relation is inductively defined; we only give here a couple of rules – see Winskel for all the rules:

$$\begin{array}{lll}
\sigma & \models^I & True \\
\sigma & \models^I & (a_0 \leq a_1), \quad \text{iff} \quad \mathcal{A}v[\![a_0]\!]I\sigma \leq \mathcal{A}v[\![a_1]\!]I\sigma \\
\sigma & \models^I & (A_0 \Rightarrow A_1), \quad \text{iff} \quad (not\ \sigma \models^I A) \ \ or \ \ (\sigma \models^I A_1) \\
\sigma & \models^I & (\forall i.A) \quad \text{iff} \quad \forall n\ .\ (\sigma \models^{I[i \mapsto n]} A) \\
\perp & \models^I & A
\end{array}$$

Claims:

$$\begin{array}{lrcl}
\forall a \in \textbf{Aexp}, & \mathcal{A}[\![a]\!]\sigma & = & \mathcal{A}v[\![a]\!]I\sigma \\
\forall b \in \textbf{Bexp}, & \mathcal{B}[\![b]\!]\sigma & = & \textbf{true}, \quad \text{iff} \quad \forall I, \sigma \models^I b
\end{array}$$

Proof. By structural induction, left as exercise.