Advanced Topics in Theoretical Computer Science

Part 3: Recursive Functions (4)

21.12.2022

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Contents

- Recapitulation: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models: e.g. Büchi Automata, λ-calculus

3. Recursive functions

- Introduction/Motivation
- Primitive recursive functions $\mapsto \mathcal{P}$
- $\mathcal{P} = \text{LOOP}$
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu = \text{WHILE}$
- Summary

Now

- Introduction/Motivation
- Primitive recursive functions $\mapsto \mathcal{P}$
- $\mathcal{P}$ = LOOP
- μ-recursive functions $\mapsto F_\mu$
- F_μ = WHILE
- Summary

μ-recursive Functions

Definition (μ Operator)

$$f(\mathbf{n}) = \mu i(g(\mathbf{n}, i) = 0) = \begin{cases} i_0 & \text{if } g(\mathbf{n}, i_0) = 0 \\ & \quad \text{and for all } 0 \leq j < i_0 \\ & \quad g(\mathbf{n}, j) \text{ defined and } \neq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

The smallest i such that $g(\mathbf{n}, i) = 0$ (undefined if no such i exists or when g is undefined before taking the value 0)

μ-recursive Functions

Notation:

$$f(\mathbf{n}) = \mu i(g(\mathbf{n}, i) = 0)$$

... without arguments:

$$f = \mu g$$

μ-recursive Functions

Definition (μ-recursive Functions)

- **Atomic functions:** The functions
  - Null $\quad 0$
  - Successor $\quad +1$
  - Projection $\quad \pi_i^k \quad (1 \leq i \leq k)$

  are μ-recursive.
- **Composition:** The functions obtained by composition from μ-recursive functions are μ-recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from μ-recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive functions are μ-recursive.

μ-recursive Functions

Definition (μ-recursive Functions)

- **Atomic functions:** The functions
  - Null $\quad 0$
  - Successor $\quad +1$
  - Projection $\quad \pi_i^k \;\; (1 \leq i \leq k)$

  are μ-recursive.
- **Composition:** The functions obtained by composition from μ-recursive functions are μ-recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from μ-recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive functions are μ-recursive.

μ-recursive Functions

Definition (μ-recursive Functions)

- **Atomic functions:** The functions
  - Null $\quad 0$
  - Successor $\quad +1$
  - Projection $\quad \pi_i^k \quad (1 \leq i \leq k)$

  are μ-recursive.
- **Composition:** The functions obtained by composition from μ-recursive functions are μ-recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from μ-recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive functions are μ-recursive.

μ-recursive Functions

Definition (μ-recursive Functions)

- **Atomic functions:** The functions
  - Null $\quad 0$
  - Successor $\quad +1$
  - Projection $\quad \pi_i^k \;\; (1 \leq i \leq k)$

  are μ-recursive.
- **Composition:** The functions obtained by composition from μ-recursive functions are μ-recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from μ-recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive functions are μ-recursive.

μ-recursive Functions

Definition (μ-recursive Functions)

- **Atomic functions:** The functions
  - Null 0
  - Successor $+1$
  - Projection π_i^k $(1 \leq i \leq k)$

  are μ-recursive.
- **Composition:** The functions obtained by composition from μ-recursive functions are μ-recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from μ-recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive functions are μ-recursive.

μ-recursive Functions

Notation:

$$
\begin{aligned}
F_\mu &= \text{Set of all total } \mu\text{-recursive functions} \\
F_\mu^{\text{part}} &= \text{Set of all } \mu\text{-recursive functions} \\
& \text{(total and partial)}
\end{aligned}
$$

μ-recursive Functions

Theorem. $F_\mu \subseteq \mathsf{WHILE}$ and $F_\mu^{\mathsf{part}} \subseteq \mathsf{WHILE}^{\mathsf{part}}$

μ-recursive Functions

Theorem. $F_\mu \subseteq \text{WHILE}$ and $F_\mu^{\text{part}} \subseteq \text{WHILE}^{\text{part}}$

Proof (Idea)

We already proved that $\mathcal{P} = \text{LOOP} \subset \text{WHILE}$.

It remains to show that the μ operator can be "implemented" as a WHILE program.

