Semantics and Verification of Software

Lecture 1: Introduction

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw/

Summer semester 2007

Outline

People

- Lectures: Thomas Noll
  - Lehrstuhl für Informatik 2, Room 4211
  - E–mail `noll@cs.rwth-aachen.de`
  - Phone (0241)80–21213
- Exercise classes: Daniel Willems
  - E–mail `willems@cs.rwth-aachen.de`
- Assistant: Lars Helge Haß
  - E–mail (`LarsHass@gmx.de`)

Target Audience

- Diploma programme (Informatik)
  - Theoretische Informatik
  - Vertiefungsfach Formale Methoden, Programmiersprachen und Softwarevalidierung
- Master programme (Software Systems Engineering)
  - Theoretical CS
  - Specialization 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 and object–oriented programming languages
  - formal languages and automata theory
  - mathematical logic

Organization

- Schedule:
  - Lecture Tue 11:45–13:15 AH 3 (starting April 10)
  - Lecture Fri 8:15–9:45 AH 2 (starting April 13)
  - Exercise class Wed 13:30-15:00 AH 2 (starting April 18)
- 1st assignment sheet: Friday
- Work on assignments in groups of three
- Examination (8 ECTS credit points):
  written or oral (depending on number of candidates);
  date: middle of July
- 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)

Semantics: "What does this program mean?"
(execution evokes state transformations of an [abstract] machine)

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

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 is the following loop 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 avois uncertainties in the implementation of algorithms
- Support for correctness proofs of
  - programs: comparison of program semantics with desired behaviour (e.g., termination properties)
  - compilers:

$$\begin{array}{ccc} \text{programming language} & \overset{\text{compiler}}{\longrightarrow} & \text{machine code} \\ \text{semantics} \downarrow & & \downarrow \text{(simple) semantics} \\ \text{meaning} & \overset{?}{=} & \text{meaning} \end{array}$$

  - optimizing transformations:

$$\begin{array}{ccc} \text{code} & \overset{\text{optimization}}{\longrightarrow} & \text{code} \\ \text{semantics} \downarrow & & \downarrow \text{semantics} \\ \text{meaning} & \overset{?}{=} & \text{meaning} \end{array}$$

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. Dataflow analysis
8. Abstract interpretation and abstraction refinement
9. Extensions: procedures and dynamic data structures

Literature

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

- Formal semantics:
  - G. Winskel: *The Formal Semantics of Programming Languages*, The MIT Press, 1996
  - H.R. Nielson, F. Nielson: *Semantics with Applications: A Formal Introduction*, Wiley, 1992
  - E. Fehr: *Semantik von Programmiersprachen*, Springer, 1989
- Dataflow analysis and abstract interpretation:
  - F. Nielson, H.R. Nielson, C. Hankin: *Principles of Program Analysis*, 2nd ed., Springer, 2005

Outline

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	$AExp$	a
Boolean expressions	$BExp$	b
Commands (statements)	Cmd	c

Syntax of WHILE Programs

Definition 1.2 (Syntax of WHILE)

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

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

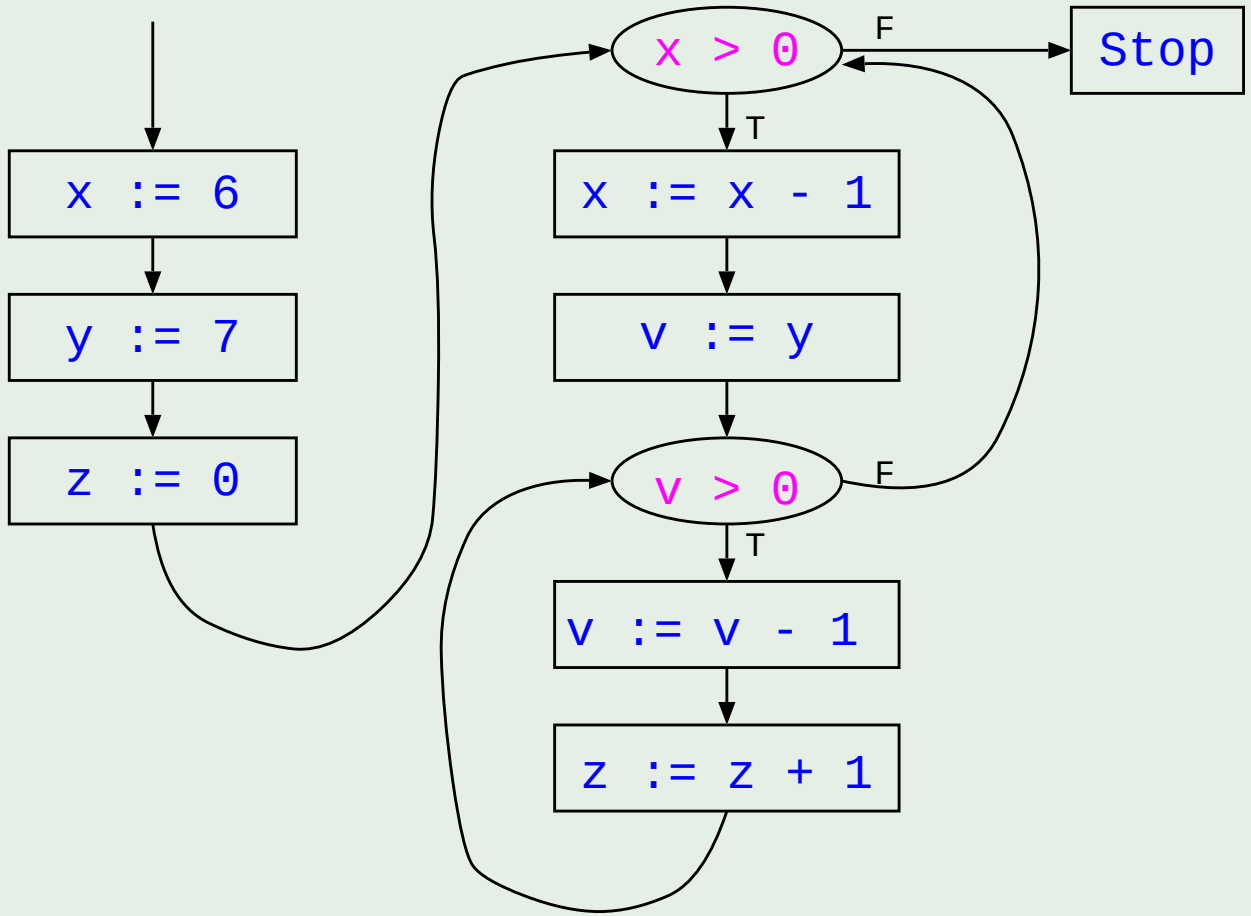
Remarks: we assume that

- the syntax of numbers, truth values and variables is given (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`