Theory of Computer Science

F1. LOOP-Computability

Gabriele Röger

University of Basel

May 27, 2020

Theory of Computer Science
May 27, 2020 — F1. LOOP-Computability

F1.1 Introduction

F1.2 LOOP Programs

F1.3 WHILE Programs

F1.4 GOTO Programs

F1.5 Comparison

F1.6 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?

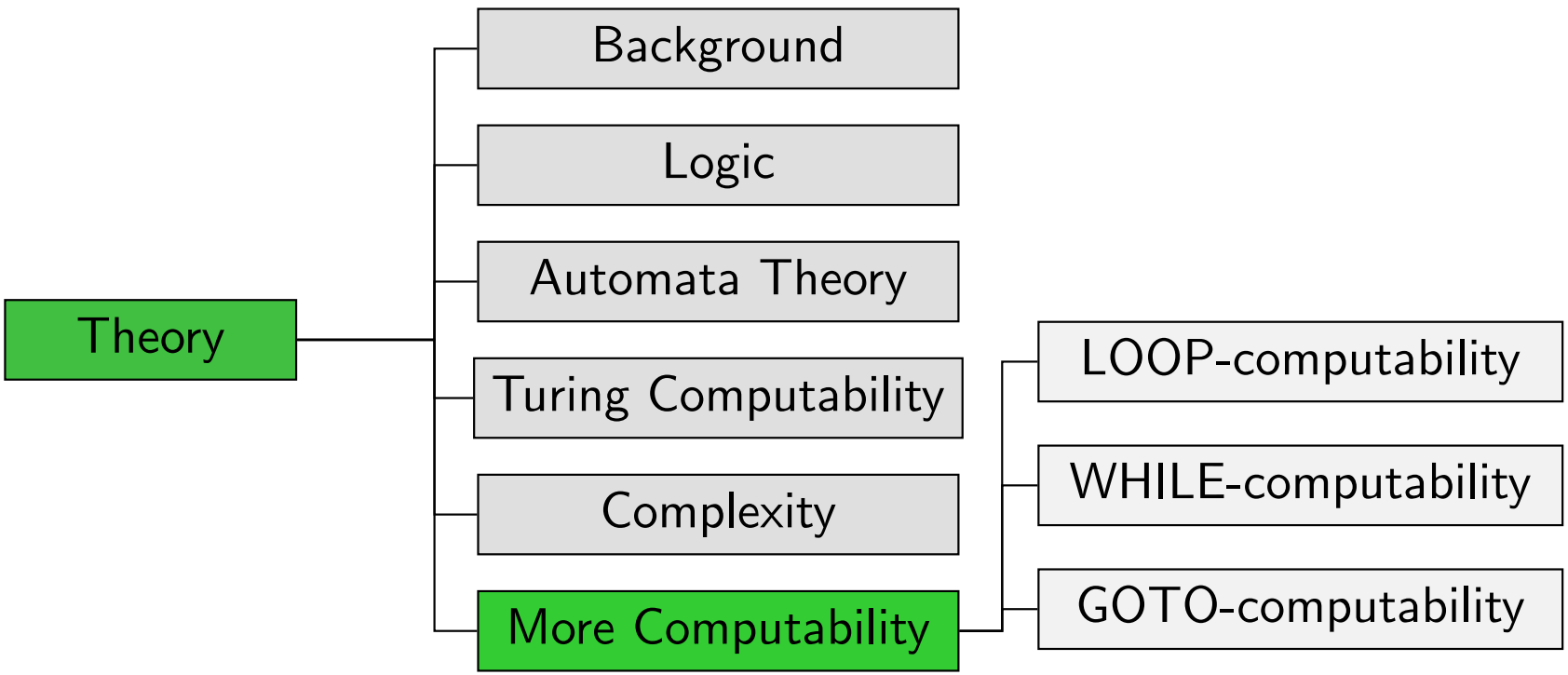
D. Turing computability ✓
   - ▷ What can be computed at all?

E. complexity theory ✓
   - ▷ What can be computed efficiently?

F. more computability theory
   - ▷ Other models of computability

F1.1 Introduction

Course Overview



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 $f : \mathbb{N}_0^k \to \mathbb{N}_0$. The computation of $f(n_1, \ldots, n_k)$ works as follows:

1. Initially, the variables $x_1, \ldots, x_k$ hold the values $n_1, \ldots, n_k$. 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 $(f(n_1, \ldots, n_k))$ 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 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?

