CS520 Lecture 6
Transition Semantics

March 22, 2020

6.1 Motivation or objective

1. So far we defined the meanings of programs in imperative languages using the denotational semantics. A good denotational semantics reveals an underlying mathematical structure of a programming language and hides the intermediate stepf of computation as much as possible. Also, it is compositional, and lets us reason about a piece of program code even when we do not know its surrounding program context.
2. However, when a programming language has advanced or complex language contructs, defining a denotational semantics of the language may be difficult. Also, sometimes we want to have a mathematical semantics of programs that tells us what happens in the middle of computation.
3. The operational semantics is an alternative approach to give a mathematical meanings to programs. It is non-compositional, and does not hide the intermediate step of computation. But it is usually very simple and also rigorous or formal enough to enable a mathematical study of a programming language and language tools such as compiler and program verifier. Also, an operational semantics of a programming language often serves as a blue print of an interpreter or a compiler of the language.
4. In this chapter, we will study so called $\underline{\text{small-step}}$ operational semantics, which Reynolds calls transition semantics.

6.2 Main idea of the small-step operational semantics

1. The key idea is to formalise one computation step of a program using a relation, called transition relation.
2. Typically, a small-step operational semantics has two main parts.

   1. $\Gamma \cdots$ a set of configurations.
      Usually, $\Gamma = \Gamma_N \cup \Gamma_T$ for some Γ_N, Γ_T with $\Gamma_N \cap \Gamma_T = \emptyset$.
      Each element $\gamma \in \Gamma$ describes the status of a machine that runs a program. If $\gamma \in \Gamma_N$, it is called $\underline{\text{nonterminal configuration}}$ and its program is not finished yet. If $\gamma \in \Gamma_T$, it is called $\underline{\text{terminal configuration}}$ and the program of its program is completed.

   2. $\to \subseteq \Gamma_N \times \Gamma \cdots$ transition relation.
      Intuitively, $(\gamma, \gamma') \in \to$ (typically written as $\gamma \to \gamma'$) means that one computation step changes the status of a machine from γ to γ'. Note that γ has to be a nonterminal configuration, because of Γ_N.
      This condition is consistent with the intuition behind nonterminal and terminal configurations.

3. Defining a small-step operational semantics amounts to defining $\Gamma, \Gamma_N, \Gamma_T$ and $\to$. We will see a few examples of the operatinal semantics in this lecture. Often if we define $\Gamma, \Gamma_N, \Gamma_T$, then the definition of $\to$ follows almost automatically. This is a bit similar to the situation in the denotational semantics that if the form of the interpretation function for commands $[\![-]\!]$ is determined, the actual definition of the function follows almost automatically.

4. When defining the $\to$ relation, we usually use the inference rule notation $\dfrac{\psi_1 \quad \dots \quad \psi_n}{\psi}$ that you saw when we discussed Hoare logic.

6.3 Small-step operational semantics of the simple imperative language

1. Let's try to give the operational semantics to the simple imperative language that we studied. Here is a reminder of its abstract grammar:

$$\begin{aligned}\langle\text{comm}\rangle ::= {} & \text{skip} \mid \langle\text{var}\rangle := \langle\text{intexp}\rangle \mid \langle\text{comm}\rangle; \langle\text{comm}\rangle \\ & \mid \text{if } \langle\text{boolexp}\rangle \text{ then } \langle\text{comm}\rangle \text{ else } \langle\text{comm}\rangle \\ & \mid \text{while } \langle\text{boolexp}\rangle \text{ do } \langle\text{comm}\rangle\end{aligned}$$

2. What should we do? First, we have to define the set of nonterminal configurations and that of terminal configurations. Here are our definitions.

$$\Gamma_N \overset{def}{=} \langle\text{comm}\rangle^{1} \times \Sigma^{23} \qquad\qquad \Gamma_T \overset{def}{=} \Sigma^{4}$$

The set of configurations is the union of the above two sets.
3. Second, we should define a binary relation

$$\to \subseteq \Gamma_N \times \Gamma$$

called transition relation, that describes single-step computation. We write $\gamma \to \gamma'$ to mean $(\gamma, \gamma') \in \to$. We define the transition relation $\to$ using the inference rule notation.

$$\frac{}{\langle skip, \sigma\rangle \to \sigma}$$

$$\frac{}{\langle v := e, \sigma\rangle \to [\sigma | v : [\![e]\!]\sigma]}$$

$$\frac{\langle c_1, \sigma\rangle \to \sigma'}{\langle c_1; c_2, \sigma\rangle \to \langle c_2, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to \langle c_1'; c_2, \sigma'}$$

[1]command that records the ramaining computation
[2]the current state of a machine
[3]the set of states, i.e. $[\langle\text{var}\rangle \to \mathbb{Z}]$
[4]The $\langle\text{comm}\rangle$ is missing because there is no remaining computation

