Theory of Computer Science

D5. Primitive/μ-Recursion vs. LOOP-/WHILE-Computability

Malte Helmert

University of Basel

May 2, 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)

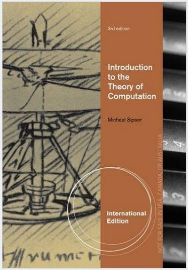
- Chapter 2.4

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!



Introduction

Formal Models of Computation: Primitive and μ-Recursion

Formal Models of Computation

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

In this chapter we compare the primitive recursive and μ-recursive functions to the previously considered models of computation.

Primitive Recursion vs. LOOP

PRFs are LOOP-Computable

Theorem

All primitive recursive functions are LOOP-computable.

(We will discuss the converse statement later.)

PRFs are LOOP-Computable: Proof (1)

Proof.

For every PRF f, we describe a LOOP program computing f.

The proof is by structural induction:

1. Show that basic functions are LOOP-computable.
2. Show that composition of LOOP-computable functions is LOOP-computable.
3. Show that primitive recursion over LOOP-computable functions is LOOP-computable.

We only use LOOP programs that are clean in the following sense:

- After execution, all variables except x_0 hold the same value as initially.
- This allows us to use a stronger inductive hypothesis.

...

PRFs are LOOP-Computable: Proof (2)

Proof (continued).

1. Show that basic functions are LOOP-computable.

- *succ*: $x_0 := x_1 + 1$
- *null*: $x_0 := 0$
- π^i_j: $x_0 := x_j$

...

PRFs are LOOP-Computable: Proof (3)

Proof (continued).

2. Show that composition of LOOP-computable functions is LOOP-computable.

Let $f(z_1, \ldots, z_k) = h(g_1(z_1, \ldots, z_k), \ldots, g_i(z_1, \ldots, z_k))$,
where $h, g_1, \ldots, g_i$ are cleanly computed by $P_h, P_{g_1}, \ldots, P_{g_i}$.

$\rightsquigarrow$ clean program for f:

$z_1 := x_1; \ldots; z_k := x_k;$	Save original inputs.
$P_{g_1}; y_1 := x_0; x_0 := 0;$	Compute $y_1 = g_1(z_1, \ldots, z_k)$.
$\ldots$	$\ldots$
$P_{g_i}; y_i := x_0; x_0 := 0;$	Compute $y_i = g_i(z_1, \ldots, z_k)$.
$x_1 := 0; \ldots; x_k := 0; x_1 := y_1; \ldots; x_i := y_i;$	Set up inputs for h.
$P_h;$	Compute $h(y_1, \ldots, y_i)$.
$x_1 := 0; \ldots; x_i := 0; x_1 := z_1; \ldots; x_k := z_k;$	Restore original inputs.
$y_1 := 0; \ldots; y_i := 0; z_1 := 0; \ldots; z_k := 0$	Clean up.

where $z_1, \ldots, z_k, y_1, \ldots, y_i$ are fresh variables. ...

PRFs are LOOP-Computable: Proof (4)

Proof (continued).

3. Show that primitive recursion over LOOP-computable functions is LOOP-computable.

Let f be created by primitive recursion, i.e.,

$$f(0, z_1, \dots, z_k) = g(z_1, \dots, z_k)$$
$$f(n+1, z_1, \dots, z_k) = h(f(n, z_1, \dots, z_k), n, z_1, \dots, z_k),$$

where g and h are cleanly computed by P_g and P_h.

$\rightsquigarrow$ clean program for f on next slide. ...

PRFs are LOOP-Computable: Proof (5)

Proof (continued).

