



Semantics and Verification of Software

Summer Semester 2019

Lecture 15: Extension by Blocks and Procedures II (Denotational Semantics)

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

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

Recap: Operational Semantics of Blocks and Procedures

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Recap: Operational Semantics of Blocks and Procedures

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

Recap: Operational Semantics of Blocks and Procedures

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 σ at location l

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

Recap: Operational Semantics of Blocks and Procedures

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

Recap: Operational Semantics of Blocks and Procedures

Execution Relation I

Definition (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 (ρ, π) , statement c transforms store σ into σ' ”) is defined by the following rules:

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

Recap: Operational Semantics of Blocks and Procedures

Execution Relation II

Definition (Execution relation; continued)

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

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

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

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

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

Denotational Semantics of Blocks and Procedures

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

The Approach

Operational semantics: “syntactic” approach

- procedure environment holds code of procedure body
- semantics of call = “inlining”

Denotational Semantics of Blocks and Procedures

The Approach

Operational semantics: “syntactic” approach

- procedure environment holds code of procedure body
- semantics of call = “inlining”

Denotational semantics: “semantic” approach

- procedure environment holds (partial) storage transformations
- semantics of call = function application
- variables handled as in operational semantics (by environment and stores)
- declarations of recursive procedures handled by fixpoint approach

Handling Variable Declarations

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Handling Variable Declarations

Handling Variable Declarations

Exactly as in operational semantics:

- **Variable environments** keep location information:

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

with $Loc := \mathbb{N}$

Handling Variable Declarations

Handling Variable Declarations

Exactly as in operational semantics:

- **Variable environments** keep location information:

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

with $Loc := \mathbb{N}$

- **Effect of variable declaration**: update of environment and store

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

$$\begin{aligned} \text{upd}_v[\text{var } x; v](\rho, \sigma) &:= \text{upd}_v[v](\rho[x \mapsto l_x], \sigma[l_x \mapsto 0]) \\ \text{upd}_v[\varepsilon](\rho, \sigma) &:= (\rho, \sigma) \end{aligned}$$

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

Handling Variable Declarations

Statement Semantics Using Variable Environments

- **First step:** reformulation of Definition 6.3 using **variable environments and locations** (initially disregarding procedures)
- **So far:** $\mathcal{C}[\cdot] : Cmd \rightarrow (\Sigma \dashrightarrow \Sigma)$

Handling Variable Declarations

Statement Semantics Using Variable Environments

- **First step:** reformulation of Definition 6.3 using **variable environments and locations** (initially disregarding procedures)
- **So far:** $\mathcal{E}[\cdot] : Cmd \rightarrow (\Sigma \dashrightarrow \Sigma)$

Definition 15.1 (Denotational semantics using locations)

The **(denotational) semantic functional for statements**,

$$\mathcal{E}'[\cdot] : Cmd \rightarrow VEnv \rightarrow (Sto \dashrightarrow Sto),$$

is given by:

$$\begin{aligned}\mathcal{E}'[\text{skip}]\rho &:= \text{id}_{Sto} \\ \mathcal{E}'[x := a]\rho \sigma &:= \sigma[\rho(x) \mapsto \mathcal{A}[a](\text{lookup } \rho \sigma)] \\ \mathcal{E}'[c_1; c_2]\rho &:= (\mathcal{E}'[c_2]\rho) \circ (\mathcal{E}'[c_1]\rho) \\ \mathcal{E}'[\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}]\rho &:= \text{cond}(\mathcal{B}[b] \circ (\text{lookup } \rho), \mathcal{E}'[c_1]\rho, \mathcal{E}'[c_2]\rho) \\ \mathcal{E}'[\text{while } b \text{ do } c \text{ end}]\rho &:= \text{fix}(\Phi)\end{aligned}$$

where $\text{lookup} : VEnv \rightarrow Sto \rightarrow \Sigma$ with $\text{lookup } \rho \sigma := \sigma \circ \rho$ and

$$\Phi : (Sto \dashrightarrow Sto) \rightarrow (Sto \dashrightarrow Sto) : f \mapsto \text{cond}(\mathcal{B}[b] \circ (\text{lookup } \rho), f \circ \mathcal{E}'[c]\rho, \text{id}_{Sto})$$

