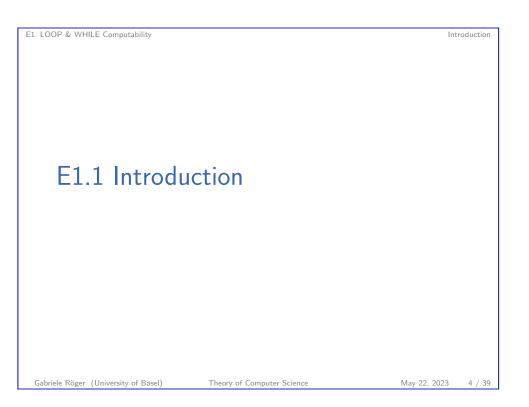
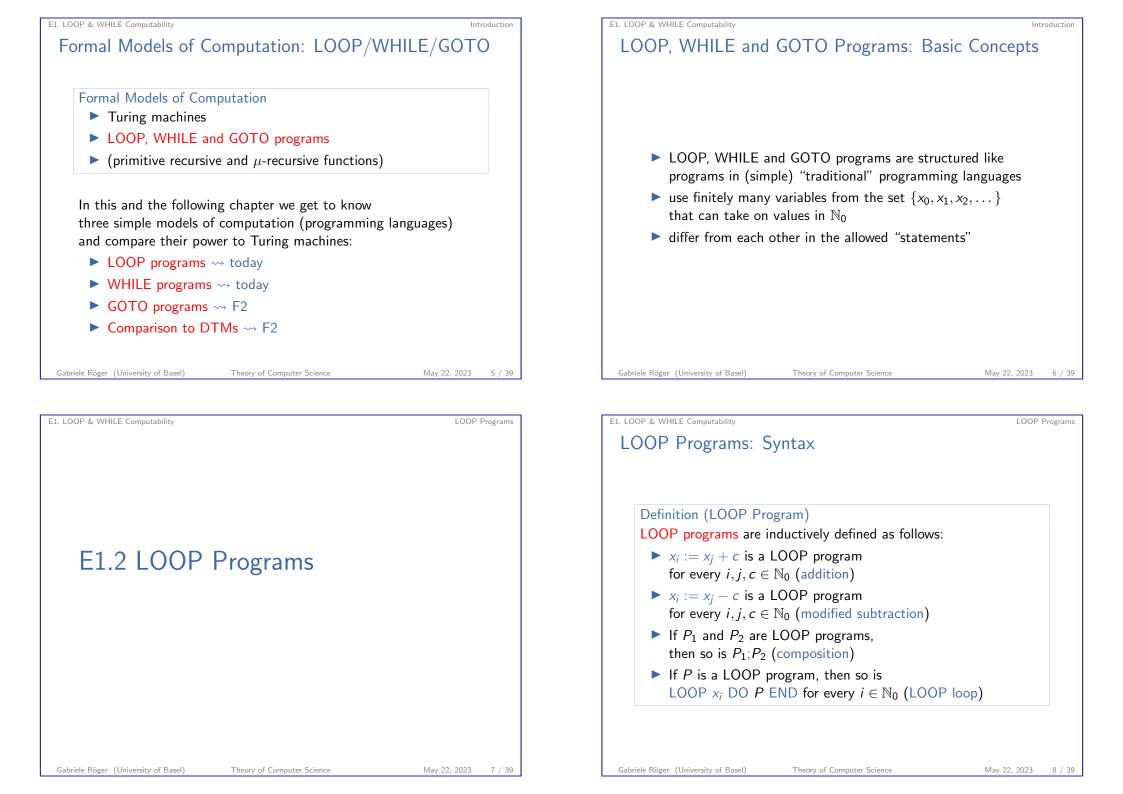
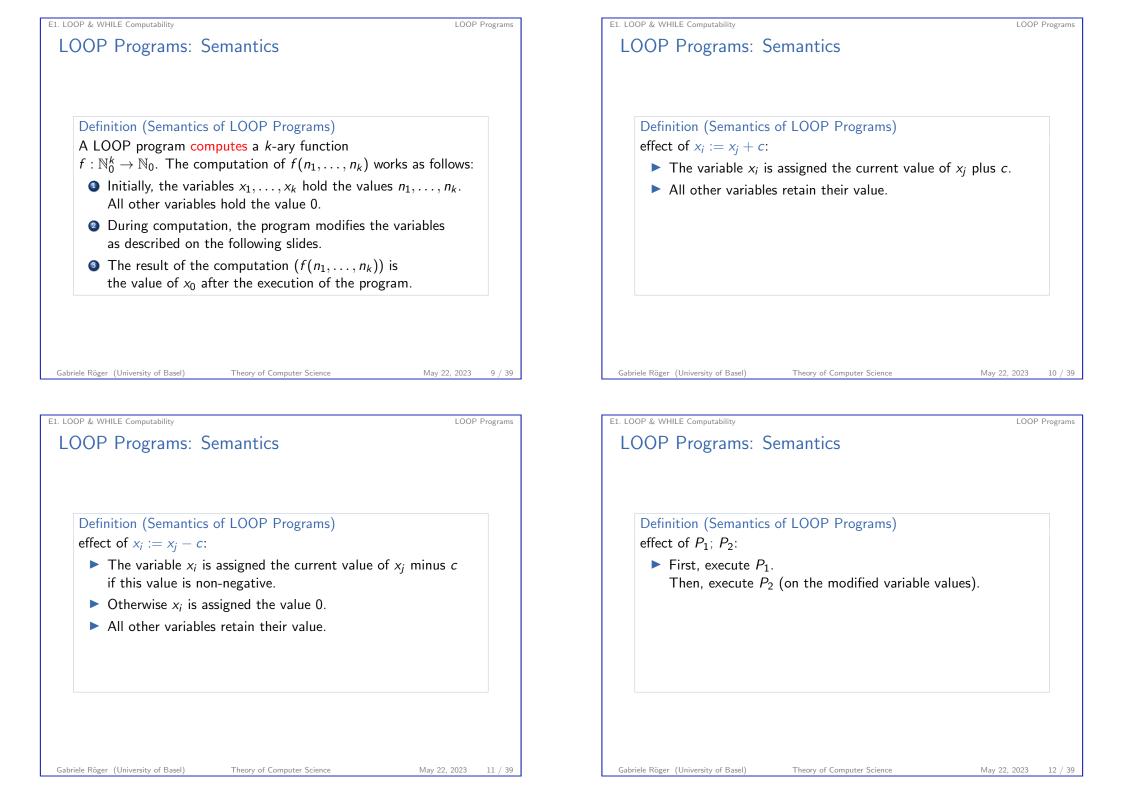


