

Semantics and Verification of Software

Lecture 1: Introduction

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw11/>

Winter Semester 2011/12

- 1 Preliminaries
- 2 Introduction
- 3 The Imperative Model Language WHILE

- Lectures: **Thomas Noll**
 - Lehrstuhl für Informatik 2, Room 4211
 - E-mail noll@cs.rwth-aachen.de
 - Phone (0241)80-21213
- Exercise classes:
 - **Christina Jansen** (christina.jansen@cs.rwth-aachen.de)
 - **Sabrina von Styp** (sabrina.von-styp@cs.rwth-aachen.de)

- Master/Diplom programme **Informatik**
 - Theoretische Informatik
 - Vertiefungsfach *Formale Methoden, Programmiersprachen und Softwarevalidierung* (Diplom)
- Master programme **Software Systems Engineering**
 - Theoretical CS
 - Specialization in *Formal Methods, Programming Languages and Software Validation*
- In general:
 - interest in **formal models** for programming languages
 - application of **mathematical reasoning methods**
- Expected: basic knowledge in
 - essential concepts of **imperative programming languages**
 - **formal languages** and **automata theory**
 - **mathematical logic**

- Schedule:
 - **Lecture** Wed 10:00–11:30 AH 2 (starting Oct 12)
 - **Lecture** Thu 13:30–15:00 AH 1 (starting Oct 13)
 - **Exercise class** Tue 08:15–09:45 AH 4 **or** 11:45–13:15 AH 6 (starting Oct 25)
- Irregular lecture dates – checkout web page!
- 1st assignment sheet: next Tuesday (Oct 18)
- Work on assignments in **groups of three**
- **Examination** (6 ECTS credits):
 - oral
 - date by agreement
- Admission requires **at least 50% of the points in the exercises**
- Solutions to exercises and exam in **English or German**

- 1 Preliminaries
- 2 Introduction
- 3 The Imperative Model Language WHILE

Aspects of Programming Languages

Syntax: “How does a program look like?”

(hierarchical composition of programs from structural components)

⇒ *Compiler Construction*

Semantics: “What does this program mean?”

(output/behavior in dependence of input/environment)

⇒ *This course*

Pragmatics:

- **length** and **understandability** of programs
- **learnability** of programming language
- **appropriateness** for specific applications, ...

⇒ *Software Engineering*

Historic development:

- **Formal syntax** since 1960s (LL/LR parsing); semantics defined by compiler/interpreter
- **Formal semantics** since 1970s (operational/denotational/axiomatic)

The Semantics of “Semantics”

Originally: study of meaning of symbols (linguistics)

Semantics of a program: meaning of a concrete program (I/O mapping, behavior, ...)

Semantics of a programming language: mapping of each (syntactically correct) program of a concrete programming language to its meaning

Semantics of software: various techniques for defining the semantics of diverse programming languages

Example 1.1

- ① How often will the following loop be traversed?

```
for i := 2 to 1 do ...
```

FORTRAN IV: once

PASCAL: never

- ② What if `p = nil` in the following program?

```
while p <> nil and p^.key < val do ...
```

Pascal: strict boolean operations ⚡

Modula: non-strict boolean operations ✓

Motivation for Rigorous Formal Treatment II

- Support for **development** of
 - new **programming languages**: missing details, ambiguities and inconsistencies can be recognized
 - **compilers**: automatic compiler generation from appropriately defined semantics
 - **programs**: exact understanding of semantics avoids uncertainties in the implementation of algorithms
- Support for **correctness proofs** of
 - **programs**: comparison of program semantics with desired behaviour (e.g., termination properties, absence of deadlocks, ...)
 - **compilers**:

programming language	$\xrightarrow{\text{compiler}}$	machine code
semantics $\downarrow$		$\downarrow$ (simple) semantics
meaning	$\stackrel{?}{=}$	meaning
 - **optimizing transformations**:

code	$\xrightarrow{\text{optimization}}$	code
semantics $\downarrow$		$\downarrow$ semantics
meaning	$\stackrel{?}{=}$	meaning

(Complementary) Kinds of Formal Semantics

Operational semantics: describes **computation** of the program on some (very) abstract machine (G. Plotkin)

- example:
$$\text{(seq)} \frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''}$$
- application: **implementation** of programming languages (compilers, interpreters, ...)