Handling Procedures

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Handling Procedures

Procedure Environments

- **Procedure environments** now store **semantic** information:
 - So far: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$
 - Now: $PEnv' := \{\pi \mid \pi : PVar \dashrightarrow (Sto \dashrightarrow Sto)\}$, to be used in $\mathcal{C}''[\cdot] : Cmd \rightarrow VEnv \rightarrow PEnv' \rightarrow (Sto \dashrightarrow Sto)$

Procedure Environments

- **Procedure environments** now store **semantic** information:
 - So far: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$
 - Now: $PEnv' := \{\pi \mid \pi : PVar \dashrightarrow (Sto \dashrightarrow Sto)\}$, to be used in $\mathcal{C}''[\cdot] : Cmd \rightarrow VEnv \rightarrow PEnv' \rightarrow (Sto \dashrightarrow Sto)$

- **Procedure declarations** (“`proc P is c end`”) update procedure environment:

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

- **non-recursive** case: P not (indirectly) called within c
 $\Rightarrow \pi(P)$ immediately given by $\mathcal{C}''[c]\rho\pi$:

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

- **recursive** case: $\pi(P)$ must be a solution of equation $f = \mathcal{C}''[c]\rho\pi[P \mapsto f]$
(cf. fixpoint semantics of `while` loop – Slide 6.12):

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

where $\Psi : (Sto \dashrightarrow Sto) \rightarrow (Sto \dashrightarrow Sto) : f \mapsto \mathcal{C}''[c]\rho\pi[P \mapsto f]$

Procedure Environments

- **Procedure environments** now store **semantic** information:
 - So far: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$
 - Now: $PEnv' := \{\pi \mid \pi : PVar \dashrightarrow (Sto \dashrightarrow Sto)\}$, to be used in $\mathcal{C}''[\cdot] : Cmd \rightarrow VEnv \rightarrow PEnv' \rightarrow (Sto \dashrightarrow Sto)$

- **Procedure declarations** (“`proc P is c end`”) update procedure environment:

$$\text{upd}_\rho[\cdot] : PDec \times VEnv \times PEnv' \rightarrow PEnv'$$

- **non-recursive** case: P not (indirectly) called within c
 $\Rightarrow \pi(P)$ immediately given by $\mathcal{C}''[c]\rho\pi$:

$$\text{upd}_\rho[\text{proc } P \text{ is } c \text{ end}; \rho](\rho, \pi) := \text{upd}_\rho[\rho](\rho, \pi[P \mapsto \mathcal{C}''[c]\rho\pi])$$

- **recursive** case: $\pi(P)$ must be a solution of equation $f = \mathcal{C}''[c]\rho\pi[P \mapsto f]$
(cf. fixpoint semantics of `while` loop – Slide 6.12):

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

where $\Psi : (Sto \dashrightarrow Sto) \rightarrow (Sto \dashrightarrow Sto) : f \mapsto \mathcal{C}''[c]\rho\pi[P \mapsto f]$

- $\text{upd}_\rho[\varepsilon](\rho, \pi) := \pi$
- **Remark:** non-recursive is special instance of recursive case

Handling Procedures

Statement Semantics Including Procedures

So far: $\mathcal{C}'[\cdot] : Cmd \rightarrow VEnv \rightarrow (Sto \dashrightarrow Sto)$

Handling Procedures

Statement Semantics Including Procedures

So far: $\mathcal{E}'[\cdot] : \text{Cmd} \rightarrow \text{VEnv} \rightarrow (\text{Sto} \dashrightarrow \text{Sto})$