Theory of Computer Science May 22, 2023 — E1. LOOP & WHILE Computability E1.1 Introduction E1.2 LOOP Programs E1.3 WHILE Programs E1.4 WHILE vs. LOOP E1.5 Summary







LOOP Programs: Semantics

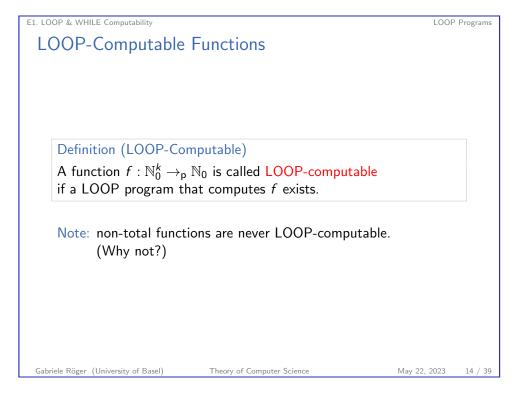
Definition (Semantics of LOOP Programs) effect of LOOP x_i DO P END:

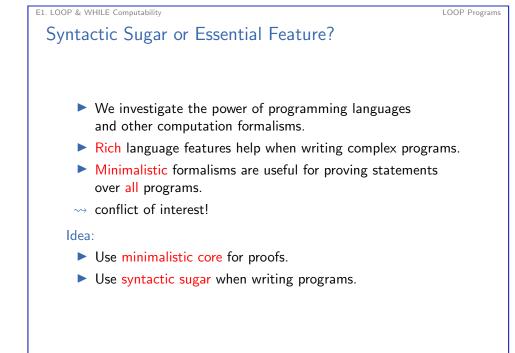
- Let *m* be the value of variable x_i at the start of execution.
- ▶ The program *P* is executed *m* times in sequence.