$rounds := x_1;\ z_1 := x_2;\ \dots;\ z_k := x_{k+1};$	Save original inputs.
$x_1 := z_1;\ \dots;\ x_k := z_k;\ x_{k+1} := 0;$	Set up inputs for g.
$P_g;\ result := x_0;\ x_0 := 0;$	Compute $r_0 = g(z_1, \dots, z_k)$.
LOOP $rounds$ DO	
$\quad x_1 := result;\ x_2 := counter;$	Set up inputs for h.
$\quad x_3 := z_1;\ \dots;\ x_{k+2} := z_k;$	Set up inputs for h.
$\quad P_h;\ result := x_0;\ x_0 := 0;$	Compute $r_{n+1} = h(r_n, n, z_1, \dots, z_k)$.
$\quad counter := counter + 1$	
END;	
$x_0 := result;$	Store final result.
$x_1 := rounds;\ x_2 := z_1;\ \dots;\ x_{k+1} := z_k;$	Restore original inputs.
$rounds := 0;\ result := 0;\ counter := 0;$	Clean up.
$x_{k+2} := 0;\ z_1 := 0;\ \dots;\ z_k := 0$	Clean up.

where $counter$, $result$, $rounds$, $z_1, \dots, z_k$ are fresh variables. □

Questions

Questions?

μ-Recursion vs. WHILE

μRFs are WHILE-Computable

Theorem

All μ-recursive functions are WHILE-computable.

(We will discuss the converse statement later.)

We omit the proof.

Proof idea?

Questions

Questions?

LOOP vs. Primitive Recursion

Encoding and Decoding: Binary Encode

Consider the function $encode : \mathbb{N}_0^2 \to \mathbb{N}_0$ with:

$$encode(x, y) := \binom{x + y + 1}{2} + x$$

- *encode* is known as the Cantor pairing function (German: Cantorsche Paarungsfunktion)
- *encode* is a PRF ($\rightsquigarrow$ exercises)
- *encode* is bijective (without proof)

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$
$y = 0$	0	2	5	9	14
$y = 1$	1	4	8	13	19
$y = 2$	3	7	12	18	25
$y = 3$	6	11	17	24	32
$y = 4$	10	16	23	31	40

Encoding and Decoding: Binary Decode

Consider the inverse functions
$decode_1 : \mathbb{N}_0 \to \mathbb{N}_0$ and $decode_2 : \mathbb{N}_0 \to \mathbb{N}_0$ of *encode*:

$$decode_1(encode(x, y)) = x$$
$$decode_2(encode(x, y)) = y$$

- $decode_1$ and $decode_2$ are PRFs (without proof)

Encoding and Decoding: *n*-ary Case

We can extend encoding and decoding to n-tuples with $n \geq 1$: functions $encode^n : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ and $decode_i^n : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ for all $1 \leq i \leq n$ such that:

$$decode_i^n(encode^n(x_1, \ldots, x_n)) = x_i.$$

- For $n = 1$, use identity function.
- For $n = 2$, use binary encode/decode from previous slides.
- For $n > 2$, define:

$$\begin{aligned} encode^n(x_1, \ldots, x_n) &:= encode(encode^{n-1}(x_1, \ldots, x_{n-1}), x_n) \\ decode_i^n(z) &:= decode_i^{n-1}(decode_1(z)) \quad \text{for all } 1 \leq i < n \\ decode_n^n(z) &:= decode_2(z) \end{aligned}$$

LOOP-Computable Functions are PRFs

Theorem

All LOOP-computable functions are primitive recursive.

LOOP-Computable Functions are PRFs: Proof (1)

Proof.

- For every LOOP program P, we show how to construct the function it computes as a PRF.
- Actually, we first construct a more general PRF: if P uses variables $x_0, \dots, x_m$, we construct a PRF f_P that computes exactly how P changes the values of these variables given any initial assignment to them:

$$f_P(\mathit{initial\_values}) = \mathit{final\_values}$$

- To allow $m + 1$ "outputs", we use encoding/decoding to represent value tuples of size $m + 1$ in one number (both for *initial_values* and *final_values*).

...

LOOP-Computable Functions are PRFs: Proof (2)

Proof (continued).

