Scribe: Jaime H. Barrera Lecturer: Radu Rugina

We have described the behavior of IMP using operational semantics. The operational semantics work well to describe the meaning of IMP programs making it ideal for building an interpreter. Operational semantics have the disadvantage of mixing syntax into the description of the behavior of programs. Therefore, comparing two different languages or translating a program written in one language into a program written in another using operational semantics is difficult. To remedy this we should define a new model for semantics that will describe the meaning of programs in terms of an underlying semantics domain. To this end, we introduce denotational semantics which map syntax into a representation of that syntax.

1 Denotationl Semantics

We will use the following

$$[\![\langle syntax\rangle]\!] = \text{denotation of } \langle\text{syntax}\rangle$$

for the denotation of ⟨syntax⟩, where $[\![\cdot]\!]$ are called semantics brackets. We now express the semantics of the IMP language, starting with arithmetic expressions.

1.1 Aexp

The denotational sematics of $a \in \mathbf{Aexp}$ can be expressed by a function

$$\mathcal{A} : \mathbf{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z}).$$

For $a \in \mathbf{Aexp}$ and $\sigma \in \Sigma$ we write

$$\begin{aligned} \mathcal{A}[\![a]\!] &\in \Sigma \rightarrow \mathbb{Z}, \\ \mathcal{A}[\![a]\!]\sigma &\in \mathbb{Z}. \end{aligned}$$

1.2 Bexp

The denotational sematics of $b \in \mathbf{Bexp}$ can be expressed by a function

$$\mathcal{B} : \mathbf{Bexp} \rightarrow (\Sigma \rightarrow \mathbb{T}).$$

For $b \in \mathbf{Bexp}$ and $\sigma \in \Sigma$ we write

$$\begin{aligned} \mathcal{B}[\![b]\!] &\in \Sigma \rightarrow \mathbb{T}, \\ \mathcal{B}[\![b]\!]\sigma &\in \mathbb{T}. \end{aligned}$$

1.3 Com

The denotational sematics of $c \in \mathbf{Com}$ can be expressed by a function

$$\mathcal{C} : \mathbf{Com} \rightarrow (\Sigma \rightharpoonup \Sigma).$$

Note that here we used "$\rightharpoonup$" to indicate a partial function rather than "$\rightarrow$" , which we use for complete functions, since for some $\sigma \in \Sigma$ and some $c \in \mathbf{Com}$ the execution of c may not terminate, while writing

"$\mathcal{C} : \mathbf{Com} \to (\Sigma \to \Sigma)$" would imply that for all $\sigma \in \Sigma$ and for all $c \in \mathbf{Com}$, c terminates. For $c \in \mathbf{Com}$ and $\sigma \in \Sigma$ we write

$$\begin{aligned} \mathcal{C}[\![c]\!] &\in \Sigma \rightharpoonup \Sigma, \\ \mathcal{C}[\![c]\!]\sigma &\in \Sigma. \end{aligned}$$

2 Notation for functions

For a function $f : A \to B$ we can describe f as the set

$$\{(a_0, b_0), \ldots, (a_i, b_i), \ldots\},$$

where $b_i = f(a_i)$. With this notation, we see that $f \subseteq (A \times B)$. Hence:

$$(A \to B) \subseteq \mathcal{P}(A \times B)$$

where $\mathcal{P}(X)$ is the power set of X.

2.1 λ-notation

Alternatively, we can describe function using the λ-notation:

$$\lambda x.e = \{(x, e) \mid x \in A\}$$

expresses the function that maps x into the expression e.

3 Back to Denotational Semantics

3.1 Denotational semantics for Aexp, $\mathcal{A}[\![a]\!]$

For $a \in \mathbf{Aexp}$ we define $\mathcal{A}[\![a]\!]$ as

$$\begin{aligned} \mathcal{A}[\![n]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\ \mathcal{A}[\![X]\!] &= \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\} \\ \mathcal{A}[\![a_0 + a_1]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma \wedge n = \mathcal{A}[\![a_0]\!]\sigma + \mathcal{A}[\![a_1]\!]\sigma\}. \end{aligned}$$

$\mathcal{A}[\![a_0 - a_1]\!]$ and $\mathcal{A}[\![a_0 * a_1]\!]$ are defined similarly. For $a \in \mathbf{Aexp}$ we can express $\mathcal{A}[\![a]\!]$ in λ-notation as

$$\begin{aligned} \mathcal{A}[\![n]\!] &= \lambda\sigma \in \Sigma.\, n, \\ \mathcal{A}[\![X]\!] &= \lambda\sigma \in \Sigma.\, \sigma(X), \\ \mathcal{A}[\![a_0 + a_1]\!] &= \lambda\sigma \in \Sigma.\, \mathcal{A}[\![a_0]\!]\sigma + \mathcal{A}[\![a_1]\!]\sigma. \end{aligned}$$

$\mathcal{A}[\![a_0 - a_1]\!]$ and $\mathcal{A}[\![a_0 * a_1]\!]$ can be written in a similar way..