Note: A LOOP-program cannot compute a non-total function. (Why not?)

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.

$\rightsquigarrow$ conflict of interest!

Idea:

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

German: syntaktischer Zucker

Example: Syntactic Sugar

Example (syntactic sugar)

The following five new syntax constructs (with the obvious semantics) can be simulated with the existing constructs.

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

F1.3 WHILE Programs

WHILE Programs: Syntax

Definition (WHILE Program)

WHILE programs are inductively defined as follows:

- $x_i := x_j + c$ is a WHILE program
  for every $i, j, c \in \mathbb{N}_0$ (addition)
- $x_i := x_j - c$ is a WHILE program
  for every $i, j, c \in \mathbb{N}_0$ (modified subtraction)
- If P_1 and P_2 are WHILE programs,
  then so is $P_1;P_2$ (composition)
- If P is a WHILE program, then so is
  WHILE $x_i \neq 0$ DO P END for every $i \in \mathbb{N}_0$ (WHILE loop)

German: WHILE-Programm, WHILE-Schleife

WHILE Programs: Semantics

Definition (Semantics of WHILE Programs)

The semantics of WHILE programs is defined exactly as for LOOP programs.

effect of WHILE $x_i \neq 0$ DO P END:

- If x_i holds the value 0, program execution finishes.
- Otherwise execute P.
- Repeat these steps until execution finishes (potentially infinitely often).

WHILE-Program: Example

Example

WHILE $x_1 \neq 0$ **DO**
$x_1 := x_1 - x_2;$
$x_0 := x_0 + 1$
END

What function does this program compute?

Syntactic Sugar

As we can simulate LOOP loops from LOOP programs with WHILE programs, we can use all syntactic sugar we have seen for LOOP programs in WHILE programs e.g.

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

F1.4 GOTO Programs

GOTO Programs: Syntax

Definition (GOTO Program)

A GOTO program is given by a finite sequence
$L_1 : A_1, L_2 : A_2, \ldots, L_n : A_n$
of labels and statements.

Statements are of the following form:

- $x_i := x_j + c$ for every $i, j, c \in \mathbb{N}_0$ (addition)
- $x_i := x_j - c$ for every $i, j, c \in \mathbb{N}_0$ (modified subtraction)
- HALT (end of program)
- GOTO L_j for $1 \leq j \leq n$ (jump)
- IF $x_i = c$ THEN GOTO L_j for $i, c \in \mathbb{N}_0$, $1 \leq j \leq n$ (conditional jump)

German: GOTO-Programm, Marken, Anweisungen, Programmende, Sprung, bedingter Sprung

GOTO Programs: Semantics

Definition (Semantics of GOTO Programs)

- Input, output and variables work exactly as in LOOP and WHILE programs.
- Addition and modified subtraction work exactly as in LOOP and WHILE programs.
- Execution begins with the statement A_1.
- After executing A_i, the statement A_{i+1} is executed. (If $i = n$, execution finishes.)
- exceptions to the previous rule:
  - HALT stops the execution of the program.
  - After GOTO L_j execution continues with statement A_j.
  - After IF $x_i = c$ THEN GOTO L_j execution continues with A_j if variable x_i currently holds the value c.

F1.5 Comparison

LOOP/WHILE/GOTO-Computable Functions

Definition (LOOP-/WHILE-/GOTO-Computable)

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

Result

Corollary

Let $f : \mathbb{N}_0^k \to_p \mathbb{N}_0$ be a function.

The following statements are equivalent:

- *f is Turing-computable.*
- *f is WHILE-computable.*
- *f is GOTO-computable.*

Moreover:

- *Every LOOP-computable function is Turing-/WHILE-/GOTO-computable.*
- *The converse is not true in general.*

F1.6 Summary

Summary

- Turing machines, WHILE and GOTO programs are equally powerful.
  - Whenever we said "Turing-computable" or "computable" in parts D or E, we could equally have said "WHILE-computable" or "GOTO-computable".
- LOOP programs are strictly less powerful.