

Semantics and Verification of Software

Lecture 12: Operational Semantics of Blocks and Procedures

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw/>

Summer semester 2007

- 1 Repetition: Total Correctness
- 2 Soundness and Completeness
- 3 Summary: Axiomatic Semantics
- 4 Semantics of Blocks and Procedures
- 5 Operational Semantics

Proving Total Correctness

Goal: syntactic derivation of valid total correctness properties

Definition (Hoare Logic for total correctness)

The **Hoare rules** for total correctness are given by

$$\begin{array}{c} \frac{}{\{A\} \text{skip} \{\Downarrow A\}} \text{ (skip)} \qquad \frac{}{\{A[x \mapsto a]\} x := a \{\Downarrow A\}} \text{ (asgn)} \\[10pt] \frac{\{A\} c_1 \{\Downarrow C\} \quad \{C\} c_2 \{\Downarrow B\}}{\{A\} c_1; c_2 \{\Downarrow B\}} \text{ (seq)} \qquad \frac{\{A \wedge b\} c_1 \{\Downarrow B\} \quad \{A \wedge \neg b\} c_2 \{\Downarrow B\}}{\{A\} \text{if } b \text{ then } c_1 \text{ else } c_2 \{\Downarrow B\}} \text{ (if)} \\[10pt] \frac{\{i \geq 0 \wedge A(i+1)\} c \{\Downarrow A(i)\}}{\{\exists i. i \geq 0 \wedge A(i)\} \text{while } b \text{ do } c \{\Downarrow A(0)\}} \text{ (while)} \\[10pt] \frac{\models (A \implies A') \quad \{A'\} c \{\Downarrow B'\} \quad \models (B' \implies B)}{\{A\} c \{\Downarrow B\}} \text{ (cons)} \end{array}$$

where $i \in LVar$, $\models (i \geq 0 \wedge A(i+1) \implies b)$, and $\models (A(0) \implies \neg b)$.

A total correctness property is **provable** (notation: $\vdash \{A\} c \{\Downarrow B\}$) if it is derivable by the Hoare rules. In case of (while), $A(i)$ is called a **(loop) invariant**.

- 1 Repetition: Total Correctness
- 2 Soundness and Completeness
- 3 Summary: Axiomatic Semantics
- 4 Semantics of Blocks and Procedures
- 5 Operational Semantics

In analogy to Theorem 9.5 we can show that the Hoare Logic for total correctness properties is also sound:

Theorem 12.1 (Soundness)

For every total correctness property $\{A\} c \{\Downarrow B\}$,

$$\vdash \{A\} c \{\Downarrow B\} \implies \models \{A\} c \{\Downarrow B\}.$$

Proof.

again by structural induction over the derivation tree of $\vdash \{A\} c \{\Downarrow B\}$
(only (while) case; on the board) □

Relative Completeness

Also the counterpart to Cook's Completeness Theorem 10.3 applies:

Theorem 12.2 (Completeness)

*The Hoare Logic for total correctness properties is **relatively complete**, i.e., for every $\{A\} c \{\Downarrow B\}$:*

$$\models \{A\} c \{\Downarrow B\} \implies \vdash \{A\} c \{\Downarrow B\}.$$

Proof.

omitted



- 1 Repetition: Total Correctness
- 2 Soundness and Completeness
- 3 Summary: Axiomatic Semantics**
- 4 Semantics of Blocks and Procedures
- 5 Operational Semantics

Summary: Axiomatic Semantics

- Formalized by **partial/total correctness properties**
- Inductively defined by **Hoare Logic** proof rules
- Technically involved (especially loop invariants)
 $\implies$ machine support (**proof assistants**) indispensable for larger programs
- **Equivalence** of axiomatic and operational/denotational semantics
- **Software engineering** aspect: integrated development of program and proof (cf. assertions in Java)

- 1 Repetition: Total Correctness
- 2 Soundness and Completeness
- 3 Summary: Axiomatic Semantics
- 4 Semantics of Blocks and Procedures**
- 5 Operational Semantics

- Extension of WHILE by **blocks** with (local) **variables** and (recursive) **procedures**
- Involves new semantic concepts:
 - variable und procedure **environments**
 - **locations** (memory addresses) and **stores** (memory states)
- Important: **scope** of variable and procedure identifiers
 - static scoping**: scope of identifier = **declaration environment** (here)
 - dynamic scoping**: scope of identifier = **calling environment**
(old Algol/Lisp dialects)

Example 12.3

```
begin
  var x;
  proc P is y := x;
  x := 1;
  begin
    var x;
    x := 2;
    call P
  end
end
```

static scoping $\implies y = 1$

dynamic scoping $\implies y = 2$

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$PVar = \{P, Q, \dots\}$	P
Procedure declarations	$PDec$	p
Variable declarations	$VDec$	v
Commands (statements)	Cmd	c

Context-free grammar:

$p ::= \text{proc } P \text{ is } c; p \mid \varepsilon \in PDec$

$v ::= \text{var } x; v \mid \varepsilon \in VDec$

$c ::= \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \mid$
 $\text{call } P \mid \text{begin } v \text{ } p \text{ } c \text{ end} \in Cmd$

- 1 Repetition: Total Correctness
- 2 Soundness and Completeness
- 3 Summary: Axiomatic Semantics
- 4 Semantics of Blocks and Procedures
- 5 Operational Semantics

Caveat: see Lecture 14 for corrections!

