Semantics and Verification of Software

Lecture 5: Operational/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/svsw10/

Summer Semester 2010

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

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* $\sigma' = \sigma''$*.*

Proof.

To show:

$$\langle c, \sigma \rangle \to \sigma', \langle c, \sigma \rangle \to \sigma'' \implies \sigma' = \sigma''$$

(by structural induction on derivation trees) □

Functional of the Operational Semantics

The determinism of the execution relation (Theorem 3.4) justifies the following definition:

Definition (Operational functional)

The functional of the operational semantics,

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

assigns to every statement $c \in Cmd$ 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. $c =$ `while true do skip`; see Corollary 3.3)

Equivalence of Statements

Definition (Operational equivalence)

Two statements $c_1, c_2 \in \mathit{Cmd}$ are called (operationally) equivalent (notation: $c_1 \sim c_2$) if

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

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

Outline

“Unwinding” of Loops

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

Lemma 5.1

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

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

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 behaviour of programs using structural induction
- Semantic functional characterizes complete input/output behaviour of programs

Outline

Denotational Semantics of WHILE

- Primary aspect of a program: its "effect", i.e., input/output behaviour
- In operational semantics: indirect definition of semantic functional $\mathfrak{O}[\![.]\!]$ by execution relation
- Now: abstract from operational details
- Denotational semantics: 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 5.2 (Denotational semantics of arithmetic expressions)

The (denotational) semantic functional for arithmetic expressions,

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

is given by:

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

Semantics of Boolean Expressions

Definition 5.3 (Denotational semantics of Boolean expressions)

The (denotational) semantic functional for Boolean expressions,

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

is given by:

$$\begin{aligned}
\mathfrak{B}[\![t]\!]\sigma &:= t \\
\mathfrak{B}[\![a_1\texttt{=}a_2]\!]\sigma &:= \begin{cases} \mathsf{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma = \mathfrak{A}[\![a_2]\!]\sigma \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\mathfrak{B}[\![a_1\texttt{>}a_2]\!]\sigma &:= \begin{cases} \mathsf{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\mathfrak{B}[\![\neg b]\!]\sigma &:= \begin{cases} \mathsf{true} & \text{if } \mathfrak{B}[\![b]\!]\sigma = \mathsf{false} \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\mathfrak{B}[\![b_1 \wedge b_2]\!]\sigma &:= \begin{cases} \mathsf{true} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \mathsf{true} \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\mathfrak{B}[\![b_1 \vee b_2]\!]\sigma &:= \begin{cases} \mathsf{false} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \mathsf{false} \\ \mathsf{true} & \text{otherwise} \end{cases}
\end{aligned}$$

Outline

Semantics of Statements I

- Now: semantic functional
$$\mathfrak{C}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma)$$
- Same type as operational functional
$$\mathfrak{O}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma)$$
(in fact, both will turn out to be the same
$\Longrightarrow$ equivalence of operational and denotational semantics)
- Inductive definition employs auxiliary functions:
  - identity on states: $\mathsf{id}_\Sigma : \Sigma \dashrightarrow \Sigma : \sigma \mapsto \sigma$
  - (strict) composition of partial state transformations:
$$\circ : (\Sigma \dashrightarrow \Sigma) \times (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma)$$
    where, for every $f, g : \Sigma \dashrightarrow \Sigma$ and $\sigma \in \Sigma$,
$$(g \circ f)(\sigma) := \begin{cases} g(f(\sigma)) & \text{if } f(\sigma) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$
  - semantic conditional:
$$\mathsf{cond} : (\Sigma \to \mathbb{B}) \times (\Sigma \dashrightarrow \Sigma) \times (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma)$$
    where, for every $p : \Sigma \to \mathbb{B}$, $f, g : \Sigma \dashrightarrow \Sigma$, and $\sigma \in \Sigma$,
$$\mathsf{cond}(p, f, g)(\sigma) := \begin{cases} f(\sigma) & \text{if } p(\sigma) = \mathsf{true} \\ g(\sigma) & \text{otherwise} \end{cases}$$

Semantics of Statements II

Definition 5.4 (Denotational semantics of statements)

The (denotational) semantic functional for statements,

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

is given by:

$$\begin{aligned}
\mathfrak{C}[\![\texttt{skip}]\!] &:= \mathsf{id}_\Sigma \\
\mathfrak{C}[\![x \texttt{ := } a]\!]\sigma &:= \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \\
\mathfrak{C}[\![c_1 \texttt{;} c_2]\!] &:= \mathfrak{C}[\![c_2]\!] \circ \mathfrak{C}[\![c_1]\!] \\
\mathfrak{C}[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!] &:= \mathsf{cond}(\mathfrak{B}[\![b]\!], \mathfrak{C}[\![c_1]\!], \mathfrak{C}[\![c_2]\!]) \\
\mathfrak{C}[\![\texttt{while } b \texttt{ do } c]\!] &:= \mathsf{fix}(\Phi)
\end{aligned}$$

where $\Phi : (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma) : f \mapsto \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$

Semantics of Statements III

Remarks:

- Definition of $\mathfrak{C}[\![c]\!]$ given by induction on syntactic structure of $c \in Cmd$
  - in particular, $\mathfrak{C}[\![\texttt{while}\ b\ \texttt{do}\ c]\!]$ only refers to $\mathfrak{B}[\![b]\!]$ and $\mathfrak{C}[\![c]\!]$ (and not to $\mathfrak{C}[\![\texttt{while}\ b\ \texttt{do}\ c]\!]$ again)
  - note difference to $\mathfrak{O}[\![c]\!]$:

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

- In $\mathfrak{C}[\![c_1 ; c_2]\!] := \mathfrak{C}[\![c_2]\!] \circ \mathfrak{C}[\![c_1]\!]$, function composition $\circ$ has to be strict since non-termination of c_1 implies non-termination of $c_1 ; c_2$ (i.e., $\mathfrak{C}[\![c_1]\!]\sigma = \text{undefined} \implies \mathfrak{C}[\![c_1 ; c_2]\!]\sigma = \text{undefined}$)
- In $\mathfrak{C}[\![\texttt{while}\ b\ \texttt{do}\ c]\!] := \mathsf{fix}(\Phi)$, fix denotes a fixpoint operator (which remains to be defined)
  $\implies$ "fixpoint semantics"

But: why fixpoints?