UNIVERSITÉ Grenoble Alpes

Inria

Programming Language Semantics and Compiler Design

(Sémantique des Langages de Programmation et Compilation)

Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Yliès Falcone

ylies.falcone@univ-grenoble-alpes.fr — www.ylies.fr

Univ. Grenoble Alpes, and LIG-Inria team CORSE

Master of Sciences in Informatics at Grenoble (MoSIG)
Master 1 info

Univ. Grenoble Alpes - UFR IM^{2}AG
www.univ-grenoble-alpes.fr — im2ag.univ-grenoble-alpes.fr

Academic Year 2020 - 2021

About

This document recalls some of the main notations, definitions, and results related to **While**, **Block**, **Proc**.

More specifically it recalls:

- the syntax,
- the static semantic analysis ($\approx$ typing),
- the operational semantics (natural and structural), and
- the axiomatic semantics.

Disclaimer

The document is not exhaustive; the reference documents remain the lecture slides.

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)

Structural Operational Semantics

Axiomatic Semantics

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax
- Syntax of Expressions Used in Semantic Analysis
- Syntax of **While**
- Syntax of **Block**
- Syntax of **Proc**

Semantic Analysis (typing)

Natural Operational Semantics (NOS)

Structural Operational Semantics

Axiomatic Semantics

Syntax of Expressions Used used in Semantic Analysis

There is one sort of expressions because types are not yet known.

Expressions are defined by an abstract grammar

There is only one sort of expressions.

$$\mathtt{e} ::= \mathtt{true} \mid \mathtt{false} \mid \mathtt{n} \mid \mathtt{x} \mid \mathtt{e\ opa\ e} \mid \mathtt{e\ oprel\ e} \mid \mathtt{e\ opb\ e}$$

where `true` and `false` are the boolean constants, `n` denotes a natural number, and `x` denotes a variable, and binary operators: arithmetic (`opa`), boolean (`opb`) and relational (`oprel`).

Syntax of Expressions Used used in Semantic Analysis

- Numbers: $n \in \mathbf{Num} = \{0, \ldots, 9\}^+$
- Variables: $x \in \mathbf{Var}$
- Arithmetic expressions:

$$a \in \mathbf{Aexp}$$
$$a ::= n \mid x \mid a + a \mid a * a \mid a - a$$

- Boolean expressions:

$$b \in \mathbf{Bexp}$$
$$b ::= \text{true} \mid \text{false} \mid a = a \mid a \leq a \mid \neg b \mid b \wedge b$$

Num, **Var**, **Aexp**, and **Bexp** are *syntactic categories*.

Remark Other operators for artihmetical expressions can be defined from the proposed ones. □

Syntax of **While**

Statements in semantic analysis are defined by an abstract grammar

S	::=	x := e	(assignment of an expression to a variable x)
		| skip	(doing nothing)
		| S ; S	(sequential composition)
		| if *e* then *S* else *S* fi	(conditional composition)
		| while *e* do *S* od	(iterative and unbounded composition)

Statements in operational semantics are defined by an abstract grammar

S	∈	**Stm**	
S	::=	x := a	(assignment of an arithmetic expression a to a variable x)
		| skip	(doing nothing)
		| S ; S	(sequential composition)
		| if *b* then *S* else *S* fi	(conditional composition)
		| while *e* do *S* od	(iterative and unbounded composition)

Stm is a *syntactic category*

Blocks and variable declarations: syntax

Extending language **While** to handle variable declarations.

Definition 1 (Language Block)

$$S \in \mathbf{Stm}$$
$$S ::= x := a \mid \text{skip} \mid S; S$$
$$\mid \text{if } b \text{ then } S \text{ else } S \text{ fi}$$
$$\mid \text{while } b \text{ do } S \text{ od}$$
$$\mid \textbf{begin } D_V \ \ S \textbf{ end}$$

Definition 2 (Syntactic category $\mathbf{Dec}_V$)

$$D_V ::= \text{var } x;\ D_V \mid \text{var } x := a;\ D_V \mid \epsilon$$

Introducing Procedures in the syntax

Extending **Block** with procedure declarations.

Definition 3 (Language Proc)

$$\begin{array}{lcl} S & \in & \textbf{Stm} \\ S & ::= & x := a \mid \text{skip} \mid S_1; S_2 \\ & & \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi} \\ & & \mid \text{while } b \text{ do } S \text{ od} \\ & & \mid \textbf{begin } D_V\ D_P\ \ S \textbf{ end} \mid \text{call } p \end{array}$$

Definition 4 (Syntactic category $\textbf{Dec}_P$)

$$D_P ::= \text{proc } p \text{ is } S;\ D_P \mid \epsilon$$

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)
- Typing of Expressions
- Typing of **While**
- Typing of **Block**
- Typing of **Proc**

Natural Operational Semantics (NOS)

Structural Operational Semantics

Axiomatic Semantics

About semantic analysis - typing

Ingredients used in the formalization of the type system

- Environment Γ: $Name \overset{\text{part.}}{\to} Types$.
- Judgments $\Gamma \vdash t : \tau$.
  "In environment Γ, term t is well-typed and has type τ."
  (free variables of t belong to the domain of Γ)
- Type system