- So far: states $\Sigma = \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\}$
- Now: explicit control over all (nested) **instances** of a variable:
 - **variable environments** $VEnv := \{\rho \mid \rho : Var \rightarrow Loc\}$
 - **(memory) locations** $Loc := \mathbb{N}$
 - **stores** $Sto := \{\sigma \mid \sigma : Loc \rightarrow \mathbb{Z}\}$

$\Rightarrow$ **Two-level access** to a variable $x \in Var$:

- ① determine current memory location of x : $l := \rho(x)$
- ② reading/writing access to σ at position l
- **Effect of procedure call** determined by its body statement and variable and procedure environment of its declaration:

$$PEnv := \{\pi \mid \pi : PVar \rightarrow Cmd \times VEnv \times PEnv\}$$

- **Effect of declaration:** update of environment

$$\text{upd}_v : VDec \times VEnv \rightarrow VEnv$$

$$\text{upd}_v(\text{var } x; v, \rho) := \text{upd}_v(v, \rho[x \mapsto \min(Lab \setminus \rho(Var))])$$

$$\text{upd}_v(\varepsilon, \rho) := \rho$$

$$\text{upd}_p : PDec \times VEnv \times PEnv \rightarrow PEnv$$

$$\text{upd}_p(\text{proc } P \text{ is } c; p, \rho, \pi) := \text{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$

$$\text{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Definition 12.4 (Execution relation)

For $c \in \text{Cmd}$, $\sigma, \sigma' \in \text{Sto}$, $\rho \in \text{VEnv}$, and $\pi \in \text{PEnv}$, the **execution relation** $\rho, \pi \vdash \langle c, \sigma \rangle \rightarrow \sigma'$ is defined by the following rules:

$$\begin{array}{c}
 \frac{}{\rho, \pi \vdash \langle \text{skip}, \sigma \rangle \rightarrow \sigma} \text{ (skip)} \quad \frac{\langle a, \sigma \circ \rho \rangle \rightarrow z}{\rho, \pi \vdash \langle x := a, \sigma \rangle \rightarrow \sigma[\rho(x) \mapsto z]} \text{ (asgn)} \\
 \frac{\rho, \pi \vdash \langle c_1, \sigma \rangle \rightarrow \sigma' \quad \rho, \pi \vdash \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\rho, \pi \vdash \langle c_1; c_2, \sigma \rangle \rightarrow \sigma''} \text{ (seq)} \\
 \frac{\rho, \pi \vdash \langle c_1; c_2, \sigma \rangle \rightarrow \sigma''}{\langle b, \sigma \circ \rho \rangle \rightarrow \text{true} \quad \rho, \pi \vdash \langle c_1, \sigma \rangle \rightarrow \sigma'} \text{ (if-t)} \\
 \frac{\rho, \pi \vdash \langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma' \quad \langle b, \sigma \circ \rho \rangle \rightarrow \text{false}}{\rho, \pi \vdash \langle c_2, \sigma \rangle \rightarrow \sigma'} \text{ (if-f)} \\
 \frac{\rho, \pi \vdash \langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma' \quad \langle b, \sigma \circ \rho \rangle \rightarrow \text{false}}{\rho, \pi \vdash \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma} \text{ (wh-f)} \\
 \frac{\rho, \pi \vdash \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma' \quad \rho, \pi \vdash \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle b, \sigma \circ \rho \rangle \rightarrow \text{true} \quad \rho, \pi \vdash \langle c, \sigma \rangle \rightarrow \sigma' \quad \rho, \pi \vdash \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''} \text{ (wh-t)} \\
 \frac{\rho, \pi \vdash \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''}{\rho', \pi'[P \mapsto (c, \rho', \pi')] \vdash \langle c, \sigma \rangle \rightarrow \sigma'} \text{ (call)} \quad \text{if } \pi(P) = (c, \rho', \pi') \\
 \frac{\rho, \pi \vdash \langle \text{call } P, \sigma \rangle \rightarrow \sigma' \quad \text{upd}_v(v, \rho), \text{upd}_p(p, \text{upd}_v(v, \rho), \pi) \vdash \langle c, \sigma \rangle \rightarrow \sigma'}{\rho, \pi \vdash \langle \text{begin } v \text{ p } c \text{ end}, \sigma \rangle \rightarrow \sigma'} \text{ (block)}
 \end{array}$$

Remarks about rule (call):

- The procedure environment associated with procedure P is extended by a P -entry to handle **recursive calls** of P :
 $\pi'[P \mapsto (c, \rho', \pi')]$
- **Static scoping** is modelled by using the environments ρ' and π' from the declaration site of procedure P (and not ρ and π from the calling site)

Example 12.5

```

c = begin
  var x; var y; } v
  proc F is
    begin
      var z;
      z := x;
      if z=1 then skip
        else x := x-1;
          call F;
          y := z * y } c2
      end
    } c1
  } cF
x := 2; y := 1; call F } c0
end

```

Diagram illustrating the execution relation for the code snippet above. The code is structured with nested blocks, each associated with a label in curly braces:

- The outermost block is labeled c_0 .
- The block containing `var x; var y;` is labeled v .
- The block containing the `proc F` definition is labeled c_F .
- The block containing the `begin` of `proc F` is labeled c_1 .
- The block containing the `if` statement and its branches is labeled c_2 .

Let $\rho_\emptyset(x) = \perp = \pi_\emptyset(P)$ for all $x \in Var$ and $P \in PVar$, and let $\sigma \in Sto$.

Notation: $\sigma_{ijkl} \Leftrightarrow \sigma(0) = i, \sigma(1) = j, \sigma(2) = k, \sigma(3) = l$

Derivation tree for $\rho_\emptyset, \emptyset \vdash \langle c, \sigma \rangle \rightarrow \sigma_{1221}$: on the board