Semantics and Verification of Software

Lecture 1: Introduction

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

Outline

People

- 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`)

Target Audience

- 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*

Target Audience

- 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

Organization

- Schedule:
  - Lecture Wed 10:00–11:30 AH 6 (starting April 21)
  - Lecture Thu 15:00–16:30 AH 5 (starting April 15)
  - Exercise class Mon 10:00-11:30 AH 2 (starting April 26)
- 1st assignment sheet: next Monday (April 19)
- Work on assignments in groups of three
- Examination (8 [6?] ECTS credit points):
  - oral
  - date by agreement
- Admission requires at least 50% of the points in the exercises
- Solutions to exercises and exam in English or German

Outline

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?"
(execution evokes state transformations of an [abstract] machine)
⇒ This course

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

⇒ Software Engineering

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?"
(execution evokes state transformations of an [abstract] machine)
⇒ 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)

Motivation for Rigorous Formal Treatment I

Example 1.1

1. How often will the following loop be traversed?

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

FORTRAN IV: once
PASCAL: never

Motivation for Rigorous Formal Treatment I

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 $\surd$

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

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)
  - compilers:

$$\begin{array}{rcl} \text{programming language} & \xrightarrow{\text{compiler}} & \text{machine code} \\ \text{semantics} \downarrow & & \downarrow \text{(simple) semantics} \\ \text{meaning} & \stackrel{?}{=} & \text{meaning} \end{array}$$

  - optimizing transformations:

$$\begin{array}{rcl} \text{code} & \xrightarrow{\text{optimization}} & \text{code} \\ \text{semantics} \downarrow & & \downarrow \text{semantics} \\ \text{meaning} & \stackrel{?}{=} & \text{meaning} \end{array}$$

Kinds of Formal Semantics

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

Kinds of Formal Semantics

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

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

Kinds of Formal Semantics

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

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

Axiomatic semantics: formalization of special properties of the program by logical formulae (assertions and proof rules; R. Floyd, T. Hoare)

Overview of the Course

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

Literature

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

- Formal semantics:
  - G. Winskel: *The Formal Semantics of Programming Languages*, The MIT Press, 1996
- Dataflow analysis:
  - F. Nielson, H.R. Nielson, C. Hankin: *Principles of Program Analysis*, 2nd ed., Springer, 2005
- Compiler correctness
  - H.R. Nielson, F. Nielson: *Semantics with Applications: A Formal Introduction*, Wiley, 1992

Outline

Syntactic Categories

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

Syntactic Categories

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

Syntactic categories:

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

Syntax of WHILE Programs

Definition 1.2 (Syntax of WHILE)

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

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

Syntax of WHILE Programs

Definition 1.2 (Syntax of WHILE)

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

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

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

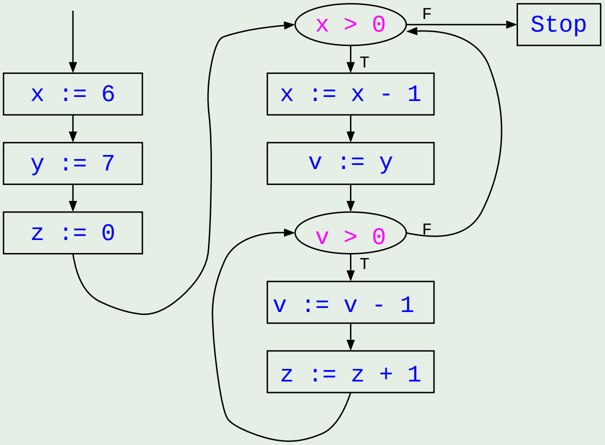
Example 1.3

```
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
```

A WHILE Program and its Flow Diagram

Example 1.3

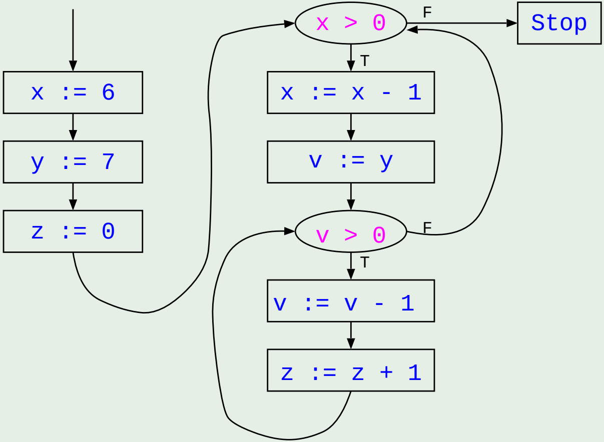
```
x := 6;
y := 7;
z := 0;
while x > 0 do
  x := x - 1;
  v := y;
  while v > 0 do
    v := v - 1;
    z := z + 1
```



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