Assuming that P computes a k-ary function (w.l.o.g. $k \leq m$), the overall function f computed by P can then be represented as:

$$f(a_1, \ldots, a_k) = decode_1^{m+1}(f_P(encode^{m+1}(0, a_1, \ldots, a_k, \underbrace{0, \ldots, 0}_{(m-k) \text{ times}})))$$

This is a PRF if f_P is a PRF. . . .

LOOP-Computable Functions are PRFs: Proof (3)

Proof (continued).

We now show by structural induction how to construct f_P for LOOP programs P of the following form:

1. minimalistic addition: $x_i := x_i + 1$
2. minimalistic modified subtraction: $x_i := x_i - 1$
3. composition: $P_1; P_2$
4. LOOP loop: LOOP x_i DO Q END

...

LOOP-Computable Functions are PRFs: Proof (4)

Proof (continued).

1. minimalistic addition: P is "$x_i := x_i + 1$"

$$\begin{aligned} f_P(z) = encode^{m+1}(&decode_1^{m+1}(z) + c_0, \\ &decode_2^{m+1}(z) + c_1, \\ &\ldots, \\ &decode_{m+1}^{m+1}(z) + c_m), \end{aligned}$$

where $c_i = 1$ and $c_j = 0$ for all $j \neq i$.

This is a PRF: use *succ* to increment by 1 and the identity function (π_1^1) to increment by 0. ...

LOOP-Computable Functions are PRFs: Proof (5)

Proof (continued).

2. minimalistic modified subtraction: P is "$x_i := x_i - 1$"

$$
\begin{aligned}
f_P(z) = encode^{m+1}(&decode_1^{m+1}(z) \ominus c_0, \\
&decode_2^{m+1}(z) \ominus c_1, \\
&\dots, \\
&decode_{m+1}^{m+1}(z) \ominus c_m),
\end{aligned}
$$

where $c_i = 1$ and $c_j = 0$ for all $j \neq i$.

This is a PRF: use *pred* to modified-decrement by 1 and the identity function (π_1^1) to modified-decrement by 0. ...

LOOP-Computable Functions are PRFs: Proof (6)

Proof (continued).

3. composition: P is "P_1; P_2"

By the induction hypothesis, f_{P_1} and f_{P_2} are PRFs. Then

$$f_P(z) = f_{P_2}(f_{P_1}(z))$$

is a PRF representation for f_P. . . .

LOOP-Computable Functions are PRFs: Proof (7)

Proof (continued).

4. LOOP loop: P is "LOOP x_i DO Q END"

By the induction hypothesis, f_Q is a PRF.

We first define an auxiliary function $g_Q : \mathbb{N}_0^2 \to \mathbb{N}_0$ such that $g_Q(k, z)$ encodes k-fold execution of Q with initial values encoded by z:

$$g_Q(0, z) = z$$
$$g_Q(n + 1, z) = f_Q(g_Q(n, z))$$

This is an application of the primitive recursion scheme and hence a PRF. Then

$$f_P(z) = g_Q(decode_{i+1}^{m+1}(z), z)$$

is a PRF representation for f_P. □

Questions

Questions?

WHILE vs. μ-Recursion

WHILE-Computable Functions are μRFs

Theorem

All WHILE-computable functions are μ-recursive.

We omit the proof.

Proof idea:

- extend the previous proof
- use μ-operator to determine how often a given WHILE loop iterates (undefined for infinite loops)
- given their number of iterations, simulate WHILE loops the same way as LOOP loops

Questions

Questions?

Summary

Final Overview: Models of Computation

Theorem (Summary of Results for Models of Computation)

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

The following statements are equivalent:

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

Final Overview: Models of Computation

Theorem (Summary of Results for Models of Computation)

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

The following statements are equivalent:

- *f is LOOP-computable.*
- *f is primitive recursive.*

Further:

- *All LOOP-computable functions/primitive recursive functions are Turing-/WHILE-/GOTO-computable/μ-recursive.*
- *The converse is not true in general.*