CS611 Lecture 08 **Denotational Semantics** 15 September 2003

Scribe: Benyah Shaparenko Lecturer: Radu Rugina

1 Inductive Definitions

1.1 Definitions

$\hat{R}$ is an inductive operator, used for building inductively-defined sets. Intuitively, given a set of inference rules, the $\hat{R}$ operator takes a set X of elements and yields all elements that could be produced in one step using the inference rules, with elements in X as the premises. More formally, for any set X,

$$\hat{R}(X) = \{y \mid \exists X' \subseteq X. \ \frac{X'}{y} \ \}$$

To define A, the set of elements producible from applications of $\hat{R}$, we start with the empty set ϕ, and recursively apply $\hat{R}$ to get the sequence, ϕ, $\hat{R}(\phi)$, $\hat{R}(\hat{R}(\phi)) = \hat{R}^2(\phi)$, ... A is the union of this sequence. Intuitively, A consists of the set of elements that can be built from axioms, then from derivations of one level, then two levels, etc. Or, equivalently, the set of elements having finite proof trees. Formally,

$$\begin{aligned} A &= \hat{R}(\phi) \cup \hat{R}^2(\phi) \cup ... \\ &= \bigcup_{n \in \mathbb{N}} \hat{R}^n(\phi) \end{aligned}$$

The proof trees for elements in $\hat{R}^n(\phi)$ can have height less than n and don't necessarily need to be full trees.

1.2 Properties of A

1. A is closed: $\hat{R}(\mathrm{A}) \subseteq \mathrm{A}$. This means that if the rules are applied to A, we get nothing more than what we already have in A.
2. A is consistent: $A \subseteq \hat{R}(A)$. This means that each element in A can be derived from A.
3. Properties 1 and 2 imply that A is a fixed point for $\hat{R}$: $\hat{R}(A) = A$.
4. A is the least fixed point of $\hat{R}$. This means that for all sets B such that $\hat{R}(B) = B$, $A \subseteq B$. This means that any other fixed point of $\hat{R}$ includes A.

1.3 Proofs of These Properties

1. This was proved in the previous lecture.
2. Let x $\in A$. Then $\exists n.\ x \in \hat{R}^n(\phi)$.
   From the definition of A, $\hat{R}^{n-1}(\phi) \subseteq A$.
   $\hat{R}$ is monotonic, so we can apply $\hat{R}$ to each side of $\hat{R}^{n-1}(\phi) \subseteq A$.
   This yields $\hat{R}(\hat{R}^{n-1}(\phi)) \subseteq \hat{R}(A)$.
   But $\hat{R}(\hat{R}^{n-1}(\phi)) = \hat{R}^n(\phi)$.
   So, $\hat{R}^n(\phi) \subseteq \hat{R}(A)$.
   Therefore, $x \in \hat{R}(A)$ since $x \in \hat{R}^n(\phi)$.
3. This follows trivially.
4. Let B be an arbitrary element of $\{B \mid \hat{R}(B) = B\}$. We want to show $A \subseteq B$.
   Let $x \in A$. Then, $\exists m.\ x \in \hat{R}^m(\phi)$ by definition of A.
   We can now prove that $\forall n.\ \hat{R}^n(\phi) \subseteq B$ by induction on n.

In the base case ($n = 0$): $\hat{R}^0(\phi) = \phi \subseteq B$, which is trivial.
In the induction step: Assume $\hat{R}^n(\phi) \subseteq B$. We want to show $\hat{R}^{n+1}(\phi) \subseteq B$.
Since $\hat{R}$ is monotonic, we can apply $\hat{R}$ to each side of our induction hypothesis.
This yields $\hat{R}(\hat{R}^n(\phi)) \subseteq \hat{R}(B)$.
But $\hat{R}(\hat{R}^n(\phi)) = \hat{R}^{n+1}(\phi)$ and $\hat{R}(B) = B$ by our choice of B.
So, $\hat{R}^{n+1}(\phi) \subseteq B$.
Therefore, $x \in B$.
So, $A \subseteq B$.

