22 Oct 2018.

Chap6 Transition Semantics.

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 steps 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 constructs, 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 steps 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 blueprint of an interpreter or a compiler of the language.

④ In this chapter, we will study the so called small-step operational semantics, which Reynolds calls transition semantics.

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.

(i) Γ ... a set of configurations.

Usually, $\Gamma = \Gamma_N \cup \Gamma_T$ for some Γ_N, Γ_T with $\Gamma_N \cap \Gamma_T = \phi$.

Each element $\gamma \in \Gamma$ describes the status of a machine that runs a program. If $\gamma \in \Gamma_N$, it is called nonterminal configuration and its program is not finished yet. If $\gamma \in \Gamma_T$, it is called terminal configuration and its program is completed (the execution of).

(ii) $\rightarrow \subseteq \Gamma_N \times \Gamma$ ... transition relation.

Intuitively, $(\gamma, \gamma') \in \rightarrow$ (typically written as $\gamma \rightarrow \gamma'$) 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 . 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 $\rightarrow$. We will see a few examples of the operational semantics in this lecture. Often if we define $\Gamma, \Gamma_N, \Gamma_T$, then the definition of $\rightarrow$ 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 $\rightarrow$ relation, we usually use the inference rule notation $\frac{\varphi_1 \quad \varphi_n}{\varphi}$ that you saw when we discussed Hoare logic