Inference rules	Axioms
$\dfrac{\Gamma_1 \vdash \mathcal{A}_1 \quad \cdots \quad \Gamma_n \vdash \mathcal{A}_n}{\Gamma \vdash \mathcal{A}}$	$\Gamma \vdash \mathcal{A}$

Remark A type system is an inference system. □

Type System for Expressions

Axioms		
bool. constant		int. constant
$\dfrac{}{\Gamma \vdash \texttt{true} : \textbf{Bool}}$	$\dfrac{}{\Gamma \vdash \texttt{false} : \textbf{Bool}}$	$\dfrac{}{\Gamma \vdash \texttt{n} : \textbf{Int}}$

Inference Rules			
variables	int opbin	bool. opbin	relational operators
$\dfrac{\Gamma(x) = t}{\Gamma \vdash x : t}$	$\dfrac{\Gamma \vdash e_1 : \textbf{Int} \quad \Gamma \vdash e_2 : \textbf{Int}}{\Gamma \vdash e_1 \text{ opa } e_2 : \textbf{Int}}$	$\dfrac{\Gamma \vdash e_1 : \textbf{Bool} \quad \Gamma \vdash e_2 : \textbf{Bool}}{\Gamma \vdash e_1 \text{ opb } e_2 : \textbf{Bool}}$	$\dfrac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 \texttt{ oprel } e_2 : \textbf{Bool}}$

Type system for Statements

Judgments
- $\Gamma \vdash t : \tau$ means "In environment Γ, term t is well-typed and has type τ."
- $\Gamma \vdash\ S$ means "statement S is well-typed within environment Γ"

Axioms	
Assignment	Skip
$\dfrac{\Gamma \vdash e : t \quad \Gamma \vdash x : t}{\Gamma \vdash x := e}$	$\dfrac{}{\Gamma \vdash \texttt{skip}}$

Inference rules		
Sequence	Iteration	Conditional
$\dfrac{\Gamma \vdash S_1 \quad \Gamma \vdash S_2}{\Gamma \vdash S_1; S_2}$	$\dfrac{\Gamma \vdash e : \text{Bool} \quad \Gamma \vdash S}{\Gamma \vdash \text{while } e \text{ do } S \text{ od}}$	$\dfrac{\Gamma \vdash e : \textbf{Bool} \quad \Gamma \vdash S_1 \quad \Gamma \vdash S_2}{\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi}}$

Type system for **Block**

Judgments
- $\Gamma \vdash\ D_V \mid \Gamma_l$ means
  "Variable declarations D_V are well typed within variable environment Γ_V. Moreover, variable declarations D_V update variable environment Γ_V into Γ'_V".
- $\Gamma \vdash\ S$ means
  "statement S is well-typed within environment Γ"

- $\texttt{DV}(D_v)$ denotes the set of variables **declared** in D_v.
- $\Gamma[y \mapsto \tau]$ denotes the environment Γ' such that:
  - $\Gamma'(x) = \Gamma(x)$ if $x \neq y$
  - $\Gamma'(y) = \tau$

Extending the Type System

Inference rule for Blocks

$$\frac{\Gamma \vdash D_V \mid \Gamma_l \quad \Gamma_l \vdash S}{\Gamma \vdash \textbf{begin } D_V\ S \textbf{ end}}$$

Inference rules for declarations

Sequential evaluation

$$\frac{}{\Gamma \vdash \epsilon \mid \Gamma} \qquad \frac{\Gamma \vdash e : t \quad \Gamma[x \mapsto t] \vdash D_V \mid \Gamma_l \quad x \notin \texttt{DV}(D_V)}{\Gamma \vdash \textbf{var } x := e\ ;\ D_V \mid \Gamma_l}$$

Collateral evaluation

$$\frac{}{\Gamma \vdash \epsilon \mid \Gamma} \qquad \frac{\Gamma \vdash e : t \quad \Gamma \vdash D_V \mid \Gamma_l \quad x \notin \texttt{DV}(D_V)}{\Gamma \vdash \textbf{var } x := e; D_V \mid \Gamma_l[x \mapsto t]}$$

The orange premise ensures that a variable should be declared at most once

Type system for **Proc**

$DP(D_P)$ denotes the set of procedures **declared** in D_P.

Procedure environment $\Gamma_P : Name \to \{proc\}$ (partial)

Extending judgments:
- $(\Gamma_V, \Gamma_P) \vdash D_P \mid \Gamma'_P$ means
  "Procedure declarations in D_P are well-typed within variable and procedure environments (Γ_V, Γ_P). Moreover, procedure declarations in D_P update procedure environment Γ_P into Γ'_P."
- $(\Gamma_V, \Gamma_P) \vdash S$ means
  "Statement S is well-typed within variable and procedure environments (Γ_V, Γ_P).