Definition 15.2 (Denotational semantics with procedures)

$$\mathcal{E}''[\cdot] : \text{Cmd} \rightarrow \text{VEnv} \rightarrow \text{PEnv}' \rightarrow (\text{Sto} \dashrightarrow \text{Sto})$$

is given by

$$\mathcal{E}''[\text{skip}]\rho \pi := \text{id}_{\text{Sto}}$$

$$\mathcal{E}''[x := a]\rho \pi \sigma := \sigma[\rho(x) \mapsto \mathcal{A}[a](\text{lookup } \rho \sigma)]$$

$$\mathcal{E}''[c_1; c_2]\rho \pi := (\mathcal{E}''[c_2]\rho \pi) \circ (\mathcal{E}''[c_1]\rho \pi)$$

$$\mathcal{E}''[\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}]\rho \pi := \text{cond}(\mathcal{B}[b] \circ (\text{lookup } \rho), \mathcal{E}''[c_1]\rho \pi, \mathcal{E}''[c_2]\rho \pi)$$

$$\mathcal{E}''[\text{while } b \text{ do } c \text{ end}]\rho \pi := \text{fix}(\Phi)$$

$$\mathcal{E}''[\text{call } P]\rho \pi := \pi(P)$$

$$\mathcal{E}''[\text{begin } v \rho c \text{ end}]\rho \pi \sigma := \mathcal{E}''[c]\rho' \pi' \sigma'$$

where $\text{upd}_v[v](\rho, \sigma) = (\rho', \sigma')$

$$\text{upd}_\rho[\rho](\rho', \pi) = \pi'$$

$$\text{lookup } \rho \sigma := \sigma \circ \rho$$

$$\Phi(f) := \text{cond}(\mathcal{B}[b] \circ (\text{lookup } \rho), f \circ \mathcal{E}''[c]\rho \pi, \text{id}_{\text{Sto}})$$

Two Examples

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Two Examples

Example: Non-Recursive Case

Example 15.3 (Non-recursive procedure call)

(also demonstrates static scoping principle)

```
c = begin
  var x;
  proc P is x := x - 1 end;
  x := 2; } c1
  begin
    var x;
    x := 3;
    call P;
  end; } c2
end
```

- Initial environments/store: $\rho_\emptyset \in VEnv$, $\pi_\emptyset \in PEnv'$, $\sigma_\emptyset \in Sto$
- Computation of $\mathcal{E}''[[c]]\rho_\emptyset \pi_\emptyset \sigma_\emptyset$: on the board

Two Examples

Example: Recursive Case

Example 15.4 (Recursive procedure call)

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

Diagram illustrating the structure of the code blocks:

- The block `proc F is ... end` is labeled c_1 .
- The block `y := 1; call F;` is labeled c_2 .
- The entire code block `c = begin ... end` is labeled p .

- Initial environments/store:
 - $\rho_1 := \rho_0[x \mapsto 0, y \mapsto 1] \in VEnv$
 - $\pi_0 \in PEnv'$
 - $\sigma_1 \in Sto$ (with $\sigma_1(0) \neq \perp \neq \sigma_1(1)$)
- Computation of $\mathcal{E}''[c]\rho_1 \pi_0 \sigma_1$:
on the board

Justification of Fixpoint Semantics

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Justification of Fixpoint Semantics

Justification of Fixpoint Semantics

Lemma 15.5

1. (cf. Lemma 7.9)

$(Sto \dashrightarrow Sto, \sqsubseteq)$ is a **CCPO** where $f \sqsubseteq g$ iff for all $\sigma, \sigma' \in Sto$: $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$

Justification of Fixpoint Semantics

Justification of Fixpoint Semantics

Lemma 15.5

1. (cf. Lemma 7.9)

$(Sto \dashrightarrow Sto, \sqsubseteq)$ is a **CCPO** where $f \sqsubseteq g$ iff for all $\sigma, \sigma' \in Sto$: $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$

2. (cf. Lemmata 7.13 and 7.16)

Let $b \in BExp$, $c \in Cmd$, $\rho \in VEnv$, $\pi \in PEnv'$, and $\Phi : (Sto \dashrightarrow Sto) \rightarrow (Sto \dashrightarrow Sto)$ with $\Phi(f) := \text{cond}(\mathfrak{B}[[b]] \circ (\text{lookup } \rho), f \circ \mathfrak{E}''[[c]]\rho \pi, \text{id}_{Sto})$.