1.4 Conclusions about Inductive Definitions

We will see later how fixed points are used for the denotational semantics of **while** loops.

In general, there may be multiple fixed points for the operator $\hat{R}$, but there is always a unique least fixed point. One good example is the set defined by these rules:

$$\frac{}{0} \qquad \frac{n}{succ(n)}$$

In this example, $\mathbb{N}$ is the least fixed point and $\mathbb{Z}$ is a different fixed point.

We have been using $\hat{R}$, but there is another common operator called $\bar{R}$, defined as $\bar{R}(X) = X \cup \hat{R}(X)$. It is possible to show that $\bar{R}$ is monotonic. It is also possible to show that $\bar{R}(\phi) = \hat{R}(\phi)$ and, in general, $\bar{R}^n(\phi) = \hat{R}^n(\phi)$. Note, however, that $\bar{R}(X) \neq \hat{R}(X)$ for arbitrary X. Only when $X = \phi$ does $\bar{R}(X) = \hat{R}(X)$.

2 Denotational Semantics

2.1 Introduction to Denotational Semantics

The goal of denotational semantics is to express the meaning of programs as mathematical functions. This approach abstract away the syntax in the program. One can further implement the abstract mathematical function in a different language. Therefore, we can think of denotational semantics in the context of compilers or translators. In contrast, operational semantics provided a model of execution that is easily mapped to an interpreter.

The notation $[\![\ ]\!]$ used in denotational semantics is called *semantic brackets*. $[\![\langle\mathsf{syntax}\rangle]\!]$ is read as "the denotation of $\langle\mathsf{syntax}\rangle$". The semantic brackets are essentially a mathematical function. In the following subsections, we present the denotational semantics for IMP.

2.2 Definitions for Aexp

The denotational semantics for $a \in \mathbf{Aexp}$ is a function:

$$\mathcal{A}[\![a]\!] : \Sigma \to \mathbb{Z}$$

This means that evaluation is represented as:

$$\mathcal{A}[\![a]\!]\sigma = z, \text{ where } z \in \mathbb{Z}$$

Note that alternatively, we could write:

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

2.3 Definitions for Bexp

The denotational semantics for $b \in \mathbf{Bexp}$ is a function:

$$\mathcal{B}[\![b]\!] : \Sigma \to \mathbb{T}$$

This means that evaluation is represented as:

$$\mathcal{B}[\![b]\!]\sigma \in \{true, false\}$$

2.4 Definitions for Com

We cannot use $\mathcal{C}[\![c]\!] : \Sigma \to \Sigma$ to represent the denotational semantics for $c \in \mathbf{Com}$ since evaluation of commands does not have to terminate. So, the denotational semantics for $c \in \mathbf{Com}$ is a partial function:

$$\mathcal{C}[\![c]\!] : \Sigma \rightharpoonup \Sigma$$

Another alternative is to use $\perp$ to signify non-termination. This is the method that we will actually use. Let $\Sigma_\perp = \Sigma \cup \{\perp\}$. Then,

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

3 Notation for Functions

3.1 Ordered Pair Notation

$A \to B$ represents the set of all total functions from A to B. Let $f \in A \to B$. Then f can be represented by a set of ordered pairs:

$$\{(a_1, b_1), ..., (a_i, b_i), ...\},$$

where $f(a_i) = b_i$ with all the a_i distinct, and $\bigcup_i a_i = A$. Note that $(A \to B) \subseteq \mathcal{P}(A \times B)$ and that $f \subseteq A \times B$.

3.2 λ-notation

The λ-notation is:

$$\lambda x.e$$

This notation expresses the function that maps x to e. For example, the successor function can be written as $\lambda x.x + 1$.