Static Binding for Procedures and Variables

Block $$\frac{\Gamma_V \vdash D_V \mid \Gamma'_V \quad (\Gamma'_V,\Gamma_P)\vdash D_P \mid \Gamma'_P \quad (\Gamma'_V,\Gamma'_P)\vdash S}{(\Gamma_V,\Gamma_P)\vdash \textbf{begin } D_V\ D_P\ S \textbf{ end}}$$

Empty proc. decl. $$\frac{}{(\Gamma_V,\Gamma_P)\vdash \epsilon \mid \Gamma_P}$$

Non-empty proc. decl. $$\frac{(\Gamma_V,\Gamma_P)\vdash S \quad (\Gamma_V,\Gamma_P[p\mapsto \textbf{proc}])\vdash D_P\mid \Gamma'_P \quad p\notin DP(D_P)}{(\Gamma_V,\Gamma_P)\vdash \textbf{proc } p \textbf{ is } S\ ;\ D_P \mid \Gamma'_P}$$

Call $$\frac{\Gamma_P(p)=\texttt{proc}}{(\Gamma_V,\Gamma_P)\vdash\ \textbf{call } p}$$

Remark The procedure environment is a partial function in $Name \to \{proc\}$. □

Remark The same considerations (as those made for variable declarations) apply concerning the possibility of redeclarations and the priority between declarations. □

Dynamic Binding for Procedures and Variables

Block $$\frac{\Gamma_V \vdash D_V \mid \Gamma'_V \quad (\Gamma'_V,\Gamma'_P)\vdash S \quad \texttt{udef}(D_P)}{(\Gamma_V,\Gamma_P)\vdash \textbf{begin } D_V\ D_P\ S \textbf{ end}}$$

Call $$\frac{(\Gamma_V,\Gamma_P)\vdash S}{(\Gamma_V,\Gamma_P)\vdash\ \textbf{call } p}\ \Gamma_P(p)=S$$

- where $\Gamma'_P = \texttt{upd}(\Gamma_P, D_P)$
- with:

$$\begin{aligned}
\texttt{upd}(\Gamma_P, \textbf{proc } p \textbf{ is } S\ ;\ D_P) &= \texttt{upd}(\Gamma_P[p\mapsto S], D_P)\\
\texttt{upd}(\Gamma_P,\varepsilon) &= \Gamma_P\\
\texttt{udef}(\textbf{proc } p \textbf{ is } S\ ;\ D_P)) &= \texttt{udef}(D_P)\wedge p\notin DP(D_P)\\
\texttt{udef}(\varepsilon) &= \texttt{true}
\end{aligned}$$

Remark The procedure environment is a partial function in $Name \to$ **Stm**. □

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)
NOS of Expressions
NOS of **While**
NOS of **Block**
NOS of **Proc**

Structural Operational Semantics

Axiomatic Semantics

Semantic domains and substitution

- Integers: $\mathbb{Z}$
- Booleans: $\mathbb{B} = \{\textbf{tt}, \textbf{ff}\}$
- States: **State** $=$ **Var** $\to \mathbb{Z}$

Definition 5 (Substituing a value to a variable)
Let $v \in \mathbb{Z}$. Then, $\sigma[y\mapsto v]$ denotes the state σ' such that:

$$\text{for all } x\in \textbf{Var}, \sigma'(x) = \begin{cases}\sigma(x) & \text{if } x\neq y,\\ v & \text{otherwise.}\end{cases}$$

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)
NOS of Expressions
NOS of **While**
NOS of **Block**
NOS of **Proc**

Structural Operational Semantics

Axiomatic Semantics

Semantic functions for arithmetic and boolean expressions

- Numerals: integers

$$\begin{aligned}\mathcal{N} &: \textbf{Num}\to\mathbb{N}\\ \mathcal{N}(n_1\cdots n_k) &= \Sigma_{i=1}^{k} n_i\times 10^{k-i}\end{aligned}$$

- Arithmetic expressions: for each state, a value in $\mathbb{Z}$

$$\mathcal{A} : \textbf{Aexp} \to (\textbf{State}\to\mathbb{Z})$$

$$\begin{aligned}
\mathcal{A}[n]\sigma &= \mathcal{N}(n)\\
\mathcal{A}[x]\sigma &= \sigma(x)\\
\mathcal{A}[a_1+a_2]\sigma &= \mathcal{A}[a_1]\sigma +_I \mathcal{A}[a_2]\sigma\\
\mathcal{A}[a_1*a_2]\sigma &= \mathcal{A}[a_1]\sigma *_I \mathcal{A}[a_2]\sigma\\
\mathcal{A}[a_1-a_2]\sigma &= \mathcal{A}[a_1]\sigma -_I \mathcal{A}[a_2]\sigma
\end{aligned}$$

- Boolean expressions: for each state, a value in $\mathbb{B}$

$$\mathcal{B} : \textbf{Bexp}\to(\textbf{State}\to\mathbb{B})$$

$$\begin{aligned}
\mathcal{B}[\text{true}]\sigma &= \textbf{tt}\\
\mathcal{B}[\text{false}]\sigma &= \textbf{ff}\\
\mathcal{B}[\neg b]\sigma &= \neg_{\mathbb{B}}\mathcal{B}[b]\sigma\\
\mathcal{B}[a_1=a_2]\sigma &= \mathcal{A}[a_1]\sigma =_I \mathcal{A}[a_2]\sigma\\
\mathcal{B}[a_1\leq a_2]\sigma &= \mathcal{A}[a_1]\sigma \leq_I \mathcal{A}[a_2]\sigma\\
\mathcal{B}[b_1\wedge b_2]\sigma &= \mathcal{B}[b_1]\sigma \wedge_{\mathbb{B}} \mathcal{B}[b_2]\sigma
\end{aligned}$$

