

Theory of Computer Science

F1. LOOP-Computability

Gabriele Röger

University of Basel

May 15, 2019

Theory of Computer Science

May 15, 2019 — F1. LOOP-Computability

F1.1 Introduction

F1.2 LOOP Programs

F1.3 Syntactic Sugar

F1.4 Summary

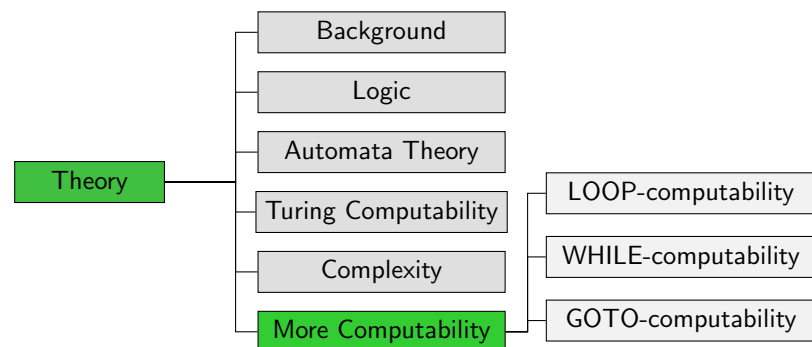
Overview: Course

contents of this course:

- A. background ✓
 - ▷ mathematical foundations and proof techniques
- B. logic ✓
 - ▷ How can knowledge be represented?
 - How can reasoning be automated?
- C. automata theory and formal languages ✓
 - ▷ What is a computation?
- D. Turing computability ✓
 - ▷ What can be computed at all?
- E. complexity theory ✓
 - ▷ What can be computed efficiently?
- F. more computability theory
 - ▷ Other models of computability

F1.1 Introduction

Course Overview



Formal Models of Computation: LOOP/WHILE/GOTO

Formal Models of Computation

- ▶ Turing machines
- ▶ LOOP, WHILE and GOTO programs
- ▶ (primitive recursive and μ -recursive functions)

In this and the following chapter we get to know three simple models of computation (programming languages) and compare their power to Turing machines:

- ▶ LOOP programs \rightsquigarrow today
- ▶ WHILE programs \rightsquigarrow F2
- ▶ GOTO programs \rightsquigarrow F3

LOOP, WHILE and GOTO Programs: Basic Concepts

- ▶ LOOP, WHILE and GOTO programs are structured like programs in (simple) “traditional” programming languages
- ▶ use finitely many variables from the set $\{x_0, x_1, x_2, \dots\}$ that can take on values in \mathbb{N}_0
- ▶ differ from each other in the allowed “statements”

F1.2 LOOP Programs

LOOP Programs: Syntax

Definition (LOOP Program)

LOOP programs are inductively defined as follows:

- ▶ $x_i := x_j + c$ is a LOOP program for every $i, j, c \in \mathbb{N}_0$ (**addition**)
- ▶ $x_i := x_j - c$ is a LOOP program for every $i, j, c \in \mathbb{N}_0$ (**modified subtraction**)
- ▶ If P_1 and P_2 are LOOP programs, then so is $P_1; P_2$ (**composition**)
- ▶ If P is a LOOP program, then so is **LOOP** x_i **DO** P **END** for every $i \in \mathbb{N}_0$ (**LOOP loop**)

German: LOOP-Programm, Addition, modifizierte Subtraktion, Komposition, LOOP-Schleife

LOOP Programs: Semantics

Definition (Semantics of LOOP Programs)

A LOOP program **computes** a k -ary function $f : \mathbb{N}_0^k \rightarrow \mathbb{N}_0$. The computation of $f(n_1, \dots, n_k)$ works as follows:

- ① Initially, the variables x_1, \dots, x_k hold the values n_1, \dots, n_k . All other variables hold the value 0.
- ② During computation, the program modifies the variables as described on the following slides.
- ③ The result of the computation ($f(n_1, \dots, n_k)$) is the value of x_0 after the execution of the program.

German: P berechnet f

LOOP Programs: Semantics

Definition (Semantics of LOOP Programs)

effect of $x_i := x_j + c$:

- ▶ The variable x_i is assigned the current value of x_j plus c .
- ▶ All other variables retain their value.

LOOP Programs: Semantics

Definition (Semantics of LOOP Programs)

effect of $x_i := x_j - c$:

- ▶ The variable x_i is assigned the current value of x_j minus c if this value is non-negative.
- ▶ Otherwise x_i is assigned the value 0.
- ▶ All other variables retain their value.

LOOP Programs: Semantics

Definition (Semantics of LOOP Programs)

effect of $P_1; P_2$:

- ▶ First, execute P_1 .
Then, execute P_2 (on the modified variable values).