μ-recursive Functions

Theorem. $F_\mu \subseteq \text{WHILE}$ and $F_\mu^{\text{part}} \subseteq \text{WHILE}^{\text{part}}$

Proof (Idea) We already proved that $\mathcal{P} = \text{LOOP} \subset \text{WHILE}$.

It remains to show that the μ operator can be "implemented" as a WHILE program (below: informal notation)

$i := 0;$
while $g(\mathbf{n}, i) \neq 0$ do $i := i + 1$ end

μ-recursive Functions

Theorem. $F_\mu \subseteq \mathrm{WHILE}$ and $F_\mu^{\mathrm{part}} \subseteq \mathrm{WHILE}^{\mathrm{part}}$

Proof (Idea) We already proved that $\mathcal{P} = \mathrm{LOOP} \subset \mathrm{WHILE}$.

It remains to show that the μ operator can be "implemented" as a WHILE program (below: informal notation)

$i := 0;$
while $g(\mathbf{n}, i) \neq 0$ do $i := i + 1$ end

It can happen that the μ operator is applied to a partial function:

- $g(\mathbf{n}, j)$ might be undefined for some j before a value i is found for which $g(\mathbf{n}, i) = 0$
- $g(\mathbf{n}, i)$ is defined for all i but is never 0.

The μ operator is defined s.t. in such cases it behaves exactly like the while program.

μ-recursive Functions

Question:

Are there μ-recursive functions which are not primitive recursive?

Ackermann Funktion

Wilhelm Ackermann (1896–1962)

- Mathematician and logician
- PhD advisor: D. Hilbert
  Co-author of Hilbert's Book:
  "Grundzüge der Theoretischen Logik"
- Mathematics teacher, Lüdenscheid

μ-recursive Functions

Definition: Ackermann function A

$$\begin{aligned} A(0, y) &= y + 1 \\ A(x+1, 0) &= A(x, 1) \\ A(x+1, y+1) &= A(x, A(x+1, y)) \end{aligned}$$

$$Ack(x) = A(x, x)$$

μ-recursive Functions

Definition: Ackermann function A

$$
\begin{aligned}
A(0, y) &= y + 1 \\
A(x + 1, 0) &= A(x, 1) \\
A(x + 1, y + 1) &= A(x, A(x + 1, y))
\end{aligned}
$$

$$Ack(x) = A(x, x)$$

x y	0	1	2	3	4	. . .	n
0	$0{+}1{=}1$	$1{+}1{=}2$	$2{+}1{=}3$	$3{+}1{=}4$	$4{+}1{=}5$		$n + 1$
1	$A(0, 1){=}2$	$A(0, A(1, 0)){=}3$	$A(0, A(1, 1)){=}4$	$A(0, A(1, 2)){=}5$	$A(0, A(1, 3)){=}6$		$n{+}2$
2	$A(1, 1){=}3$	$A(1, A(2, 0)){=}5$	$A(1, A(2, 1)){=}7$	$A(1, A(2, 2)){=}9$	$A(1, A(2, 3)){=}11$		$2n{+}3$
3	$A(2, 1){=}5$	$A(2, A(3, 0)){=}13$	$A(2, A(3, 1)){=}29$	$A(2, A(3, 2)){=}61$	$A(2, A(3, 3)){=}125$		$2^{n+3} - 3$
4	$A(3, 1)$ $= 2^{2^2} - 3$ $= 13$	$A(3, A(4, 0))$ $= 2^{2^{2^2}} - 3$ $= 65533$	$A(3, A(4, 1))$ $= 2^{2^{2^{2^2}}} - 3$	$A(3, A(4, 2))$ $= 2^{2^{2^{2^{2^2}}}} - 3$	$A(3, A(4, 3))$ $= 2^{2^{2^{65536}}} - 3$		$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}}_{n+3} - 3$
. . .							

μ-recursive Functions

Theorem. The Ackermann function is:

- total
- μ-recursive
- not primitive recursive

μ-recursive Functions

Theorem. The Ackermann function is:

- total
- μ-recursive
- not primitive recursive

Proof: The Ackermann function is total. (In every recursion step one of the arguments is smaller.)