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)
NOS of Expressions
NOS of **While**
NOS of **Block**
NOS of **Proc**

Structural Operational Semantics

Axiomatic Semantics

Semantic and transition system for statements

- Statements: $\mathcal{S}_{\rm ns} : \textbf{Stm}\to(\textbf{State}\xrightarrow{part.}\textbf{State})$

$$\mathcal{S}_{\rm ns}[S]\sigma = \begin{cases}\sigma' & \text{if } (S,\sigma)\to\sigma',\\ \text{undef} & \text{otherwise,}\end{cases}$$

Relation $\to$ is defined in terms of a transition system.

Transition system for Natural Operational Semantics

- Configurations: $(\textbf{Stm}\times\textbf{State})\cup\textbf{State}$.
- Final configurations (a subset of the set of configurations): **State**.
  (Configurations in **Stm** $\times$ **State** are called non-final.)
- Transition relation: $\to\subseteq(\textbf{Stm}\times\textbf{State})\times\textbf{State}$
  We note $(S,\sigma)\to\sigma'$, when the program moves from configuration (S,σ) to the terminal configuration σ'.
  - "The execution of S from σ *terminates* in state σ'"
  - Goal: to describe how the result of a program execution is obtained.

Axioms and rules defining the transition relation

Axioms

$$\frac{}{(x := a, \sigma) \to \sigma[x \mapsto \mathcal{A}[a]\sigma]}$$

$$\frac{}{(\text{skip}, \sigma) \to \sigma}$$

Rule for Sequential Statements

$$\frac{(S_1, \sigma) \to \sigma' \quad (S_2, \sigma') \to \sigma''}{(S_1; S_2, \sigma) \to \sigma''}$$

Rules for Conditional Statements

$$\frac{(S_1, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \to \sigma'} \text{ if } \mathcal{B}[b]\sigma = \mathbf{tt}$$

$$\frac{(S_2, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \to \sigma'} \text{ if } \mathcal{B}[b]\sigma = \mathbf{ff}$$

Rules for Iterative Statements (unbounded iteration)

$$\frac{(S, \sigma) \to \sigma' \quad (\text{while } b \text{ do } S \text{ od}, \sigma') \to \sigma''}{(\text{while } b \text{ do } S \text{ od}, \sigma) \to \sigma''} \text{ if } \mathcal{B}[b]\sigma = \mathbf{tt}$$

$$\frac{}{(\text{while } b \text{ do } S \text{ od}, \sigma) \to \sigma} \text{ if } \mathcal{B}[b]\sigma = \mathbf{ff}$$

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)
- NOS of Expressions
- NOS of While
- NOS of **Block**
- NOS of Proc

Structural Operational Semantics

Axiomatic Semantics

Preliminaries: stacks - definition

We use a stack structure to *manage local declarations*.

Let $\mathcal{F}$ be a set of (partial) functions with the same signature.

Elements of $\mathcal{F}$ are denoted by f (which can be subscripted and primed).

We note [] the empty partial function (defined nowhere, i.e., $\mathtt{Dom}([]) = \emptyset$)

Stack notation over partial functions

- The set of stacks over $\mathcal{F}$ is denoted by $\mathcal{F}^*$.
- Elements of $\mathcal{F}^*$ are noted $\hat{f}, \hat{f}_1, \hat{f}_2, \ldots$

Definition 6 (Stack)

Stacks are defined inductively:

- The empty stack is denoted by $\emptyset$.
- Given a stack $\hat{f}$ and a partial function f, $\hat{f} \oplus f$ denotes the stack composed of the stack $\hat{f}$ on top of which is partial function f.

Preliminaries: evaluation on stacks and substitution on partial functions

Definition 7 (Evaluation on stacks)

Evaluation of a value x in the domain of the partial functions is defined *inductively* on stacks:

- For a non empty stack $\hat{f} \oplus f'$:
$$(\hat{f} \oplus f')(x) = \begin{cases} f'(x) & \text{if } x \in \mathtt{Dom}(f'), \\ \hat{f}(x) & \text{otherwise.} \end{cases}$$
($\hat{f} \oplus f'$ is the stack resulting in pushing function f' to stack $\hat{f}$.)
- For the empty stack: $\emptyset(x) = \mathtt{undef}$.

Definition 8 (Substitution on partial functions)

Given some (partial) function $f : E \to F$, $y \in E$, and $v \in F$, $f[y \mapsto v]$ is the partial function defined as:

$$f[y \mapsto v](x) = \begin{cases} v & \text{if } x = y, \\ f(x) & \text{otherwise.} \end{cases}$$

Refining the notion of state

States are replaced by a symbol table plus a memory

Definition 9 (Symbol table: variable environment)

$$\mathbf{Env}_V = \mathbf{Var} \overset{part.}{\to} \mathbf{Loc}$$

ρ denotes an element of $\mathbf{Env}_V$.
Thus, $\hat{\rho} \in {\mathbf{Env}_V}^*$ denotes a stack of tables.

