Semantics and Verification of Software

Lecture 4: Operational vs. Denotational Semantics of WHILE

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw13/

Summer Semester 2013

Outline

Execution of Statements

Remember:

$c ::= \texttt{skip} \mid x \texttt{ := } a \mid c_1 \texttt{;} c_2 \mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } b \texttt{ do } c \in Cmd$

Definition (Execution relation for statements)

For $c \in Cmd$ and $\sigma, \sigma' \in \Sigma$, the execution relation $\langle c, \sigma \rangle \to \sigma'$ is defined by the following rules:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, \sigma \rangle \to \sigma} \qquad \text{(asgn)}\ \frac{\langle a, \sigma \rangle \to z}{\langle x \texttt{ := } a, \sigma \rangle \to \sigma[x \mapsto z]}$$

$$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \to \sigma' \quad \langle c_2, \sigma' \rangle \to \sigma''}{\langle c_1 \texttt{;} c_2, \sigma \rangle \to \sigma''} \qquad \text{(if-t)}\ \frac{\langle b, \sigma \rangle \to \mathsf{true} \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'}$$

$$\text{(if-f)}\ \frac{\langle b, \sigma \rangle \to \mathsf{false} \quad \langle c_2, \sigma \rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'} \qquad \text{(wh-f)}\ \frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \sigma}$$

$$\text{(wh-t)}\ \frac{\langle b, \sigma \rangle \to \mathsf{true} \quad \langle c, \sigma \rangle \to \sigma' \quad \langle \texttt{while } b \texttt{ do } c, \sigma' \rangle \to \sigma''}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \sigma''}$$

Determinism of Execution Relation I

This operational semantics is well defined in the following sense:

Theorem

The execution relation for statements is deterministic, i.e., whenever $c \in Cmd$ and $\sigma, \sigma', \sigma'' \in \Sigma$ such that $\langle c, \sigma \rangle \to \sigma'$ and $\langle c, \sigma \rangle \to \sigma''$, then $\sigma' = \sigma''$.

- How to prove this theorem?
- Idea:
  - employ corresponding result for expressions (Lemma 3.6)
  - use induction on the syntactic structure of c $\lightning$
- Instead: structural induction on derivation trees

Outline

Functional of the Operational Semantics

The determinism of the execution relation (Theorem 3.5) justifies the following definition:

Definition 4.1 (Operational functional)

The functional of the operational semantics,

$$\mathfrak{O}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma),$$

assigns to every statement $c \in \mathit{Cmd}$ a partial state transformation $\mathfrak{O}[\![c]\!] : \Sigma \dashrightarrow \Sigma$, which is defined as follows:

$$\mathfrak{O}[\![c]\!]\sigma := \begin{cases} \sigma' & \text{if } \langle c, \sigma \rangle \to \sigma' \text{ for some } \sigma' \in \Sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

Remark: $\mathfrak{O}[\![c]\!]\sigma$ can indeed be undefined
(consider e.g. $c = \texttt{while true do skip}$; see Corollary 3.4)

Equivalence of Statements

Underlying principle: two (syntactic) objects are considered (semantically) equivalent if they have the same "meaning"

- finite automata: $A_1 \sim A_2$ iff $L(A_1) = L(A_2)$
- context-free grammars: $G_1 \sim G_2$ iff $L(G_1) = L(G_2)$
- Turing machines: $T_1 \sim T_2$ iff both compute same function

Definition 4.2 (Operational equivalence)

Two statements $c_1, c_2 \in Cmd$ are called (operationally) equivalent (notation: $c_1 \sim c_2$) iff

$$\mathfrak{O}[\![c_1]\!] = \mathfrak{O}[\![c_2]\!].$$

Thus:

- $c_1 \sim c_2$ iff $\mathfrak{O}[\![c_1]\!]\sigma = \mathfrak{O}[\![c_2]\!]\sigma$ for every $\sigma \in \Sigma$
- In particular, $\mathfrak{O}[\![c_1]\!]\sigma$ is undefined iff $\mathfrak{O}[\![c_2]\!]\sigma$ is undefined

“Unwinding” of Loops

Simple application of statement equivalence: test of execution condition in a `while` loop can be represented by an `if` statement

Lemma 4.3

For every $b \in BExp$ *and* $c \in Cmd$,

$$\texttt{while}\ b\ \texttt{do}\ c \sim \texttt{if}\ b\ \texttt{then}\ (c\,;\texttt{while}\ b\ \texttt{do}\ c)\ \texttt{else}\ \texttt{skip}.$$

Proof.

on the board □

Outline

Summary: Operational Semantics

- Formalized by evaluation/execution relations
- Inductively defined by derivation trees using structural operational rules
- Enables proofs about operational behavior of programs using structural induction on derivation trees
- Semantic functional characterizes complete input/output behavior of programs

Outline

Denotational Semantics of WHILE

- Primary aspect of a program: its “effect”, i.e., input/output behavior
- In operational semantics: indirect definition of semantic functional $\mathfrak{O}[\![.]\!] : Cmd \to (\Sigma \dashrightarrow \Sigma)$ by execution relation
- Now: abstract from operational details
- Denotational semanics: direct definition of program effect by induction on its syntactic structure

Outline

Semantics of Arithmetic Expressions

Again: value of an expression determined by current state

Definition 4.4 (Denotational semantics of arithmetic expressions)

The (denotational) semantic functional for arithmetic expressions,

$$\mathfrak{A}[\![.]\!] : AExp \to (\Sigma \to \mathbb{Z}),$$

is given by:

$$\begin{aligned} \mathfrak{A}[\![z]\!]\sigma &:= z & \mathfrak{A}[\![a_1\texttt{+}a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma + \mathfrak{A}[\![a_2]\!]\sigma \\ \mathfrak{A}[\![x]\!]\sigma &:= \sigma(x) & \mathfrak{A}[\![a_1\texttt{-}a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma - \mathfrak{A}[\![a_2]\!]\sigma \\ & & \mathfrak{A}[\![a_1\texttt{*}a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma \cdot \mathfrak{A}[\![a_2]\!]\sigma \end{aligned}$$

Semantics of Boolean Expressions

Definition 4.5 (Denotational semantics of Boolean expressions)

The (denotational) semantic functional for Boolean expressions,

$$\mathfrak{B}[\![.]\!] : BExp \to (\Sigma \to \mathbb{B}),$$

is given by:

$$\mathfrak{B}[\![t]\!]\sigma := t$$

$$\mathfrak{B}[\![a_1\texttt{=}a_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma = \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![a_1\texttt{>}a_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![\neg b]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b]\!]\sigma = \text{false} \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![b_1 \wedge b_2]\!]\sigma := \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{true} \\ \text{false} & \text{otherwise} \end{cases}$$

$$\mathfrak{B}[\![b_1 \vee b_2]\!]\sigma := \begin{cases} \text{false} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{false} \\ \text{true} & \text{otherwise} \end{cases}$$