We show that *Ack* is μ-recursive. Idea of proof:

Ack is TM-computable: We can store the recursion stack on the tape of a TM.

We will show that $F_\mu = \text{WHILE}$ and that $\text{TM} \subseteq F_\mu$
From this it will follow that *Ack* is μ-recursive.

μ-recursive Functions

Theorem. The Ackermann function is:

- total
- μ-recursive
- not primitive recursive

Proof: *Ack* is not primitive recursive. Idea of proof:

For a primitive recursive function f, the depth of function unwind needed to compute $f(n)$ is the same for all n. But *Ack* cannot be computed with constant unwind depth. (The detailed proof is complicated.)

μ-recursive Functions

Theorem. The Ackermann function is:

- total
- μ-recursive
- not primitive recursive

Proof: Ack is not primitive recursive. Idea of proof:

For a primitive recursive function f, the depth of function unwind needed to compute $f(n)$ is the same for all n. But Ack cannot be computed with constant unwind depth. (The detailed proof is complicated.)

Alternative proof: We can show that the Ackermann function grows faster than all p.r. functions. (Proof by structural induction)

3. Recursive functions

- Introduction/Motivation
- Primitive recursive functions $\mapsto \mathcal{P}$
- $\mathcal{P} =$ LOOP
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu =$ WHILE
- Summary

3. Recursive functions

- Introduction/Motivation
- Primitive recursive functions $\mapsto \mathcal{P}$
- $\mathcal{P} = \text{LOOP}$
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu = \text{WHILE}$
- Summary

Overview

We know that:

- $\mathsf{LOOP} \subseteq \mathsf{WHILE} = \mathsf{GOTO} \subseteq \mathsf{TM}$
- $\mathsf{WHILE} = \mathsf{GOTO} \subsetneq \mathsf{WHILE}^{\mathsf{part}} = \mathsf{GOTO}^{\mathsf{part}} \subseteq \mathsf{TM}^{\mathsf{part}}$
- $\mathsf{LOOP} \neq \mathsf{TM}$

In this section we proved:

- $\mathsf{LOOP} = \mathcal{P}$
- $F_\mu \subseteq \mathsf{WHILE}$ and $F_\mu^{\mathsf{part}} \subseteq \mathsf{WHILE}^{\mathsf{part}}$

Still to show:

- $\mathsf{TM} \subseteq F_\mu$
- $\mathsf{TM}^{\mathsf{part}} \subseteq F_\mu^{\mathsf{part}}$

TM revisited

(1) Gödelisation of Turing machines

We can associate with every TM

$$M = (K, \Sigma, \delta, s)$$

a unique Gödel number

$$\langle M \rangle \in \mathbb{N}$$

such that

- the coding function (computing $\langle M \rangle$ from M)
- the decoding function (computing the components of M from $\langle M \rangle$)

  are primitive recursive

TM revisited

(2) Gödelisation of configurations of Turing machines

We can associate with every configuration of a given TM

$$C : \quad q, w\underline{a}u$$

a unique Gödel number

$$\langle C \rangle \in \mathbb{N}$$

such that

- the coding function (computing $\langle C \rangle$ from the components of the configuration C)
- the decoding function (computing the components of C from $\langle C \rangle$) are primitive recursive

The Simulation Lemma

Lemma (Simulation Lemma)

There exists a primitive recursive function

$$f_U : \mathbb{N}^3 \to \mathbb{N}$$

such that for every Turing machine M the following hold:

If $C_0, \ldots, C_t$ are configurations of M (where $t \geq 0$) with

$$C_i \vdash_M C_{i+1} \quad (0 \leq i < t)$$

then:

$$f_U(\langle M \rangle, \langle C_0 \rangle, t) = \langle C_t \rangle$$

The Simulation Lemma

Proof. (Idea)

- The coding/decoding functions for TM and configurations are primitive recursive
- Every single step of a TM is primitive recursive
- A given number t of steps in a TM is primitive recursive

Therefore, f_U is primitive recursive.