Definition 10 (Memory)

$$\mathbf{Store} = \mathbf{Loc} \overset{part.}{\to} \mathbb{Z}$$

σ denotes an element of **Store**.

Notation: new() is a function that returns a *fresh* memory location.

Revisiting the semantic functions for arithmetic and Boolean expressions

Definition 11 (Semantic function for arithmetic expressions)

$$\mathcal{A} : \mathbf{Aexp} \to (({\mathbf{Env}_V}^* \times \mathbf{Store}) \to \mathbb{Z})$$

$$\begin{array}{ll} \mathcal{A}[n](\hat{\rho}, \sigma) & = \mathcal{N}[n] \\ \mathcal{A}[x](\hat{\rho}, \sigma) & = \sigma(\hat{\rho}(x)) \\ \mathcal{A}[a_1 + a_2](\hat{\rho}, \sigma) & = \mathcal{A}[a_1](\hat{\rho}, \sigma) +_I \mathcal{A}[a_2](\hat{\rho}, \sigma) \\ \mathcal{A}[a_1 * a_2](\hat{\rho}, \sigma) & = \mathcal{A}[a_1](\hat{\rho}, \sigma) *_I \mathcal{A}[a_2](\hat{\rho}, \sigma) \\ \mathcal{A}[a_1 - a_2](\hat{\rho}, \sigma) & = \mathcal{A}[a_1](\hat{\rho}, \sigma) -_I \mathcal{A}[a_2](\hat{\rho}, \sigma) \end{array}$$

Definition 12 (Semantic function for boolean expressions)

$$\mathcal{B} : \mathbf{Bexp} \to (({\mathbf{Env}_V}^* \times \mathbf{Store}) \to \mathbb{B})$$

Same principle.

Revisiting the semantic of statements

Definition 13 (Transition system for **While**)

Configurations: $(\mathbf{Stm} \times {\mathbf{Env}_V}^* \times \mathbf{Store}) \cup \mathbf{Store}$

Final configurations: **Store**

Transitions: $(\mathbf{Stm} \times {\mathbf{Env}_V}^* \times \mathbf{Store}) \cup \mathbf{Store}$

- Assignment:
$$\frac{}{(x := a, \hat{\rho}, \sigma) \to \sigma[\hat{\rho}(x) \mapsto \mathcal{A}[a](\hat{\rho}, \sigma)]}$$
- Skip:
$$\frac{}{(\text{skip}, \hat{\rho}, \sigma) \to \sigma}$$
- While:
  - if $\mathcal{B}[b](\hat{\rho}, \sigma) = \mathbf{ff}$
  $$\frac{}{(\text{while } b \text{ do } S \text{ od}, \hat{\rho}, \sigma) \to \sigma}$$
  - if $\mathcal{B}[b](\hat{\rho}, \sigma) = \mathbf{tt}$
  $$\frac{(S, \hat{\rho}, \sigma) \to \sigma' \quad (\text{while } b \text{ do } S \text{ od}, \hat{\rho}, \sigma') \to \sigma''}{(\text{while } b \text{ do } S \text{ od}, \hat{\rho}, \sigma) \to \sigma''}$$
- Sequential composition:
$$\frac{(S_1, \hat{\rho}, \sigma) \to \sigma' \quad (S_2, \hat{\rho}, \sigma') \to \sigma''}{(S_1; S_2, \hat{\rho}, \sigma) \to \sigma''}$$
- If:
  - if $\mathcal{B}[b](\hat{\rho}, \sigma) = \mathbf{ff}$
  $$\frac{(S_1, \hat{\rho}, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \hat{\rho}, \sigma) \to \sigma'}$$
  - if $\mathcal{B}[b](\hat{\rho}, \sigma) = \mathbf{tt}$
  $$\frac{(S_2, \hat{\rho}, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \hat{\rho}, \sigma) \to \sigma'}$$

Transition rules for blocks

Definition 14 (Transition system for variable declarations)

- Configurations: $(\mathbf{Dec}_V \times {\mathbf{Env}_V}^* \times \mathbf{Env}_V \times \mathbf{Store}) \cup (\mathbf{Env}_V \times \mathbf{Store})$ i.e., of the form $(D_v, \hat{\rho}, \rho', \sigma)$ or (ρ', σ), where:
  - D_V: sequence of declarations
  - ρ': local symbol table
  - $\hat{\rho}$: global symbol table
  - σ: memory
- Final configurations: $\mathbf{Env}_V \times \mathbf{Store}$ (i.e., of the form (ρ', σ))
- Transitions:
$$\to_D \subseteq (\mathbf{Dec}_V \times {\mathbf{Env}_V}^* \times \mathbf{Env}_V \times \mathbf{Store}) \times (\mathbf{Env}_V \times \mathbf{Store})$$
i.e., of the form $(D_v, \hat{\rho}, \rho', \sigma) \to_D (\rho'', \sigma'')$

$$(\epsilon, \hat{\rho}, \rho', \sigma) \to_D (\rho', \sigma)$$

$$\frac{(D_V, \hat{\rho}, \rho[x \mapsto l], \sigma) \to_D (\rho', \sigma')}{(\text{var } x;\ D_V, \hat{\rho}, \rho, \sigma) \to_D (\rho', \sigma')}$$

$$\frac{(D_V, \hat{\rho}, \rho[x \mapsto l], \sigma[l \mapsto \mathcal{A}[a](\hat{\rho} \oplus \rho, \sigma)]) \to_D (\rho', \sigma')}{(\text{var } x := a;\ D_V, \hat{\rho}, \rho, \sigma) \to_D (\rho', \sigma')}$$

