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

Outline

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

$$\frac{}{\{A\}\,\texttt{skip}\,\{\Downarrow A\}}\ (\text{skip}) \qquad \frac{}{\{A[x \mapsto a]\}\,x\ \texttt{:=}\ a\,\{\Downarrow A\}}\ (\text{asgn})$$

$$\frac{\{A\}\,c_1\,\{\Downarrow C\}\ \ \{C\}\,c_2\,\{\Downarrow B\}}{\{A\}\,c_1\,;c_2\,\{\Downarrow B\}}\ (\text{seq}) \qquad \frac{\{A \wedge b\}\,c_1\,\{\Downarrow B\}\ \ \{A \wedge \neg b\}\,c_2\,\{\Downarrow B\}}{\{A\}\,\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\,\{\Downarrow B\}}\ (\text{if})$$

$$\frac{\{i \geq 0 \wedge A(i+1)\}\,c\,\{\Downarrow A(i)\}}{\{\exists i.\, i \geq 0 \wedge A(i)\}\,\texttt{while}\ b\ \texttt{do}\ c\,\{\Downarrow A(0)\}}\ (\text{while})$$

$$\frac{\models (A \implies A')\ \ \{A'\}\,c\,\{\Downarrow B'\}\ \ \models (B' \implies B)}{\{A\}\,c\,\{\Downarrow B\}}\ (\text{cons})$$

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.

Outline

Soundness

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) □

Soundness

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 □

Outline

Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties
- Inductively defined by Hoare Logic proof rules
- Technically involved (especially loop invariants)
  $\Longrightarrow$ 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)

Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties
- Inductively defined by Hoare Logic proof rules
- Technically involved (especially loop invariants)
  $\Longrightarrow$ 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)

Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties
- Inductively defined by Hoare Logic proof rules
- Technically involved (especially loop invariants)
  $\Longrightarrow$ 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)

Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties
- Inductively defined by Hoare Logic proof rules
- Technically involved (especially loop invariants)
  $\Longrightarrow$ 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)

Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties
- Inductively defined by Hoare Logic proof rules
- Technically involved (especially loop invariants)
  $\Longrightarrow$ 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)

Outline

Blocks and Procedures

- 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)

Blocks and Procedures

- 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)

Blocks and Procedures

- 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)

Static and Dynamic Scoping

Example 12.3

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

Static and Dynamic Scoping

Example 12.3

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

static scoping $\Longrightarrow$ y = 1

Static and Dynamic Scoping

Example 12.3

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

dynamic scoping $\Longrightarrow$ y = 2

Extending the Syntax

Syntactic categories:

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

Context–free grammar:

$$p ::= \mathtt{proc}\ P\ \mathtt{is}\ c\,;p \mid \varepsilon \in PDec$$
$$v ::= \mathtt{var}\ x\,;v \mid \varepsilon \in VDec$$
$$c ::= \mathtt{skip} \mid x\ \mathtt{:=}\ a \mid c_1\,;c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c \mid \mathtt{call}\ P \mid \mathtt{begin}\ v\ p\ c\ \mathtt{end} \in Cmd$$

Extending the Syntax

Syntactic categories:

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

Context–free grammar:

$$p ::= \texttt{proc}\ P\ \texttt{is}\ c\texttt{;}p \mid \varepsilon \in PDec$$
$$v ::= \texttt{var}\ x\texttt{;}v \mid \varepsilon \in VDec$$
$$c ::= \texttt{skip} \mid x\ \texttt{:=}\ a \mid c_1\texttt{;}c_2 \mid \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ b\ \texttt{do}\ c \mid \texttt{call}\ P \mid \texttt{begin}\ v\ p\ c\ \texttt{end} \in Cmd$$

Outline

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ 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:

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

- Effect of declaration: update of environment

$$\mathsf{upd}_v : VDec \times VEnv \to VEnv$$
$$\mathsf{upd}_v(\texttt{var}\ x\,;v, \rho) := \mathsf{upd}_v(v, \rho[x \mapsto \min(Lab \setminus \rho(Var))])$$
$$\mathsf{upd}_v(\varepsilon, \rho) := \rho$$
$$\mathsf{upd}_p : PDec \times VEnv \times PEnv \to PEnv$$
$$\mathsf{upd}_p(\texttt{proc}\ P\ \texttt{is}\ c\,;p, \rho, \pi) := \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ 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:

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

- Effect of declaration: update of environment

$$\mathsf{upd}_v : VDec \times VEnv \to VEnv$$
$$\mathsf{upd}_v(\mathtt{var}\ x\,;v, \rho) := \mathsf{upd}_v(v, \rho[x \mapsto \mathsf{min}(Lab \setminus \rho(Var))])$$
$$\mathsf{upd}_v(\varepsilon, \rho) := \rho$$
$$\mathsf{upd}_p : PDec \times VEnv \times PEnv \to PEnv$$
$$\mathsf{upd}_p(\mathtt{proc}\ P\ \mathtt{is}\ c\,;p, \rho, \pi) := \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ Two–level access to a variable $x \in \mathit{Var}$:

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:

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

- Effect of declaration: update of environment

$$\mathsf{upd}_v : \mathit{VDec} \times \mathit{VEnv} \to \mathit{VEnv}$$
$$\mathsf{upd}_v(\mathtt{var}\ x\,;v, \rho) := \mathsf{upd}_v(v, \rho[x \mapsto \min(\mathit{Lab} \setminus \rho(\mathit{Var}))])$$
$$\mathsf{upd}_v(\varepsilon, \rho) := \rho$$
$$\mathsf{upd}_p : \mathit{PDec} \times \mathit{VEnv} \times \mathit{PEnv} \to \mathit{PEnv}$$
$$\mathsf{upd}_p(\mathtt{proc}\ P\ \mathtt{is}\ c\,;p, \rho, \pi) := \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ Two–level access to a variable $x \in \mathit{Var}$:

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:

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

- Effect of declaration: update of environment

$$\mathsf{upd}_v : \mathit{VDec} \times \mathit{VEnv} \to \mathit{VEnv}$$
$$\mathsf{upd}_v(\mathtt{var}\ x\,;v, \rho) := \mathsf{upd}_v(v, \rho[x \mapsto \min(\mathit{Lab} \setminus \rho(\mathit{Var}))])$$
$$\mathsf{upd}_v(\varepsilon, \rho) := \rho$$
$$\mathsf{upd}_p : \mathit{PDec} \times \mathit{VEnv} \times \mathit{PEnv} \to \mathit{PEnv}$$
$$\mathsf{upd}_p(\mathtt{proc}\ P\ \mathtt{is}\ c\,;p, \rho, \pi) := \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ Two–level access to a variable $x \in \mathit{Var}$:

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:

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

- Effect of declaration: update of environment

$$\mathsf{upd}_v : \mathit{VDec} \times \mathit{VEnv} \to \mathit{VEnv}$$
$$\mathsf{upd}_v(\mathtt{var}\ x\,; v, \rho) := \mathsf{upd}_v(v, \rho[x \mapsto \min(\mathit{Lab} \setminus \rho(\mathit{Var}))])$$
$$\mathsf{upd}_v(\varepsilon, \rho) := \rho$$
$$\mathsf{upd}_p : \mathit{PDec} \times \mathit{VEnv} \times \mathit{PEnv} \to \mathit{PEnv}$$
$$\mathsf{upd}_p(\mathtt{proc}\ P\ \mathtt{is}\ c\,; p, \rho, \pi) := \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p(\varepsilon, \rho, \pi) := \pi$$

Operational Semantics

Caveat: see Lecture 14 for corrections!

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

$\Longrightarrow$ Two–level access to a variable $x \in \mathit{Var}$:

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:

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

- Effect of declaration: update of environment

$$\begin{aligned}
\mathsf{upd}_v &: \mathit{VDec} \times \mathit{VEnv} \to \mathit{VEnv} \\
\mathsf{upd}_v(\mathtt{var}\ x\,;v, \rho) &:= \mathsf{upd}_v(v, \rho[x \mapsto \mathsf{min}(\mathit{Lab} \setminus \rho(\mathit{Var}))]) \\
\mathsf{upd}_v(\varepsilon, \rho) &:= \rho \\
\mathsf{upd}_p &: \mathit{PDec} \times \mathit{VEnv} \times \mathit{PEnv} \to \mathit{PEnv} \\
\mathsf{upd}_p(\mathtt{proc}\ P\ \mathtt{is}\ c\,;p, \rho, \pi) &:= \mathsf{upd}_p(p, \rho, \pi[P \mapsto (c, \rho, \pi)]) \\
\mathsf{upd}_p(\varepsilon, \rho, \pi) &:= \pi
\end{aligned}$$

Execution Relation I

Definition 12.4 (Execution relation)

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

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

$$\frac{\rho, \pi \vdash \langle c_1, \sigma \rangle \to \sigma' \quad \rho, \pi \vdash \langle c_2, \sigma' \rangle \to \sigma''}{\rho, \pi \vdash \langle c_1 \texttt{;} c_2, \sigma \rangle \to \sigma''} \text{(seq)}$$

$$\frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{true} \quad \rho, \pi \vdash \langle c_1, \sigma \rangle \to \sigma'}{\rho, \pi \vdash \langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'} \text{(if–t)}$$

$$\frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{false} \quad \rho, \pi \vdash \langle c_2, \sigma \rangle \to \sigma'}{\rho, \pi \vdash \langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \rangle \to \sigma'} \text{(if–f)}$$

$$\frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{false}}{\rho, \pi \vdash \langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \sigma} \text{(wh–f)}$$

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

$$\frac{\rho', \pi'[P \mapsto (c, \rho', \pi')] \vdash \langle c, \sigma \rangle \to \sigma'}{\rho, \pi \vdash \langle \texttt{call } P, \sigma \rangle \to \sigma'} \text{(call)} \qquad \text{if } \pi(P) = (c, \rho', \pi')$$

$$\frac{\mathsf{upd}_v(v, \rho), \mathsf{upd}_p(p, \mathsf{upd}_v(v, \rho), \pi) \vdash \langle c, \sigma \rangle \to \sigma'}{\rho, \pi \vdash \langle \texttt{begin } v\ p\ c \texttt{ end}, \sigma \rangle \to \sigma'} \text{(block)}$$

Execution Relation II

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)

Execution Relation III

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;          } c2
                    y := z * y       } c1
      end                                    } cF
    x := 2; y := 1; call F           } c0   } p
end
```

Let $\rho_\emptyset(x) = \bot = \pi_\emptyset(P)$ 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 $\rho_\emptyset, \emptyset \vdash \langle c, \sigma \rangle \to \sigma_{1221}$: on the board