(Detailed, constructive proof in which the functions are explicitly given: 4 pages in [Erk, Priese])

TM computable functions are μ-recursive

Theorem Every TM computable function is μ-recursive.

$$\text{TM} \subseteq F_\mu \text{ and } \text{TM}^{\text{part}} \subseteq F_\mu^{\text{part}}$$

Proof (Sketch)

Let $f : \mathbb{N}^k \to \mathbb{N}$ be a TM computable function. Let M be a TM which computes f.

$f(n_1, \dots, n_k) = n_{k+1}$ iff $s, \# \underbrace{| \dots |}_{n_1} \# \dots \# \underbrace{| \dots |}_{n_k} \underline{\#} \quad \vdash_M \quad h, \underbrace{| \dots |}_{n_{k+1}} \underline{\#}$

Hence: $f(n_1, \dots, n_k) = (f_U(\langle M \rangle, start, \mu i((f_U(\langle M \rangle, start, i))_{\text{State}} = \langle h \rangle)))_w$, where:

- $start = \left\langle s, \# \underbrace{| \dots |}_{n_1} \# \dots \# \underbrace{| \dots |}_{n_k} \# \right\rangle$
- $\langle h \rangle$ is the Gödelisation of the end state
- $(\cdot)_{\text{State}}$ is the decoding of the state of a configuration
- $(\cdot)_w$ is the decoding of the word left to the writing head

$\mu i(g(\mathbf{n}, i) = h(\mathbf{n}, i))$ is an abbreviation for $\mu i((g(\mathbf{n}, i) \dot{-} h(\mathbf{n}, i)) + (h(\mathbf{n}, i) \dot{-} g(\mathbf{n}, i)) = 0)$
(smallest i for which $g(\mathbf{n}, i) = h(\mathbf{n}, i)$)

Kleene Normal Form

Corollary (Kleene Normal Form)

For every μ-recursive function f there are primitive recursive functions g, h such that

$$f(\mathbf{n}) = g(\mu i(h(\mathbf{n}) = 0))$$

so $f = g \circ \mu h$.

Consequence

$$\boxed{F_\mu = \mathrm{TM} = \mathrm{WHILE}}$$

Summary

Classes of computable functions:

- $\mathsf{LOOP} = \mathcal{P} \subset \mathsf{WHILE} = \mathsf{GOTO} = \mathsf{TM} = F_\mu$
- $\mathsf{WHILE}^{\mathsf{part}} = \mathsf{GOTO}^{\mathsf{part}} = \mathsf{TM}^{\mathsf{part}} = F_\mu^{\mathsf{part}}$
- $\mathsf{LOOP} \neq \mathsf{TM}$

Contents

- Recapitulation: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models: e.g. Büchi Automata, λ-calculus

Contents

- Recapitulation: Turing machines and Turing computability
- Register machines (LOOP, WHILE, GOTO)
- Recursive functions
- The Church-Turing Thesis
- Computability and (Un-)decidability
- Complexity
- Other computation models: e.g. Büchi Automata, λ-calculus

The Church-Turing Thesis

Informally: The functions which are intuitively computable are exactly the functions which are Turing computable.

The Church-Turing Thesis

Informally: The functions which are intuitively computable are exactly the functions which are Turing computable.

Instances of this thesis: all known models of computation

- Turing machines
- Recursive functions
- λ-functions
- all known programming languages (imperative, functional, logic)

provide the same notion of computability

Alonzo Church

Alonzo Church (1903-1995)

- studied in Princeton; PhD in Princeton
- Postdoc in Göttingen
- Professor: Princeton and UCLA
- Layed the foundations of theoretical computer science (e.g. introduced the λ-calculus)
- One of the most important computer scientists

Alonzo Church

PhD Students:

- **Peter Andrews**: automated reasoning
- **Martin Davis**: Davis-Putnam procedure (automated reasoning)
- **Leon Henkin**: (Standard) proof of completeness of predicate logic
- **Stephen Kleene**: Regular expressions
- **Dana Scott**: Denotational Semantics, Automata theory
- **Raymond Smullyan**: Tableau calculi
- **Alan Turing**: Turing machines, Undecidability of the halting problem
- ... and many others

Next time

- Recapitulation: Turing machines and Turing computability
- Recursive functions
- Register machines (LOOP, WHILE, GOTO)
- The Church-Turing Thesis
- **Computability and (Un-)decidability**
- Complexity
- Other computation models