Transition rules for blocks

Transition system for statements

Definition 15 (Natural operational semantics of **Block**)

- Configurations:

$$\textbf{Stm} \times \textbf{Env}_V{}^* \times \textbf{Store} \cup \textbf{Store}$$

- Transitions:

$$\frac{(D_V, \hat{\rho}, [], \sigma) \rightarrow_D (\rho_l, \sigma') \quad (S, \hat{\rho} \oplus \rho_l, \sigma') \rightarrow \sigma''}{(\textbf{begin } D_V \;\; S \textbf{ end}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

- **OR** Transitions (when there is only un-initialised variables)

$$\frac{(D_V, []) \rightarrow_D \rho_l \quad (S, \hat{\rho} \oplus \rho_l, \sigma) \rightarrow \sigma'}{(\textbf{begin } D_V \;\; S \textbf{ end}, \hat{\rho}, \sigma) \rightarrow \sigma'}$$

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)
- NOS of Expressions
- NOS of **While**
- NOS of **Block**
- NOS of **Proc**

Structural Operational Semantics

Axiomatic Semantics

Semantics with dynamic scope for variables and procedures

Semantic domains

Procedure names belong to a syntactic category called **Pname**.

Semantic domains for dynamic scope

$\textbf{Env}_V$	=	$\textbf{Var} \xrightarrow{part.} \textbf{Loc} \ni \rho$	Variable environment
Store	=	$\textbf{Loc} \xrightarrow{part.} \mathbb{Z} \ni \sigma$	Store
$\textbf{Env}_P$	=	$\textbf{Pname} \xrightarrow{part.} \textbf{Stm} \ni \lambda$	Procedure environment

Additional/replacement semantic domains for static scope

$\textbf{Env}_P$	=	$\textbf{Pname} \xrightarrow{part.} \textbf{Stm} \times \textbf{Env}_P{}^* \times \textbf{Env}_V{}^* \ni \lambda$	Local procedure environment
$\textbf{Env}_P{}^*$	=	stacks over $\textbf{Env}_P \ni \hat{\lambda}$	Global procedure env.

Semantics with dynamic scope: transition system

Configurations: $\underbrace{(\textbf{Stm} \times \textbf{Env}_P{}^* \times \textbf{Env}_V{}^* \times \textbf{Store})}_{\text{non-final configurations}} \cup \underbrace{\textbf{Store}}_{\text{final configurations}}$

Transition rules:

$$\frac{(D_V, \hat{\rho}, [], \sigma) \rightarrow_D (\rho_l, \sigma') \quad (S, \hat{\lambda} \oplus \text{upd}([], D_P), \hat{\rho} \oplus \rho_l, \sigma') \rightarrow \sigma''}{(\textbf{begin } D_V \; D_P \;\; S \textbf{ end}, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

OR (when there is only uninitialised variables):

$$\frac{(D_V, \hat{\rho}) \rightarrow_D \hat{\rho}' \quad (S, \hat{\lambda} \oplus \text{upd}([], D_P), \hat{\rho}', \sigma) \rightarrow \sigma''}{(\textbf{begin } D_V \; D_P \;\; S \textbf{ end}, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

where $\text{upd}(\lambda, \epsilon) = \lambda$ and $\text{upd}(\lambda, \text{proc } p \text{ is } S; D_P) = \text{upd}(\lambda[p \mapsto S], D_P)$

$$\frac{(\hat{\lambda}(p), \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma'}{(\text{call } p, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma'}$$

We "load" the code associated with p.

Updating the rule for sequential composition:

$$\frac{(S_1, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma' \quad (S_2, \hat{\lambda}, \hat{\rho}, \sigma') \rightarrow \sigma''}{(S_1; S_2, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

Remark S_1 and S_2 execute within the same environments. □

Similarly, other rules are adapted in a straightforward manner...

Semantics with static scope for variables and procedures: transition system

Definition 16 (Updating the procedure environment)

$$\text{upd}: \underbrace{\textbf{Env}_P{}^*}_{\substack{\text{global}\\ \text{proc. env.}}} \times \underbrace{\textbf{Env}_V{}^*}_{\substack{\text{global}\\ \text{var. env.}}} \times \underbrace{\textbf{Env}_P}_{\substack{\text{current local}\\ \text{proc. env.}}} \times \underbrace{\textbf{Dec}_P}_{\substack{\text{procedure}\\ \text{declaration}}} \longrightarrow \underbrace{\textbf{Env}_P}_{\substack{\text{produced}\\ \text{proc. env.}}}$$

- $\text{upd}(\hat{\lambda}_g, \hat{\rho}, \lambda_l, \epsilon) = \lambda_l$, and
- $\text{upd}(\hat{\lambda}_g, \hat{\rho}, \lambda_l, \text{proc } p \text{ is } S; D_P) = \text{upd}(\hat{\lambda}_g, \hat{\rho}, \lambda_l[p \mapsto (S, \hat{\lambda}_g \oplus \lambda_l, \hat{\rho})], D_P)$.