LOOP Programs: Semantics

Definition (Semantics of LOOP Programs)

effect of `LOOP x_i DO P END`:

- ▶ Let m be the value of variable x_i at the start of execution.
- ▶ The program P is executed m times in sequence.

LOOP-Computable Functions

Definition (LOOP-Computable)

A function $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ is called **LOOP-computable** if a LOOP program that computes f exists.

German: f ist LOOP-berechenbar

Note: non-total functions are never LOOP-computable.
(Why not?)

LOOP Programs: Example

Example (LOOP program for $f(x_1, x_2)$)

```
LOOP  $x_1$  DO
  LOOP  $x_2$  DO
     $x_0 := x_0 + 1$ 
  END
END
```

Which (binary) function does this program compute?

F1.3 Syntactic Sugar

Syntactic Sugar or Essential Feature?

- ▶ We investigate the power of programming languages and other computation formalisms.
 - ▶ **Rich** language features help when writing complex programs.
 - ▶ **Minimalistic** formalisms are useful for proving statements over **all** programs.
- ↔ conflict of interest!

Idea:

- ▶ Use **minimalistic core** for proofs.
- ▶ Use **syntactic sugar** when writing programs.

German: syntaktischer Zucker

Example: Syntactic Sugar

Example (syntactic sugar)

We propose five new syntax constructs (with the obvious semantics):

- ▶ $x_i := x_j$ for $i, j \in \mathbb{N}_0$
- ▶ $x_i := c$ for $i, c \in \mathbb{N}_0$
- ▶ $x_i := x_j + x_k$ for $i, j, k \in \mathbb{N}_0$
- ▶ **IF** $x_i \neq 0$ **THEN** P **END** for $i \in \mathbb{N}_0$
- ▶ **IF** $x_i = c$ **THEN** P **END** for $i, c \in \mathbb{N}_0$

Can we simulate these with the existing constructs?

Example: Syntactic Sugar

Example (syntactic sugar)

$x_i := x_j$ for $i, j \in \mathbb{N}_0$

Simple abbreviation for $x_i := x_j + 0$.

Example: Syntactic Sugar

Example (syntactic sugar)

$x_i := c$ for $i, c \in \mathbb{N}_0$

Simple abbreviation for $x_i := x_j + c$,
where x_j is a fresh variable, i.e., an otherwise unused variable
that is not an input variable.
(Thus x_j must always have the value 0 in all executions.)

Example: Syntactic Sugar

Example (syntactic sugar)

$x_i := x_j + x_k$ for $i, j, k \in \mathbb{N}_0$

Abbreviation for:

```
x_i := x_j;
LOOP x_k DO
  x_i := x_i + 1
END
```

Analogously we will also use the following:

- ▶ $x_i := x_j - x_k$
- ▶ $x_i := x_j + x_k - c - x_m + d$
- ▶ etc.

Example: Syntactic Sugar

Example (syntactic sugar)

IF $x_i \neq 0$ THEN P END for $i \in \mathbb{N}_0$

Abbreviation for:

```
x_j := 0;
LOOP x_i DO
  x_j := 1
END;
LOOP x_j DO
  P
END
```

where x_j is a fresh variable.

Example: Syntactic Sugar

Example (syntactic sugar)

IF $x_i = c$ THEN P END for $i, c \in \mathbb{N}_0$

Abbreviation for:

```
x_j := 1;
x_k := x_i - c;
IF x_k \neq 0 THEN x_j := 0 END;
x_k := c - x_j;
IF x_k \neq 0 THEN x_j := 0 END;
IF x_j \neq 0 THEN
  P
END
```

where x_j and x_k are fresh variables.

Can We Be More Minimalistic?

- ▶ We see that some common structural elements such as IF statements are unnecessary because they are syntactic sugar.
- ▶ Can we make LOOP programs even more minimalistic than in our definition?

Simplification 1

Instead of $x_i := x_j + c$ and $x_i := x_j - c$ it suffices to only allow the constructs

- ▶ $x_i := x_j$,
- ▶ $x_i := x_j + 1$ and
- ▶ $x_i := x_j - 1$.

Why?

Can We Be More Minimalistic?

- ▶ We see that some common structural elements such as IF statements are unnecessary because they are syntactic sugar.
- ▶ Can we make LOOP programs even more minimalistic than in our definition?

Simplification 2

The construct $x_i := x_j$ can be omitted because it can be simulated with other constructs:

```

LOOP  $x_i$  DO
   $x_j := x_j - 1$ 
END;
LOOP  $x_j$  DO
   $x_i := x_i + 1$ 
END
  
```

F1.4 Summary

Summary

LOOP programs

- ▶ new model of computation for numerical functions
- ▶ closer to typical programming languages than Turing machines