Denotational semantics: mathematical definition of **input/output relation** of the program by induction on its syntactic structure (D. Scott, C. Strachey)

- example:
$$\begin{aligned} \mathcal{C}[\![\cdot]\!] &: Cmd \rightarrow (\Sigma \dashrightarrow \Sigma) \\ \mathcal{C}[\![c_1; c_2]\!] &:= \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!] \end{aligned}$$
- application: program **analysis**

Axiomatic semantics: formalization of special properties of programs by **logical formulae** (assertions/proof rules; R. Floyd, T. Hoare)

- example:
$$\text{(seq)} \frac{\{A\} c_1 \{C\} \quad \{C\} c_2 \{B\}}{\{A\} c_1; c_2 \{B\}}$$
- application: program **verification**

- ① The imperative model language WHILE
- ② Operational semantics of WHILE
- ③ Denotational semantics of WHILE
- ④ Equivalence of operational and denotational semantics
- ⑤ Axiomatic semantics of WHILE
- ⑥ Extensions: procedures and dynamic data structures
- ⑦ Applications: compiler correctness etc.

(also see the collection [“Handapparat”] at the CS Library)

- Formal semantics:
 - G. Winskel: *The Formal Semantics of Programming Languages*, The MIT Press, 1996
- Compiler correctness
 - H.R. Nielson, F. Nielson: *Semantics with Applications: A Formal Introduction*, Wiley, 1992

- 1 Preliminaries
- 2 Introduction
- 3 The Imperative Model Language WHILE

WHILE: simple imperative programming language without procedures or advanced data structures

Syntactic categories:

Category	Domain	Meta variable
Numbers	$\mathbb{Z} = \{0, 1, -1, \dots\}$	z
Truth values	$\mathbb{B} = \{\text{true}, \text{false}\}$	t
Variables	$Var = \{x, y, \dots\}$	x
Arithmetic expressions	$AExp$ (next slide)	a
Boolean expressions	$BExp$ (next slide)	b
Commands (statements)	Cmd (next slide)	c

Definition 1.2 (Syntax of WHILE)

The **syntax of WHILE Programs** is defined by the following context-free grammar:

$$a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp$$
$$b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp$$
$$c ::= \text{skip} \mid x := a \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \in Cmd$$

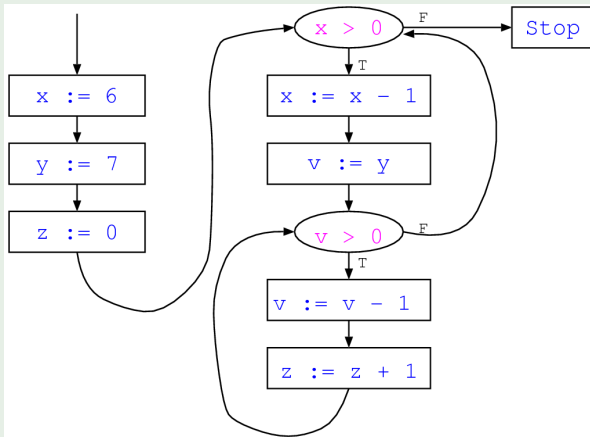
Remarks: we assume that

- the syntax of numbers, truth values and variables is predefined (i.e., no “lexical analysis”)
- the syntax of ambiguous constructs is uniquely determined (by brackets, priorities, or indentation)

A WHILE Program and its Flow Diagram

Example 1.3

```
x := 6;  
y := 7;  
z := 0;  
while x > 0 do  
  x := x - 1;  
  v := y;  
  while v > 0 do  
    v := v - 1;  
    z := z + 1
```



Effect: $z := x * y = 42$