Gabriele Röger (University of Basel)

Theory of Computer Science

E1. LOOP & WHILE Computability 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?



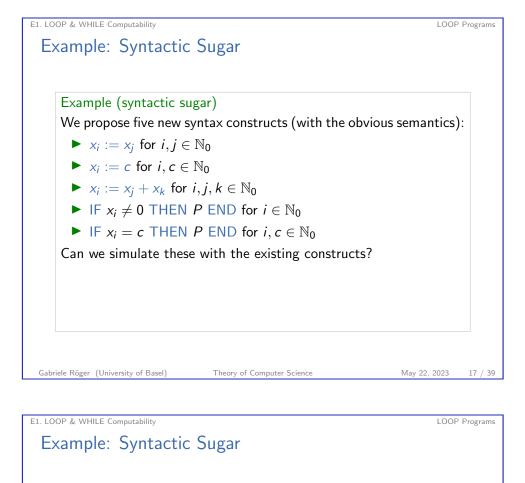


LOOP Programs

May 22, 2023

13 / 39

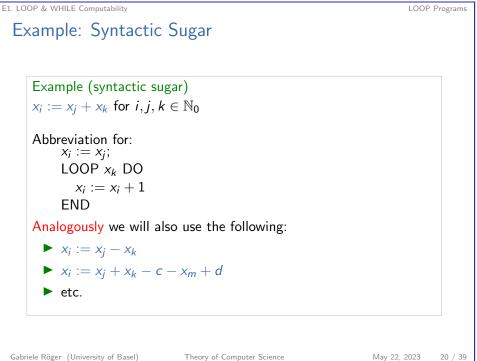
Gabriele Röger (University of Basel)



	ple (syntactic su $_{t}^{c}$ for $i,c\in\mathbb{N}_{0}$	gar)		
where that i	s not an input va	iable, i.e., an o rriable.	otherwise unused v e 0 in all execution	

Theory of Computer Science

E1. LOOP & WHILE Computability		LOOP F	rograms
Example: Syntactic	Sugar		
Example (syntactic supervisition $x_i := x_j$ for $i, j \in \mathbb{N}_0$	gar)		
Simple abbreviation fo	$r x_i := x_j + 0.$		
Gabriele Röger (University of Basel)	Theory of Computer Science	May 22, 2023	18 / 39





LOOP Programs

Example: Syntactic Sugar

Example (syntactic sugar) IF $x_i \neq 0$ THEN *P* END for $i \in \mathbb{N}_0$

Abbreviation for: $x_i := 0;$

Gabriele Röger (University of Basel)

E1. LOOP & WHILE Computability

LOOP x_i DO $x_i := 1$ END; LOOP x_i DO Ρ END where x_i is a fresh variable.

May 22, 2023

21 / 39

WHILE Programs

E1.3 WHILE Programs

Example: Syntactic Sugar

Example (syntactic sugar) IF $x_i = c$ THEN *P* END for $i, c \in \mathbb{N}_0$

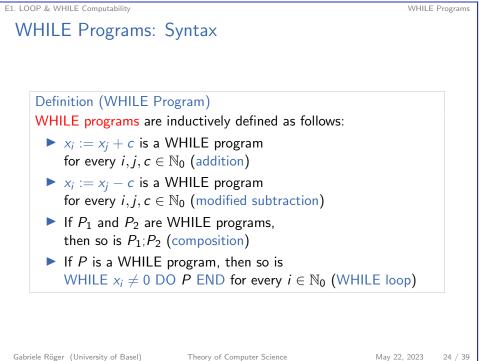
Abbreviation for: $x_i := 1;$ $x_k := x_i - c;$ IF $x_k \neq 0$ THEN $x_i := 0$ END; $x_k := c - x_i;$ IF $x_k \neq 0$ THEN $x_i := 0$ END; IF $x_i \neq 0$ THEN Ρ END where x_i and x_k are fresh variables.

