

# Semantics and Verification of Software

## Lecture 15: Semantics of Blocks and Procedures

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

- 1 Summary: Axiomatic Semantics
- 2 Extension by Blocks and Procedures
- 3 Operational Semantics of Blocks and Procedures

# Summary: Axiomatic Semantics

- Formalized by partial/total correctness properties

# Summary: Axiomatic Semantics

- Formalized by **partial/total correctness properties**
- Inductively defined by **Hoare Logic** proof rules

# 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

# 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

# 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 Summary: Axiomatic Semantics
- 2 Extension by Blocks and Procedures
- 3 Operational Semantics of Blocks and Procedures



- Extension of WHILE by **blocks** with **(local) variables** and **(recursive) procedures**

# 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)

- 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 15.1

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

## Example 15.1

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

static scoping  $\Rightarrow y = 1$

## Example 15.1

```
begin
  var x; var y;
  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)	$Cmd$	$c$

## 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 Summary: Axiomatic Semantics
- 2 Extension by Blocks and Procedures
- 3 Operational Semantics of Blocks and Procedures

- So far: **states**  $\Sigma = \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\}$

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

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

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

- 1 determine current memory location of  $x$ :

$$l := \rho(x)$$

- 2 reading/writing access to  $\sigma$  at position  $l$

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

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

- 1 determine current memory location of  $x$ :

$$l := \rho(x)$$

- 2 reading/writing access to  $\sigma$  at position  $l$

- Thus: previous **state** information represented as  $\sigma \circ \rho$

- **Effect of procedure call** determined by its body statement and variable and procedure environment of its declaration:

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

denotes the set of **procedure environments**

- **Effect of procedure call** determined by its body statement and variable and procedure environment of its declaration:

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

denotes the set of **procedure environments**

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

$$\text{upd}_v[\![\cdot]\!] : VDec \times VEnv \times Sto \rightarrow VEnv \times Sto$$

$$\text{upd}_v[\![\text{var } x; v]\!](\rho, \sigma) := \text{upd}_v[\![v]\!](\rho[x \mapsto l_x], \sigma[l_x \mapsto 0])$$

$$\text{upd}_v[\![\varepsilon]\!](\rho, \sigma) := (\rho, \sigma)$$

$$\text{upd}_p[\![\cdot]\!] : PDec \times VEnv \times PEnv \rightarrow PEnv$$

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

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

where  $l_x := \min\{l \in \mathbb{N} \mid \sigma(l) = \perp\}$

## Definition 15.2 (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:

$$\text{(skip)} \frac{}{(\rho, \pi) \vdash \langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

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

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

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

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



## Definition 15.2 (Execution relation; continued)

$$(\text{wh-f}) \frac{\langle b, \sigma \circ \rho \rangle \rightarrow \text{false}}{(\rho, \pi) \vdash \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$$

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

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

$$(\text{block}) \frac{\text{upd}_v \llbracket v \rrbracket (\rho, \sigma) = (\rho', \sigma') \quad (\rho', \text{upd}_p \llbracket p \rrbracket (\rho', \pi)) \vdash \langle c, \sigma' \rangle \rightarrow \sigma''}{(\rho, \pi) \vdash \langle \text{begin } v \text{ } p \text{ } c \text{ end}, \sigma \rangle \rightarrow \sigma''}$$

**Remarks** about rules (call) and (block):

- **Static scoping** is modelled in (call) by using the environments  $\rho'$  and  $\pi'$  (as determined in (block)) from the declaration site of procedure  $P$  (and not  $\rho$  and  $\pi$  from the calling site)
- In (call), the procedure environment associated with procedure  $P$  is extended by a  $P$ -entry to handle **recursive calls** of  $P$ :

$$\pi'[P \mapsto (c, \rho', \pi')]$$

## Example 15.3

```

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
    end
  x := 2; y := 1; call F } c0
end

```

Diagrammatic annotations for the execution relation:

- A large right curly brace  $\}$  groups the entire procedure `F` and is labeled  $c_1$ .
- A right curly brace  $\}$  groups the `begin` block of `F` and is labeled  $c_F$ .
- A right curly brace  $\}$  groups the `if` statement and the `call F` statement, and is labeled  $p$ .

Let  $\sigma_\emptyset(l) = \rho_\emptyset(x) = \pi_\emptyset(P) = \perp$  for all  $l \in Loc, x \in Var, P \in PVar$

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

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