Definition 17 (Transition system for **Proc** with static scope)

Configurations: $\underbrace{(\textbf{Stm} \times \textbf{Env}_P{}^* \times \textbf{Env}_V{}^* \times \textbf{Store})}_{\text{non-final configurations}} \cup \underbrace{\textbf{Store}}_{\text{final configurations}}$

Transition rules:

- Block:

$$\frac{(D_V, \hat{\rho}, [], \sigma) \rightarrow_D (\rho_l, \sigma') \quad (S, \hat{\lambda} \oplus \text{upd}(\hat{\lambda}, \hat{\rho} \oplus \rho_l, [], D_P), \hat{\rho} \oplus \rho_l, \sigma') \rightarrow \sigma''}{(\textbf{begin } D_V \; D_P \;\; S \textbf{ end}, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

- Procedure call:

$$\frac{(S, \hat{\lambda}', \hat{\rho}', \sigma) \rightarrow \sigma''}{(\text{call } p, \hat{\lambda}, \hat{\rho}, \sigma) \rightarrow \sigma''}$$

where $\hat{\lambda}(p) = (S, \hat{\lambda}', \hat{\rho}')$.

We "load" the code and environments associated with p *(memory is "loaded" as is).*

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)

Structural Operational Semantics

Axiomatic Semantics

Transition system

Transition system for structural operational semantics

1. $\Gamma = (\textbf{Stm} \times \textbf{State}) \cup \textbf{State}$
2. $T = \textbf{State}$
3. $\Rightarrow \subseteq (\textbf{Stm} \times \textbf{State}) \times ((\textbf{Stm} \times \textbf{State}) \cup \textbf{State})$
4. $\Rightarrow$ defined by derivation sequences

Non-final configurations are related to non-final and final ones.

Rules defining the transitions

Axioms

$$\frac{}{(\text{skip}, \sigma) \Rightarrow \sigma}\,[\text{skip}_{\text{sos}}] \qquad \frac{}{(x := a, \sigma) \Rightarrow \sigma[x \mapsto \mathcal{A}[a]\sigma]}\,[\text{ass}_{\text{sos}}]$$

Rules for sequential statements

$$\frac{(S_1, \sigma) \Rightarrow \sigma'}{(S_1; S_2, \sigma) \Rightarrow (S_2, \sigma')}\,[\text{comp}^1_{\text{sos}}] \qquad \frac{(S_1, \sigma) \Rightarrow (S_1', \sigma')}{(S_1; S_2, \sigma) \Rightarrow (S_1'; S_2, \sigma')}\,[\text{comp}^2_{\text{sos}}]$$

"execution of S_1 *has* terminated" "execution of S_1 *has not* terminated"

Rules for conditional statements

If $\mathcal{B}[b]\sigma = \textbf{tt}$, then

$$\frac{}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \Rightarrow (S_1, \sigma)}\,[\text{if}^{\textbf{tt}}_{\text{sos}}]$$

If $\mathcal{B}[b]\sigma = \textbf{ff}$, then

$$\frac{}{(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, \sigma) \Rightarrow (S_2, \sigma)}\,[\text{if}^{\textbf{ff}}_{\text{sos}}]$$

Rule for iterative statements (unbounded)

$$\frac{}{(\text{while } b \text{ do } S \text{ od}, \sigma) \Rightarrow (\text{if } b \text{ then } (S; \text{while } b \text{ do } S \text{ od}) \text{ else skip fi}, \sigma)}\,[\text{while}_{\text{sos}}]$$

Derivation sequence and execution

Definition 18 (Derivation sequences)

$\gamma_1, \gamma_1, \ldots, \gamma_k$ or $\gamma_1, \gamma_2, \ldots$

where:

- $\gamma_i \Rightarrow \gamma_{i+1}$, for $i \geq 1$, and
- $\gamma_k \not\Rightarrow$

Definition 19 (Execution of a statement)

The execution(s) of a statement S on a state σ is/are the maximal derivation sequence(s) starting with the initial configuration (S, σ).

Definition 20 (The $\mathcal{S}_{\rm sos}$ semantic function)

$$\mathcal{S}_{\rm sos}[S]\sigma = \begin{cases} \sigma' & \text{if } (S,\sigma) \Rightarrow^* \sigma' \\ \text{undef} & \text{otherwise} \end{cases}$$

Properties with respect to **While**

Lemma 1 (Composing statements)

*For every statement $S_1, S_2 \in$ **Stm**, state $\sigma \in$ **State**, and $k \in \mathbb{N}$:*

$$(S_1, \sigma) \Rightarrow^k \sigma' \text{ implies } (S_1; S_2, \sigma) \Rightarrow^k (S_2; \sigma')$$

(Executing a statement is not influenced by the sequentially composed statement – S_2 in the lemma)

Lemma 2 (Decomposing computations in SOS)

*For every statement $S_1, S_2 \in$ **Stm**, state $\sigma \in$ **State**, and $k \in \mathbb{N}$:*

$(S_1; S_2, \sigma) \Rightarrow^k \sigma''$ *implies*
there exist σ' *and* k_1 *s.t.* $(S_1, \sigma) \Rightarrow^{k_1} \sigma'$ *and* $(S_2, \sigma') \Rightarrow^{k-k_1} \sigma''$.

