Theory of Computer Science

D3. GOTO-Computability

Malte Helmert

University of Basel

April 25, 2016

Overview: Computability Theory

Computability Theory

- imperative models of computation:
  - D1. Turing-Computability
  - D2. LOOP- and WHILE-Computability
  - D3. GOTO-Computability
- functional models of computation:
  - D4. Primitive Recursion and μ-Recursion
  - D5. Primitive/μ-Recursion vs. LOOP-/WHILE-Computability
- undecidable problems:
  - D6. Decidability and Semi-Decidability
  - D7. Halting Problem and Reductions
  - D8. Rice's Theorem and Other Undecidable Problems
  - ~~Post's Correspondence Problem~~
  - ~~Undecidable Grammar Problems~~
  - ~~Gödel's Theorem and Diophantine Equations~~

Further Reading (German)

Literature for this Chapter (German)

Theoretische Informatik – kurz gefasst
by Uwe Schöning (5th edition)

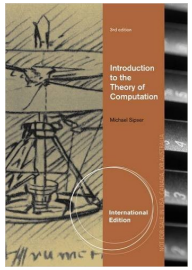
- Chapter 2.3

Further Reading (English)

Literature for this Chapter (English)

~~Introduction to the Theory of Computation by Michael Sipser (3rd edition)~~

- This topic is not discussed by Sipser!



D3.1 WHILE vs. Turing

WHILE-Computability vs. Turing-Computability

Theorem

Every WHILE-computable function is Turing-computable.

(We will discuss the converse statement later.)

WHILE-Computability vs. Turing-Computability

Proof sketch.

Given any WHILE program, we construct an equivalent deterministic Turing machine.

Let $x_1, \ldots, x_k$ be the input variables of the WHILE program, and let $x_0, \ldots, x_m$ be all used variables.

General ideas:

- The DTM simulates the individual execution steps of the WHILE program.
- Before and after each WHILE program step the tape contains the word $bin(n_0)\#bin(n_1)\#\ldots\#bin(n_m)$, where n_i is the value of WHILE program variable x_i.
- It is enough to simulate "minimalistic" WHILE programs ($x_i := x_i + 1$, $x_i := x_i - 1$, composition, WHILE loop).

...

WHILE-Computability vs. Turing-Computability

Proof sketch (continued).

The DTM consists of three sequential parts:

- initialization:
  - Write 0# in front of the used part of the tape.
  - $(m - k)$ times, write #0 behind the used part of the tape.
- execution:
  Simulate the WHILE program (see next slide).
- clean-up:
  - Replace all symbols starting from the first # with $\square$, then move to the first symbol that is not $\square$.

...

WHILE-Computability vs. Turing-Computability

Proof sketch (continued).

Simulation of $x_i := x_i + 1$:

1. Move left until a blank is reached, then one step to the right.
2. $(i + 1)$ times: move right until # or $\square$ is reached.
3. Move one step to the left.

$\rightsquigarrow$ We are now on the last digit of the encoding of x_i.

4. Execute DTM for increment by 1. (Most difficult part: "make room" if the number of binary digits increases.)

...

WHILE-Computability vs. Turing-Computability

Proof sketch (continued).

Simulation of $x_i := x_i - 1$:

1. Move to the last digit of x_i (see previous slide).
2. Test if the digit is a 0 and the symbol to its left is # or $\square$. If so: done.
3. Otherwise: execute DTM for decrement by 1. (Most difficult part: "contract" the tape if the decrement reduces the number of digits.)

...

WHILE-Computability vs. Turing-Computability

Proof sketch (continued).

Simulation of P_1; P_2:

1. Recursively build DTMs M_1 for P_1 and M_2 for P_2.
2. Combine these to a DTM for P_1; P_2 by letting all transitions to end states of M_1 instead go to the start state of M_2.

...

WHILE-Computability vs. Turing-Computability

Proof sketch (continued).

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

1. Recursively build DTM M for P.
2. Build a DTM M' for WHILE $x_i \neq 0$ DO P END that works as follows:
   1. Move to the last digit of x_i.
   2. Test if that symbol is 0 and the symbol to its left is # or $\square$. If so: done.
   3. Otherwise execute M, where all transitions to end states of M are replaced by transitions to the start state of M'.

$\square$

D3.2 GOTO Programs

Intermediate Summary

We now know:

- WHILE programs are strictly more powerful than LOOP programs.
- Deterministic Turing machines are at least as powerful as WHILE programs.

Are DTMs strictly more powerful than WHILE programs or equally powerful?

To answer this question, we make a detour over one more programming formalism.

GOTO Programs: Syntax

Definition (GOTO Program)

A GOTO program is given by a finite sequence $\mathrm{L}_1 : A_1, \mathrm{L}_2 : A_2, \ldots, \mathrm{L}_n : A_n$ 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.

GOTO-Computable Functions

Definition (GOTO-Computable)

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

German: GOTO-berechenbar

D3.3 GOTO vs. WHILE

GOTO-Computability vs. WHILE-Computability

Theorem

Every GOTO-computable function is WHILE-computable.

If we allow IF statements, a single WHILE loop is sufficient for this.

(We will discuss the converse statement later.)

GOTO-Computability vs. WHILE-Computability

Proof sketch.

Given any GOTO program, we construct an equivalent WHILE program with a single WHILE loop (and IF statements).

Ideas:

- Use a fresh variable to store the number of the statement to be executed next.
  - $\rightsquigarrow$ The variable of course has the form x_i, but for readability we write it as *pc* for "program counter".
- GOTO is simulated as an assignment to *pc*.
- If *pc* has the value 0, the program terminates.

...

GOTO-Computability vs. WHILE-Computability

Proof sketch (continued).

Let $L_1 : A_1$, $L_2 : A_2$, ..., $L_n : A_n$ be the given GOTO program.

basic structure of the WHILE program:

$pc := 1$;
WHILE $pc \neq 0$ DO
  IF $pc = 1$ THEN (translation of A_1) END;
  ...
  IF $pc = n$ THEN (translation of A_n) END;
  IF $pc = n + 1$ THEN $pc := 0$ END
END

...

GOTO-Computability vs. WHILE-Computability

Proof sketch (continued).

Translation of the individual statements:

- $x_i := x_j + c$
  $\rightsquigarrow$ $x_i := x_j + c$; $pc := pc + 1$
- $x_i := x_j - c$
  $\rightsquigarrow$ $x_i := x_j - c$; $pc := pc + 1$
- HALT
  $\rightsquigarrow$ $pc := 0$
- GOTO L_j
  $\rightsquigarrow$ $pc := j$
- IF $x_i = c$ THEN GOTO L_j
  $\rightsquigarrow$ $pc := pc + 1$; IF $x_i = c$ THEN $pc := j$ END

□

Intermediate Summary

We now know:

- WHILE programs are strictly more powerful than LOOP programs.
- Deterministic Turing machines are at least as powerful as WHILE programs.
- WHILE programs are at least as powerful as GOTO programs.

We now show that GOTO programs are at least as powerful as DTMs, closing the cycle DTM–WHILE–GOTO.

D3.4 Turing vs. GOTO

Turing-Computability vs. GOTO-Computability

Theorem (Turing-Computability vs. GOTO-Computability)

Every Turing-computable numerical function is GOTO-computable.

Proof.

$\rightsquigarrow$ blackboard. □

Final 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.*

D3.5 Summary

Summary

results of the investigation:

- another new model of computation: GOTO programs
- Turing machines, WHILE and GOTO programs are equally powerful.
- LOOP programs are strictly less powerful.