Then Φ is **monotonic and continuous** w.r.t. $(Sto \dashrightarrow Sto, \sqsubseteq)$.

Justification of Fixpoint Semantics

Justification of Fixpoint Semantics

Lemma 15.5

1. (cf. Lemma 7.9)

$(\text{Sto} \dashrightarrow \text{Sto}, \sqsubseteq)$ is a **CCPO** where $f \sqsubseteq g$ iff for all $\sigma, \sigma' \in \text{Sto}$: $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$

2. (cf. Lemmata 7.13 and 7.16)

Let $b \in \text{BExp}$, $c \in \text{Cmd}$, $\rho \in \text{VEnv}$, $\pi \in \text{PEnv}'$, and $\Phi : (\text{Sto} \dashrightarrow \text{Sto}) \rightarrow (\text{Sto} \dashrightarrow \text{Sto})$ with $\Phi(f) := \text{cond}(\mathfrak{B}[[b]] \circ (\text{lookup } \rho), f \circ \mathfrak{E}''[[c]]\rho \pi, \text{id}_{\text{Sto}})$.

Then Φ is **monotonic and continuous** w.r.t. $(\text{Sto} \dashrightarrow \text{Sto}, \sqsubseteq)$.

3. Let $\text{proc } P \text{ is } c \text{ end} \in \text{PDec}$, $\rho \in \text{VEnv}$, $\pi \in \text{PEnv}'$, and

$\Psi : (\text{Sto} \dashrightarrow \text{Sto}) \rightarrow (\text{Sto} \dashrightarrow \text{Sto})$ with $\Psi(f) := \mathfrak{E}''[[c]]\rho \pi[P \mapsto f]$.

Then Ψ is **monotonic and continuous** w.r.t. $(\text{Sto} \dashrightarrow \text{Sto}, \sqsubseteq)$.

Proof.

omitted □

Summary: Blocks and Procedures in Operational/Denotational Semantics

Outline of Lecture 15

Recap: Operational Semantics of Blocks and Procedures

Denotational Semantics of Blocks and Procedures

Handling Variable Declarations

Handling Procedures

Two Examples

Justification of Fixpoint Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures
- Requires concept of **locations** to support instantiation of variables

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures
- Requires concept of **locations** to support instantiation of variables
- **Static scoping**: meaning of identifier determined by declaration (rather than calling) context

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures
- Requires concept of **locations** to support instantiation of variables
- **Static scoping**: meaning of identifier determined by declaration (rather than calling) context
- Meaning of **variable declaration**: storage allocation

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures
- Requires concept of **locations** to support instantiation of variables
- **Static scoping**: meaning of identifier determined by declaration (rather than calling) context
- Meaning of **variable declaration**: storage allocation
- Meaning of **procedure call**:
 - operationally: **execution** of procedure body
 - ⇒ procedure environment holds body statement (“symbol table”)
 - denotationally: **application** of procedure meaning
 - ⇒ procedure environment holds (partial) store transformation
 - recursive behaviour again handled by **fixpoint approach**

Summary: Blocks and Procedures in Operational/Denotational Semantics

Summary: Blocks and Procedures in Operational/Denotational Semantics

- **Blocks** allow to declare local variables and recursive procedures
- Requires concept of **locations** to support instantiation of variables
- **Static scoping**: meaning of identifier determined by declaration (rather than calling) context
- Meaning of **variable declaration**: storage allocation
- Meaning of **procedure call**:
 - operationally: **execution** of procedure body
 - ⇒ procedure environment holds body statement (“symbol table”)
 - denotationally: **application** of procedure meaning
 - ⇒ procedure environment holds (partial) store transformation
 - recursive behaviour again handled by **fixpoint approach**
- Further extensions:
 - **axiomatic semantics** of procedures (see following lecture)
 - **procedure parameters** and **higher-order procedures**