

Lectures #7–8: Denotational Semantics of IMP

CS 6371: Advanced Programming Languages

September 11, 2008

1 Syntax

commands	$c ::= \text{skip} \mid c_1; c_2 \mid v := a \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c$
boolean expressions	$b ::= \text{true} \mid \text{false} \mid a_1 \leq a_2 \mid b_1 \ \&\& \ b_2 \mid b_1 \ \ \ \ b_2 \mid !b$
arithmetic expressions	$a ::= n \mid v \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$
variable names	v
integer constants	n

2 Denotational Semantics

2.1 Semantic Domain

$$\begin{aligned}\Sigma &= v \rightarrow \mathbb{Z} \\ \mathcal{A} &: a \rightarrow (\Sigma \rightarrow \mathbb{Z}) \\ \mathcal{B} &: b \rightarrow (\Sigma \rightarrow \{T, F\}) \\ \mathcal{C} &: c \rightarrow (\Sigma \rightarrow \Sigma)\end{aligned}$$

2.2 Arithmetic Expressions

Function \mathcal{A} takes an IMP arithmetic expression as its input and returns an equivalent mathematical function from stores to integers.

$$\begin{aligned}\mathcal{A}[n] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\ \mathcal{A}[x] &= \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in \text{Dom}(\sigma)\} \\ \mathcal{A}[a_1 + a_2] &= \{(\sigma, n_1 + n_2) \mid (\sigma, n_1) \in \mathcal{A}[a_1], (\sigma, n_2) \in \mathcal{A}[a_2]\} \\ \mathcal{A}[a_1 - a_2] &= \{(\sigma, n_1 - n_2) \mid (\sigma, n_1) \in \mathcal{A}[a_1], (\sigma, n_2) \in \mathcal{A}[a_2]\} \\ \mathcal{A}[a_1 * a_2] &= \{(\sigma, n_1 n_2) \mid (\sigma, n_1) \in \mathcal{A}[a_1], (\sigma, n_2) \in \mathcal{A}[a_2]\}\end{aligned}$$

2.3 Boolean Expressions

Function \mathcal{B} takes an IMP boolean expression as its input and returns an equivalent mathematical function from stores to truth values.

$$\begin{aligned}
\mathcal{B}[\mathbf{true}] &= \{(\sigma, T) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\mathbf{false}] &= \{(\sigma, F) \mid \sigma \in \Sigma\} \\
\mathcal{B}[a_1 \leq a_2] &= \{(\sigma, T) \mid (\sigma, n_1) \in \mathcal{A}[a_1], (\sigma, n_2) \in \mathcal{A}[a_2], n_1 \leq n_2\} \cup \\
&\quad \{(\sigma, F) \mid (\sigma, n_1) \in \mathcal{A}[a_1], (\sigma, n_2) \in \mathcal{A}[a_2], n_1 > n_2\} \\
\mathcal{B}[b_1 \ \&\& \ b_2] &= \{(\sigma, T) \mid (\sigma, T) \in \mathcal{B}[b_1], (\sigma, T) \in \mathcal{B}[b_2]\} \cup \\
&\quad \{(\sigma, F) \mid (\sigma, F) \in \mathcal{B}[b_1]\} \cup \\
&\quad \{(\sigma, F) \mid (\sigma, F) \in \mathcal{B}[b_2]\} \\
\mathcal{B}[b_1 \ || \ b_2] &= \{(\sigma, T) \mid (\sigma, T) \in \mathcal{B}[b_1]\} \cup \\
&\quad \{(\sigma, T) \mid (\sigma, T) \in \mathcal{B}[b_2]\} \cup \\
&\quad \{(\sigma, F) \mid (\sigma, F) \in \mathcal{B}[b_1], (\sigma, F) \in \mathcal{B}[b_2]\} \\
\mathcal{B}[\mathbf{!}b] &= \{(\sigma, T) \mid (\sigma, F) \in \mathcal{B}[b]\} \cup \\
&\quad \{(\sigma, F) \mid (\sigma, T) \in \mathcal{B}[b]\}
\end{aligned}$$

2.4 Commands

Function \mathcal{C} takes an IMP command as input and returns an equivalent mathematical function from stores to stores.

$$\begin{aligned}
\mathcal{C}[\mathbf{skip}] &= \{(\sigma, \sigma) \mid \sigma \in \Sigma\} = \iota_{\Sigma \rightarrow \Sigma} \\
\mathcal{C}[c_1; c_2] &= \{(\sigma, \mathcal{C}[c_2](\mathcal{C}[c_1](\sigma))) \mid \sigma \in \Sigma\} = \mathcal{C}[c_2] \circ \mathcal{C}[c_1] \\
\mathcal{C}[v := a] &= \{(\sigma, \sigma[v \mapsto n]) \mid (\sigma, n) \in \mathcal{A}[a]\} \\
\mathcal{C}[\mathbf{if} \ b \ \mathbf{then} \ c_1 \ \mathbf{else} \ c_2] &= \{(\sigma, \mathcal{C}[c_1](\sigma)) \mid (\sigma, T) \in \mathcal{B}[b]\} \cup \\
&\quad \{(\sigma, \mathcal{C}[c_2](\sigma)) \mid (\sigma, F) \in \mathcal{B}[b]\} \\
\mathcal{C}[\mathbf{while} \ b \ \mathbf{do} \ c] &= \bigcup_{i \geq 0} \Gamma^i(\perp_{\Sigma \rightarrow \Sigma}) = \mathit{fix}(\Gamma) \\
\text{where } \Gamma(f) &= \{(\sigma, (f \circ \mathcal{C}[c])(\sigma)) \mid (\sigma, T) \in \mathcal{B}[b]\} \cup \\
&\quad \{(\sigma, \sigma) \mid (\sigma, F) \in \mathcal{B}[b]\}
\end{aligned}$$

Here, $\perp_{\Sigma \rightarrow \Sigma} = \{\}$. That is, $\perp_{\Sigma \rightarrow \Sigma}$ is the partial function from stores to stores that is undefined for all inputs. Function Γ takes a partial function as input and returns a partial function as output. So Γ has type $\Gamma : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$. The operator $\mathit{fix}()$ denotes the *least fixed point* of its argument. That is, $\mathit{fix}(\Gamma)$ yields the smallest set f such that $\Gamma(f) = f$. So the result of $\mathit{fix}(\Gamma)$ is a function of type $\Sigma \rightarrow \Sigma$.