3.2 Denotational semantics for Bexp, $\mathcal{B}[\![b]\!]$

For $b \in \mathbf{Bexp}$ we define $\mathcal{B}[\![b]\!]$ as

$$
\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!] &= \{(\sigma, \mathbf{true}) \mid \sigma \in \Sigma\}, && (1)\\
\mathcal{B}[\![\mathbf{false}]\!] &= \{(\sigma, \mathbf{false}) \mid \sigma \in \Sigma\}, && (2)\\
\mathcal{B}[\![a_0 = a_1]\!] &= \{(\sigma, t) \mid \sigma \in \Sigma \wedge t = (\mathcal{A}[\![a_0]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma)\}.
\end{aligned}
$$

$\mathcal{B}[\![a_0 \leq a_1]\!]$ and $\mathcal{B}[\![\neg b]\!]$ can be defined similarly. Notice that in the right-hand sides of (1) and (2) the **true**/**false**s are truth values while the **true**/**false**s on the left-hand sides of (1) and (2) are pieces of syntax.

$$\mathcal{B}[\![b_0 \wedge b_1]\!] = \{(\sigma, t) \mid \sigma \in \Sigma \wedge t = (\mathcal{B}[\![b_0]\!]\sigma \wedge \mathcal{B}[\![b_1]\!]\sigma)\}$$

defines $\mathcal{B}[\![b_0 \wedge b_1]\!]$, while $\mathcal{B}[\![b_0 \vee b_1]\!]$ is defined similarly.

3.3 Denotational semantics for Com, $\mathcal{C}[\![c]\!]$

For $c \in \mathbf{Com}$ we define $\mathcal{C}[\![c]\!]$ as

$$
\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!] &= \{(\sigma, \sigma) \mid \sigma \in \Sigma\},\\
\mathcal{C}[\![X := a]\!] &= \{(\sigma, \sigma[X \to \mathcal{A}[\![a]\!]\sigma]) \mid \sigma \in \Sigma\}.
\end{aligned}
$$

3.4 Non-termination

We represent commands that don't terminate using partial functions, as in the following examples:
Let $w_1 \equiv (\mathbf{while\,true\,do\,skip})$ then

$$\mathcal{C}[\![w_1]\!] = \emptyset$$

and if $w_2 \equiv (\mathbf{while}\,x = 0\,\mathbf{do\,skip})$ then

$$\mathcal{C}[\![w_2]\!] = \{(\sigma, \sigma) \mid \sigma \in \Sigma \wedge \sigma(X) \neq 0\}$$

and there is no mapping if $\sigma(X) = 0$.
An alternative is to represent nontermination explictly, using a special state that characterizes non-termination. This motivates the next section.

3.5 $\perp$

Read "bottom", the symbol $\perp$ refers to non-termination. We can now express the the semantics of commands as a complete function by adding onto Σ the non-terminating state $\perp$ in the following manner

$$\mathcal{C}[\![c]\!] : \Sigma \to \Sigma_\perp$$

where $\Sigma_\perp = \Sigma \cup \{\perp\}$. In operational semantics the only way to express that a command did not terminate would be to point out that no finite proof tree existed for such a command. Denotational semantics explicitly takes care of this with bottom.

3.6 Denotational semantics for Com, $\mathcal{C}[\![c]\!]$ (cont'd)

We now attempt to complete the definition of the commands in denotational semantics

$$
\begin{aligned}
\mathcal{C}[\![c_0; c_1]\!] \quad &:= \quad \{(\sigma, \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)) \mid \sigma \in \Sigma \wedge \mathcal{C}[\![c_0]\!]\sigma \neq \perp\} \\
& \qquad \cup \{(\sigma, \perp) \mid \sigma \in \Sigma \wedge \mathcal{C}[\![c_0]\!]\sigma = \perp\}, \\
\mathcal{C}[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!] \quad &:= \quad \{(\sigma, \mathcal{C}[\![c_0]\!]\sigma) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma\} \\
& \qquad \cup \{(\sigma, \mathcal{C}[\![c_1]\!]\sigma) \mid \sigma \in \Sigma \wedge \neg\mathcal{B}[\![b]\!]\sigma\}, \\
\mathcal{C}[\![w]\!] \quad &:= \quad \{(\sigma, \mathcal{C}[\![\mathbf{skip}]\!]\sigma) \mid \sigma \in \Sigma \wedge \neg\mathcal{B}[\![b]\!]\sigma\} \\
& \qquad \cup \{(\sigma, \mathcal{C}[\![c; w]\!]\sigma) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma\} \\
&= \quad \{(\sigma, \sigma) \mid \sigma \in \Sigma \wedge \neg\mathcal{B}[\![b]\!]\sigma\} \\
& \qquad \cup \{(\sigma, \perp) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma \wedge \mathcal{C}[\![c]\!]\sigma = \perp\} \\
& \qquad \cup \{(\sigma, \mathcal{C}[\![w]\!](\mathcal{C}[\![c]\!]\sigma)) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma \wedge \mathcal{C}[\![c]\!]\sigma \neq \perp\},
\end{aligned}
$$

where $w \equiv \mathbf{while}\, b\, \mathbf{do}\, c$. We see here that the the above relation for w is not a definition for $\mathcal{C}[\![w]\!]$, but rather a recursive equation. We will further explore the denotational semantics of the while loop in the next lecture.