Gabriele Röger (University of Basel)

Theory of Computer Science

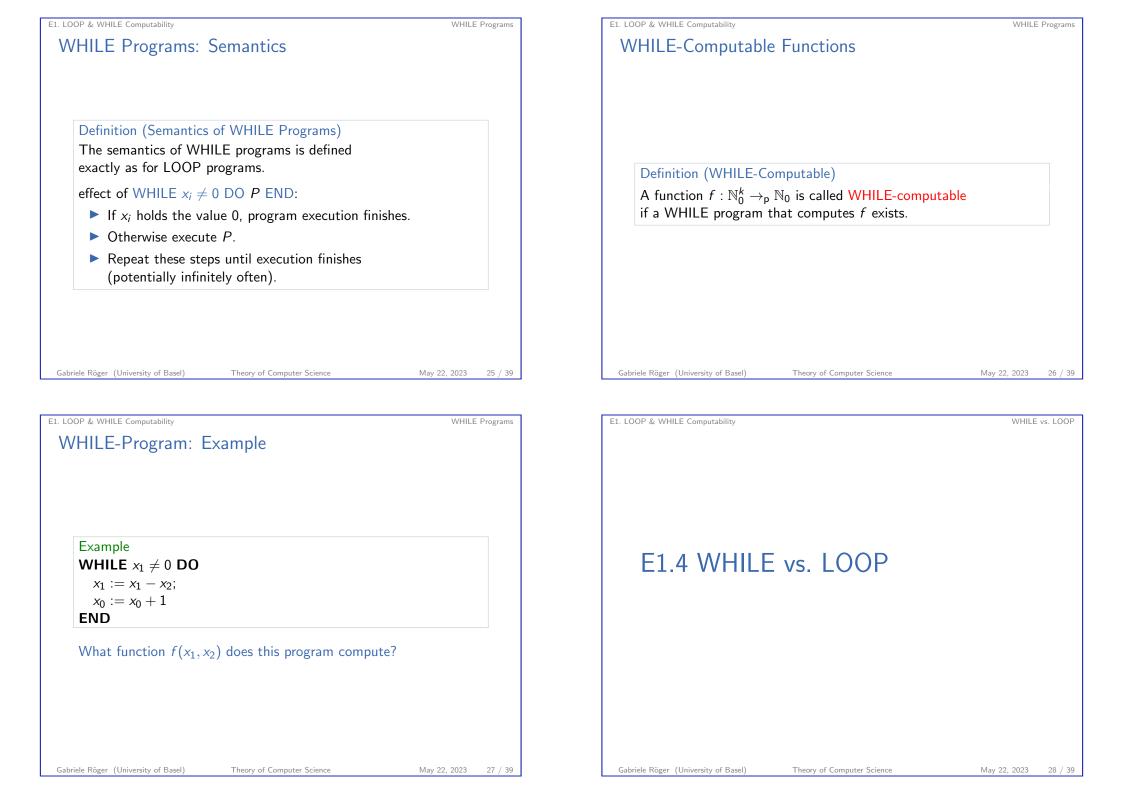
May 22, 2023 22 / 39

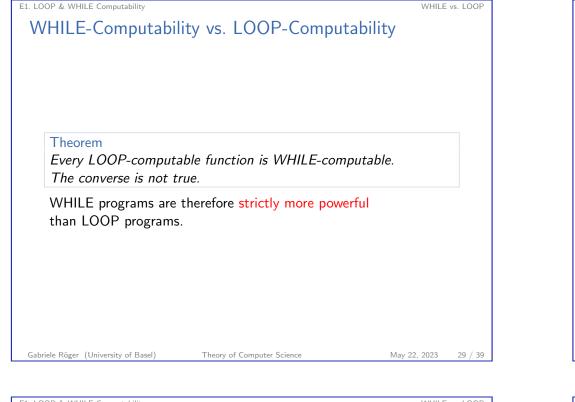
LOOP Programs



Theory of Computer Science

Gabriele Röger (University of Basel)





E1. LOOP & WHILE Computability

WHILE vs. LOOP

```
WHILE-Computability vs. LOOP-Computability
   Proof (continued).
   Part 2: Not all WHILE-computable functions are
   LOOP-computable.
   The WHILE program
        x_1 := 1;
        WHILE x_1 \neq 0 DO
          x_1 := 1
        END
   computes the function \Omega : \mathbb{N}_0 \to_p \mathbb{N}_0 that is undefined everywhere.
   \Omega is hence WHILE-computable, but not LOOP-computable
   (because LOOP-computable functions are always total).
                                                                       Gabriele Röger (University of Basel)
                              Theory of Computer Science
                                                              May 22, 2023
                                                                         31 / 39
```

E1. LOOP & WHILE Computability

WHILE-Computability vs. LOOP-Computability

Proof.