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma\rangle \to \langle c_1, \sigma\rangle}\ (\llbracket b \rrbracket\sigma = tt)$$

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma\rangle \to \langle c_2, \sigma\rangle}\ (\llbracket b \rrbracket\sigma = ff)$$

$$\frac{}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma}\ (\llbracket b \rrbracket\sigma = ff)$$

$$\frac{}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \langle c; \text{while } b \text{ do } c, \sigma\rangle}\ (\llbracket b \rrbracket\sigma = tt)$$

Note that the right-hand side of $\to$ may include a command that is not a sub-command of the one on the left-hand side. Look at 6.3 and 6.3. This indicates that the semantics is not compositional. All these rules correspond to our intuitive understandinf of one computation step. They can form the basis of the implementation of a simple interpreter, which just needs to run the $\to$ step repeatedly.
4. Formal properties of the operational semantics.

1. $\gamma \to \gamma_1$ and $\gamma \to \gamma_2 \Rightarrow \gamma_1 = \gamma_2$
   The semantics is deterministic.

2. $\forall \gamma \in \Gamma_N . \exists \gamma'$ s.t. $\gamma \to \gamma'$
   In this semantics, executions never get stuck.

3. From 1. and 2. it follows that for every $\gamma \in \Gamma$, there exists a unique maximal sequence.

   $$\gamma_0, \gamma_1, \ldots, \gamma_n{}^5$$

   such that

   $$\gamma = \gamma_0 \wedge \gamma_0 \to \gamma_1 \to \ldots \to \gamma_n \wedge (\gamma_n \in \Gamma_T \text{ or } n \text{ is infinite})$$

   This maximal finite or infinite sequence represents the full computation starting from γ.

4. We write $\gamma \uparrow$ if the maximal execution sequence from γ is infinite. Then, for all commands c and states σ,

   $$\llbracket c \rrbracket\sigma = \bot \text{ iff } \langle c, \sigma\rangle \uparrow$$

   $$\llbracket c \rrbracket\sigma = \sigma' \text{ iff } \langle c, \sigma\rangle \to^* \sigma'{}^6$$

exercise Prove 1, 2, and 4.
exercise Explain why the reasoning in 3 is true.

[5] may be infinite

[6] reflexive and transitive closure of $\to$. i.e., $\to^* \overset{def}{=} \cup_{n=0}^{\infty} (\to)^n$

6.4 Extension with newvar

1. Extended the language with variable declaration:

$$\langle \text{comm} \rangle ::= \ldots \mid \text{newvar } \langle \text{var} \rangle := \langle \text{intexp} \rangle \text{ in } \langle \text{comm} \rangle$$

2. How should we modify the $\rightarrow$ relation? Add a rule for newvar.
1) Option 1.

$$\frac{}{\langle \text{newvar } v := e \text{ in } c, \sigma \rangle \rightarrow \langle c; v := n, [\sigma | v : [\![e]\!]\sigma] \rangle} \text{ where } n = \sigma(v)$$

2) Option 2.

$$\frac{\langle c, [\sigma | v : [\![e]\!]\sigma] \rangle \rightarrow \sigma'}{\langle \text{newvar } v := e \text{ in } c, \sigma \rangle \rightarrow [\sigma' | v : \sigma(v)]}$$

3) Both options are acceptable. But 2) is better. Only 2) works when we extend the language with primitives for concurrent executions.
3. Note that we did not change Γ_N, Γ_T. Thus, adding newvar doesn't change the operational semantics much. In a sense, this small change means that newvar doesn't change the language much, either.

6.5 Adding fail

$$\langle \text{comm} \rangle ::= \ldots \mid fail$$

1. When we add fail, we have to change the set Γ_T of terminal configuration, because we now have two types of terminations, normal one and abnormal one.

$$\Gamma_T \overset{def}{=} \hat{\Sigma} = \Sigma \cup \{abort\} \times \Sigma (\text{or } = \Sigma + \Sigma)$$

Γ_N remains unchanged.
2. Since Γ_T and so Γ are changed, we should change the definition of $\rightarrow$. We will also have to add a rule for fail. Here is the new set of rules.

$$\frac{}{\langle fail, \sigma \rangle \rightarrow \underbrace{\langle abort, \sigma \rangle}_{terminalconfiguration}}$$

$$\frac{}{\langle skip, \sigma \rangle \rightarrow \sigma} \quad \frac{}{\langle v := e, \sigma \rangle \rightarrow [\sigma | v : [\![e]\!]\sigma]}$$

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle} \quad [\![b]\!]\sigma = tt$$