4 Denotational Semantics for IMP

4.1 Denotational Semantics for Aexp

We will define the denotational semantics $\mathcal{A}[\![a]\!]$ for $a \in \mathbf{Aexp}$ based on the structure of arithmetic expressions:

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

$\mathcal{A}[\![a_0 - a_1]\!]$ and $\mathcal{A}[\![a_0 \times a_1]\!]$ are similar to $\mathcal{A}[\![a_0 + a_1]\!]$. Alternatively, we can write the denotational semantics using the λ-notation:

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

4.2 Denotational Semantics for Bexp

Next, we define the denotational semantics $\mathcal{B}[\![b]\!] : \Sigma \to \mathbb{T}$ for $b \in \mathbf{Bexp}$ based on the structure of boolean expressions:

$$
\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!] &= \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![\mathbf{false}]\!] &= \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![a_0 < a_1]\!] &= \{(\sigma, t) \mid \sigma \in \Sigma \wedge t = \mathcal{A}[\![a_0]\!]\sigma < \mathcal{A}[\![a_1]\!]\sigma\} \\
\mathcal{B}[\![b_0 \wedge b_1]\!] &= \{(\sigma, t) \mid \sigma \in \Sigma \wedge t = \mathcal{B}[\![b_0]\!]\sigma \wedge \mathcal{B}[\![b_1]\!]\sigma\} \\
\mathcal{B}[\![\neg b_0]\!] &= \{(\sigma, \neg\mathcal{B}[\![b_0]\!]\sigma) \mid \sigma \in \Sigma\}
\end{aligned}
$$

$\mathcal{B}[\![a_0 = a_1]\!]$ is similar to $\mathcal{B}[\![a_0 < a_1]\!]$. $\mathcal{B}[\![b_0 \vee b_1]\!]$ is similar to $\mathcal{B}[\![b_0 \wedge b_1]\!]$.

4.3 Denotational Semantics for Com

Next, we define the denotational semantics $\mathcal{C}[\![c]\!] : \Sigma \to \Sigma_\perp$ for $c \in \mathbf{Com}$ based on the structure of commands of IMP:

$$
\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!] &= \{(\sigma, \sigma) \mid \sigma \in \Sigma\} \\[1em]
\mathcal{C}[\![x := a]\!] &= \{(\sigma, \sigma[x \mapsto n]) \mid \sigma \in \Sigma \wedge n = \mathcal{A}[\![a]\!]\sigma\} \\[1em]
\mathcal{C}[\![c_0; c_1]\!] &= \{(\sigma, \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)) \mid \sigma \in \Sigma \wedge \mathcal{C}[\![c_0]\!]\sigma \neq \perp\} \\
&\cup\ \{(\sigma, \perp) \mid \sigma \in \Sigma \wedge \mathcal{C}[\![c_0]\!]\sigma = \perp\} \\[1em]
\mathcal{C}[\![\mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1]\!] &= \{(\sigma, \mathcal{C}[\![c_0]\!]\sigma) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma = \mathit{true}\} \\
&\cup\ \{(\sigma, \mathcal{C}[\![c_1]\!]\sigma) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma = \mathit{false}\}
\end{aligned}
$$

For loops, we try to write the semantics of $w \equiv \mathbf{while}\, b\, \mathbf{do}\, c$ by unrolling the loop as $\mathbf{if}\, b\, \mathbf{then}\, c; w\, \mathbf{else}\, \mathbf{skip}$. We get:

$$
\begin{aligned}
\mathcal{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!] &= \{(\sigma, \sigma) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma = \mathit{false}\} \\
&\cup\ \{(\sigma, \perp) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma = \mathit{true} \wedge \mathcal{C}[\![c]\!]\sigma = \perp\} \\
&\cup\ \{(\sigma, \sigma') \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma = \mathit{true} \wedge \mathcal{C}[\![c]\!]\sigma \neq \perp \wedge \sigma' = \mathcal{C}[\![\mathbf{while}\, b\, \mathbf{do}\, c]\!](\mathcal{C}[\![c]\!]\sigma)\}
\end{aligned}
$$

We are not entirely happy with the last definition, since it is merely a recursive equation and not a function definition. We will have to solve it for a fixed point, which we will see next time.