



# Semantics and Verification of Software

Summer Semester 2019

Lecture 14: Extension by Blocks and Procedures I (Operational Semantics)

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-19/sv-sw/>

# Extension by Blocks and Procedures

---

## Blocks and Procedures

- Extension of WHILE by nested **blocks** with local **variables** and recursive **procedures**
  - Simple memory model ( $\Sigma := \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\}$ ) not sufficient any more as variables can occur in several **instances**
- ⇒ Involves new semantic concepts:
- variable and procedure **environments**
  - **locations** (memory addresses) and **stores** (memory states)
- Important: **scope** of variable and procedure identifiers
- static scoping**: scope of identifier = **declaration environment**  
(also: “lexical” scoping; here)
- dynamic scoping**: scope of identifier = **calling environment**  
(old Algol/Lisp dialects)

# Extension by Blocks and Procedures

---

## Static and Dynamic Scoping

### Example 14.1

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

static scoping  $\Rightarrow y = 1$

dynamic scoping  $\Rightarrow y = 2$

# Extending the Syntax

---

## Extending the Syntax

### 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:

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

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

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

- All used variable/procedure identifiers have to be declared
- Identifiers declared within a block must be distinct

# New Semantic Domains

---

## Locations and Stores

- So far: **states**  $\Sigma = \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\}$
- Now: explicit control over all (nested) **instances** of a variable:
  - **variable environments**

$$VEnv := \{\rho \mid \rho : Var \dashrightarrow Loc\}$$

(partial function to maintain **declaredness** information)

- **locations**

$$Loc := \mathbb{N}$$

- **stores**

$$Sto := \{\sigma \mid \sigma : Loc \dashrightarrow \mathbb{Z}\}$$

(partial function to maintain **allocation** information)

⇒ **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 location  $l$

- Thus: previous **state** information represented as  $\sigma \circ \rho : Var \dashrightarrow \mathbb{Z}$

# New Semantic Domains

---

## Procedure Environments and Declarations

- **Effect of procedure call** determined by its body 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 (and store)

$$upd_v[\cdot] : VDec \times VEnv \times Sto \rightarrow VEnv \times Sto$$

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

$$upd_v[\varepsilon](\rho, \sigma) := (\rho, \sigma)$$

$$upd_p[\cdot] : PDec \times VEnv \times PEnv \rightarrow PEnv$$

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

$$upd_p[\varepsilon](\rho, \pi) := \pi$$

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

# The Execution Relation

## Execution Relation I

### Definition 14.2 (Execution relation)

For  $c \in \mathit{Cmd}$ ,  $\sigma, \sigma' \in \mathit{Sto}$ ,  $\rho \in \mathit{VEnv}$ , and  $\pi \in \mathit{PEnv}$ , the **execution relation**  $(\rho, \pi) \vdash \langle c, \sigma \rangle \rightarrow \sigma'$  (“in environment  $(\rho, \pi)$ , statement  $c$  transforms store  $\sigma$  into  $\sigma'$ ”) is defined by the following rules:

$$\begin{array}{c} \text{(skip)} \\ \hline (\rho, \pi) \vdash \langle \text{skip}, \sigma \rangle \rightarrow \sigma \\ \\ \langle a, \sigma \circ \rho \rangle \rightarrow z \\ \text{(asgn)} \\ \hline (\rho, \pi) \vdash \langle x := a, \sigma \rangle \rightarrow \sigma[\rho(x) \mapsto z] \\ \\ (\rho, \pi) \vdash \langle c_1, \sigma \rangle \rightarrow \sigma' \quad (\rho, \pi) \vdash \langle c_2, \sigma' \rangle \rightarrow \sigma'' \\ \text{(seq)} \\ \hline (\rho, \pi) \vdash \langle c_1 ; c_2, \sigma \rangle \rightarrow \sigma'' \\ \\ \langle b, \sigma \circ \rho \rangle \rightarrow \text{true} \quad (\rho, \pi) \vdash \langle c_1, \sigma \rangle \rightarrow \sigma' \\ \text{(if-t)} \\ \hline (\rho, \pi) \vdash \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow \sigma' \end{array}$$

# The Execution Relation

## Execution Relation II

### Definition 14.2 (Execution relation; continued)

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

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

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

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

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



# The Execution Relation

---

## Execution Relation III

### Remarks:

- Evaluation of (arithmetic and Boolean) expressions can now **fail** due to undeclared variables
- In rules for composite statements, the execution of sub-statements can have an effect on the environments (due to nested environments), but this effect is **transient**
- **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  $P$ ,  $\pi'$ , is extended by a  $P$ -entry to handle **recursive calls** of  $P$ :

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

# The Execution Relation

## Execution Relation IV

### Example 14.3

```
begin
  var x; var y;
  proc F is
    begin
      var z;
      z:=x;
      if z=1 then skip
        else x:=x-1; call F; y:=z*y end
    end
  end;
  x:=2; y:=1; call F
end
```

$\} v$   
 $\} c_1$   
 $\} c_F$   
 $\} p$   
 $\} c$   
 $\} c_0$

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