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle} \quad [\![b]\!]\sigma = ff$$

$$\frac{\langle c_1, \sigma\rangle \to \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to \langle c_1'; c_2, \sigma'\rangle} \quad \frac{\langle c_1, \sigma\rangle \to \sigma'}{\langle c_1; c_2, \sigma\rangle \to \langle c_2, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to \langle abort, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to \langle abort, \sigma'\rangle}$$

$$\frac{}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \sigma} \; [\![b]\!]\sigma = ff$$

$$\frac{}{\langle \text{while } b \text{ do } c, \sigma\rangle \to \langle c; \text{while } b \text{ do } c, \sigma\rangle} \; [\![b]\!]\sigma = tt$$

$$\frac{\langle c, [\sigma|v : [\![e]\!]\sigma]\rangle \to \langle abort, \sigma'\rangle}{\langle \text{newvar } v := e \text{ in } c, \sigma\rangle \to \langle abort, [\sigma'|v : \sigma(v)]\rangle}$$

$$\frac{\langle c, [\sigma|v : [\![e]\!]\sigma]\rangle \to \sigma'}{\langle \text{newvar } v := e \text{ in } c, \sigma\rangle \to [\sigma'|v : \sigma(v)]}$$
$$\frac{\langle c, [\sigma|v : [\![e]\!]\sigma]\rangle \to \langle c', \sigma'\rangle}{\langle \text{newvar } v := e \text{ in } c, \sigma\rangle \to \langle \text{newvar } v := \sigma'(v) \text{ in } c', [\sigma'|v : \sigma(v)]\rangle}$$

6.6 Handling input and output

$$\langle\text{comm}\rangle ::= \ldots |?\langle\text{var}\rangle|!\langle\text{intexp}\rangle$$

1. This time we have to change the form or type of $\to$. It is no longer a binary relation, but a ternary relation

$$\to \subseteq \Gamma_N \times \Lambda \times \Gamma$$

$$\lambda \in \Lambda \overset{def}{=} \{\epsilon\}^{7} \cup \{?n|n \in \mathbb{Z}\}^{8} \cup \{!n|n \in \mathbb{Z}\}^{9}$$

We write $\langle c, \sigma\rangle \xrightarrow{\lambda} \gamma$ to mean $\langle\langle c, \sigma\rangle, \lambda, \gamma\rangle \in \to$. We also often omit λ if $\lambda = \epsilon$.
2. Why do we make this change? It is because adding $?v$ and $!e$ to the language makes it necessary to describe some aspects of intermediate steps of computatiosn explicitly.
3. We include all the rules (except the ones for $c_1; c_2$ and newvar) that we defined in 5. Of course, the occurence of $\to$ in those old rules should be understood as $\xrightarrow{\epsilon}$ with ϵ omitted for simplicity. In addition to these rules, we have the follwing rules:

[7] transition or execution without input or output
[8] transition with an input
[9] transition with an output

$$\frac{}{\langle ?v,\sigma\rangle \xrightarrow{?n} [\sigma|v:n]} \qquad \frac{}{\langle !e,\sigma\rangle \xrightarrow{!\llbracket e\rrbracket\sigma} \sigma}$$

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \sigma'}{\langle c_0;c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_1,\sigma'\rangle} \qquad \frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \langle c_0',\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_0';c_1,\sigma'\rangle} \qquad \frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \langle abort,\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \xrightarrow{\lambda} \langle abort,\sigma'\rangle}$$

$$\frac{\langle c,[\sigma|v:\llbracket e\rrbracket\sigma]\rangle \xrightarrow{\lambda} \sigma'}{\langle \text{newvar } v := e \text{ in } c,\sigma\rangle \xrightarrow{\lambda} [\sigma'|v:\sigma(v)]}$$

$$\frac{\langle c,[\sigma|v:\llbracket e\rrbracket\sigma] \xrightarrow{\lambda} \langle abort,\sigma'\rangle}{\langle \text{newvar } v := e \text{ in } c,\sigma\rangle \xrightarrow{\lambda} \langle abort,[\sigma'|v:\sigma(v)]}$$

$$\frac{\langle c,[\sigma|v:\llbracket e\rrbracket\sigma]\rangle \xrightarrow{\lambda} \langle c',\sigma'\rangle}{\langle \text{newvar } v := e \text{ in } c,\sigma\rangle \xrightarrow{\lambda} \langle \text{newvar } v := \sigma'(v) \text{ in } c',[\sigma'|v:\sigma(v)]\rangle}$$

Whenever an old rule contains a premise, we copy the rule and put λ above $\to$ in the premise and the concolusion, so that the label λ gets propagated from the execution of a subcommand to that of the original command.

4. The operational semantics corresponds to the denotational semantics that we studied. The correspondence is formalised by the function F in p134 of the textbook. Intuitively, F runs a configuration until it finishes, or outputs a number, or waits for an input. F then returns what it gets when it completes this execution. In a sense, the correspondence says hat the denotational semantics comes from the operational semantics after unobservable intermediate states are abstracted away. For detail, look at the textbook.