Theorem: equivalence of NOS and SOS for **While**

For every statement S in **Stm**: $\mathcal{S}_{\rm ns}[S] = \mathcal{S}_{\rm sos}[S]$.

Properties with respect to extensions of **While**

SOS distinguishes between blocking and non-termination.

Natural/structural operational semantics and looping

- In NOS, non-determinism "hides" looping, if possible.
- In SOS, non-determinism does not "hide" looping.

Natural vs Structural (operational) semantics and interleaving

- Natural semantics:
  - does not allow to express interleaving
  - executions of atomic constituents are atomic
- Structural semantics:
  - allows to express interleaving
  - we focus on the small steps of computations

Outline - Notations and main results in **While**, **Block**, **Proc**, and the various semantics

Syntax

Semantic Analysis (typing)

Natural Operational Semantics (NOS)

Structural Operational Semantics

Axiomatic Semantics

Definitions

Definition 21 (Hoare Triple - Assertion)

$\{P\}\ S\ \{Q\}$, with S: statement, P: pre-condition, Q: post-condition.
A logical variable is a variable not appearing in the program.

Definition 22 (Predicate)

A predicate is a function from **State** to $\{\mathbf{tt}, \mathbf{ff}\}$ denoted using the syntactic category **Bexp** extended with logical variables.

Definition 23 (Predicates `True` and `False`)

Predicates `True` and `False` hold on all and no states, respectively.

Boolean operators

- $P_1 \wedge P_2$ denotes the function associating $P_1(\sigma)$ and $P_2(\sigma)$,
- $P_1 \vee P_2$ denotes the function associating $P_1(\sigma)$ or $P_2(\sigma)$,
- $\neg P$: denotes the function associating not $(P(\sigma))$,
- $P_1 \Rightarrow P_2$ denotes the function associating $P_1(\sigma)$ implies $P_2(\sigma)$,

to any state $\sigma \in$ **State**.

Definition 24 ((Syntactic) substitution)

For $x \in$ **Var** and $a \in$ **Aexp**, $P[a/x]$ is a predicate obtained by replacing each occurrence of x by a in P.

The complete inference system

Rule name	original	generalized
Skip	$\{P\}$ skip $\{P\}$	$\dfrac{P \implies Q}{\{P\}\ \text{skip}\ \{Q\}}$
Assignment	$\{P[a/x]\}\ x := a\ \{P\}$	$\dfrac{Q \implies P[a/x]}{\{Q\}\ x := a\ \{P\}}$
Sequential	$\dfrac{\{P\}\ S_1\ \{Q\} \quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$	$\dfrac{\{P\}\ S_1\ \{R_1\} \quad R_1 \implies R_2 \quad \{R_2\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$
Conditional	$\dfrac{\{b \wedge P\}\ S_1\ \{Q\} \quad \{\neg b \wedge P\}\ S_2\ \{Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{Q\}}$	
Iterative	$\dfrac{\{b \wedge P\}\ S\ \{P\}}{\{P\}\ \text{while } b \text{ do } S \text{ od}\ \{\neg b \wedge P\}}$	$\dfrac{P \implies I \quad \{b \wedge I\}\ S\ \{P\} \quad I \wedge \neg b \implies Q}{\{P\}\ \text{while } b \text{ do } S \text{ od}\ \{Q\}}$
Consequence	If $P \Rightarrow P'$ and $Q' \Rightarrow Q$, then: $\dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$	

When inferring $\{P\}\ S\ \{Q\}$ (with rules and axioms), we note: $\vdash_p \{P\}\ S\ \{Q\}$.

Properties of the semantics

Definition 25 (Semantic equivalence between programs)

S_1 and S_2 are provably equivalent according to the axiomatic semantics (for partial correctness) if

- for all pre-conditions P,
- for all post-conditions Q:

Definition 26 (Validity of a Hoare triple)

Triple $\{P\}\ S\ \{Q\}$ is valid, noted

$$\models_p \{P\}\ S\ \{Q\}$$

iff for all states $\sigma, \sigma' \in$ **State**:

- if $P(\sigma)$ and $(S, \sigma) \rightarrow \sigma'$
- then $Q(\sigma')$.

We say that S is partially correct wrt. P and Q.

Soundness (We can infer *only* valid triples)

If $\vdash_p \{P\}\ S\ \{Q\}$ then $\models_p \{P\}\ S\ \{Q\}$

Completeness (We can infer *all* valid triples)

If $\models_p \{P\}\ S\ \{Q\}$ then $\vdash_p \{P\}\ S\ \{Q\}$