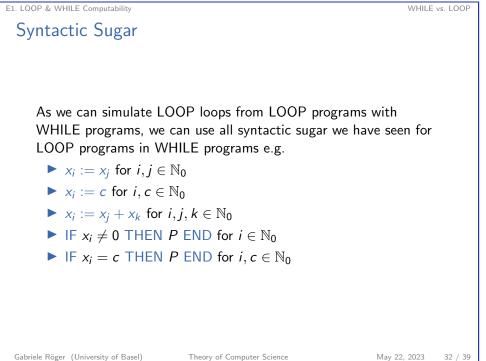
Part 1: Every LOOP-computable function is WHILE-computable.

Given any LOOP program, we construct an equivalent WHILE program, i.e., one computing the same function. To do so, replace each occurrence of LOOP x_i DO P END with $x_i := x_i;$ WHILE $x_i \neq 0$ DO $x_i := x_i - 1;$ P END where x_i is a fresh variable. . . .

Gabriele Röger (University of Basel)

Theory of Computer Science

May 22, 2023 30 / 39



E1. LOOP & WHILE Computability WHILE vs. LOOP E1. LOOP & WHILE Computability LOOP vs. WHILE: Is There a Practical Difference? Ackermann Function: History We have shown that WHILE programs are strictly more powerful than LOOP programs. ▶ The example we used is not very relevant in practice because our argument only relied on the fact Wilhelm Ackermann refuted the conjecture that LOOP-computable functions are always total. by supplying a counterexample (1928). ▶ To terminate for every input is not much of a problem in practice. (Quite the opposite.) \rightsquigarrow here: simplified version ► Are there any total functions that are WHILE-computable, but not LOOP-computable? Theory of Computer Science Gabriele Röger (University of Basel) May 22, 2023 33 / 39 Gabriele Röger (University of Basel) Theory of Computer Science E1. LOOP & WHILE Computability E1. LOOP & WHILE Computability WHILE vs. LOOP

Definition (Ackermann function)	
The Ackermann function $a: \mathbb{N}_0^2 \to \mathbb{N}_0$ is	defined as follows:
a(0,y)=y+1	for all $y \ge 0$
a(x,0)=a(x-1,1)	for all $x > 0$
	for all $x, y > 0$

▶ David Hilbert (1926) conjectured that all computable total functions are primitive recursive (= LOOP-computable). ▶ The counterexample was simplified by Rózsa Péter (1935). May 22, 2023 34 / 39 WHILE vs. LOOP Table of Values v = 1y = 2y = 3y = ky = 0a(0, y)1 2 3 4 k+1a(1, y)2 3 4 5 k+2a(2, y)3 5 7 9 2k + 3 $2^{k+3} - 3$ a(3, y)5 13 29 61

a(4, y)

13

65533

265536 - 3

2²⁶⁵⁵³⁶

WHILE vs. LOOP

E1.	LOOP	&	WHILE	Computability

Computability of the Ackermann Function

Theorem

The Ackermann function is WHILE-computable, but not LOOP-computable.

(Without proof.)

Gabriele Röger (University of Basel) Theory of Computer Science

E1. LOOP & WHILE Computability
Summary

new models of computation for numerical functions: LOOP and WHILE programs

- closer to typical programming languages than Turing machines
- ► WHILE programs strictly more powerful than LOOP programs.
- ▶ WHILE-, but not LOOP-computable functions:
 - simple example: function that is undefined everywhere
 - more interesting example (total function): Ackermann function, which grows too fast to be LOOP-computable

E1. LOOP & WHILE Computability

E1.5 Summary

Gabriele Röger (University of Basel)

Theory of Computer Science

Summarv

May 22, 2023

38 / 39

Gabriele Röger (University of Basel)

Theory of Computer Science

May 22, 2023

37 / 39

Summarv

WHILE vs. LOOP