Advanced Topics in Theoretical Computer Science

Part 3: Recursive functions (4)

13.12.2012

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Contents

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

Reminder: Goal

Show that $\mathcal{P} = \text{LOOP}$

Idea:

To show that $\mathcal{P} \supseteq \text{LOOP}$ we have to show that every LOOP computable function can be expressed as a primitive recursive function.

For this, we will encode the contents of arbitrarily many registers in one natural number (used as input for this primitive recursive function).

For this encoding we will use Gödelisation. We will use the fact that Gödelisation is primitive recursive.

To show that $\mathcal{P} \subseteq \text{LOOP}$ we have to show that:

– all atomic primitive recursive functions are LOOP computable, and
– LOOP is closed under composition of functions and primitive recursion.

$\mathcal{P}$ = LOOP

Theorem ($\mathcal{P}$ = LOOP). The set of all LOOP computable functions is equal to the set of all primitive recursive functions

Proof (Idea)

1. $\mathcal{P} \subseteq$ LOOP

1a: We showed that all atomic primitive recursive functions are LOOP computable

1b: We showed that LOOP is closed under composition of functions

1c: We showed that LOOP is closed under primitive recursion

$\mathcal{P} = \textbf{LOOP}$

Theorem ($\mathcal{P} = \textbf{LOOP}$). The set of all LOOP computable functions is equal to the set of all primitive recursive functions

Proof (Idea)

2. LOOP $\subseteq \mathcal{P}$

Let P be a LOOP program which:

- uses registers $x_1, \dots, x_l$
- has m loop instructions

We construct a primitive recursive function f_P which "simulates" P

$$f(\langle n_1, \dots, n_l, h_1, \dots, h_m \rangle) = \langle n'_1, \dots, n'_l, h_1, \dots, h_m \rangle$$

if and only if:

P started with n_i in register x_i terminates with n'_i in x_i $(1 \leq i \leq l)$.

In h_j it is "recorded" how long loop j should still run.

$\mathcal{P}$ = LOOP

Proof (ctd)

At the beginning and at the end of the simulation of P we have

$$h_1 = 0, \ldots, h_m = 0.$$

Assume that we can construct a primitive recursive function f_P which "simulates" P, i.e. $f(\langle n_1, \ldots, n_l, h_1, \ldots, h_m \rangle) = \langle n'_1, \ldots, n'_l, h_1, \ldots, h_m \rangle$ if and only if:

P started with n_i in register x_i terminates with n'_i in x_i $(1 \leq i \leq l)$.

The function computed by the LOOP program P is then primitive recursive, since:

$$g(n_1, \ldots, n_l) = (f_P(\langle n_1, \ldots, n_l, 0, 0, \ldots \rangle))_{l+1}$$

$\mathcal{P}$ = LOOP

Proof (ctd) **Construction of f_P:**

2a: P is $x_i := x_i + 1$

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i + 1, (n)_{i+1}, \ldots \rangle$$

P is $x_i := x_i - 1$

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i - 1, (n)_{i+1}, \ldots \rangle$$

$\mathcal{P} = \textbf{LOOP}$

Proof (ctd) **Construction of f_P:**

2a: P is $x_i := x_i + 1$

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i + 1, (n)_{i+1}, \ldots \rangle$$

P is $x_i := x_i - 1$

$$f_P(n) = \langle (n)_1, \ldots, (n)_{i-1}, (n)_i - 1, (n)_{i+1}, \ldots \rangle$$

2b: P is $P_1; P_2$

$$f_P = f_{P_2} \circ f_{P_1} \qquad i.e.\ f_P(n) = f_{P_2}(f_{P_1}(n))$$

$\mathcal{P}$ = **LOOP**

Proof (ctd) **Construction of f_P:**

2c: P is loop x_i do P_1 end

Let f_{P_1} be the p.r. function which computes what P_1 computes.

Initialize the j-th loop:

$$f_1(n) = \left\langle (n)_1, \dots, (n)_l, (n)_{l+1}, \dots (n)_{l+j-1}, (n)_i, (n)_{l+j+1}, \dots \right\rangle$$

Let the j-th loop run:

$$f_2(n) = \begin{cases} n & \text{if } (n)_{l+j} = 0 \\ f_{P_1}(f_2(\langle \dots, (n)_{l+j} - 1, \dots \rangle)) & \text{otherwise} \end{cases}$$

Then:

$$f_P(n) = f_2(f_1(n)) = (f_2 \circ f_1)(n)$$

$\mathcal{P}$ = LOOP

Proof (ctd) **Construction of f_P:**

2c: P is loop x_i do P_1 end

Let f_{P_1} be the p.r. function which computes what P_1 computes.

Initialize the j-th loop:

$$f_1(n) = \left\langle (n)_1, \dots, (n)_l, (n)_{l+1}, \dots (n)_{l+j-1}, (n)_i, (n)_{l+j+1}, \dots \right\rangle$$

$f_1 = n * p(l+j)^{(n)_i}$. if $(n)_{l+j} = 0$ before the loop is executed

Let the j-th loop run:

$$f_2(n) = \begin{cases} n & \text{if } (n)_{l+j} = 0 \\ f_{P_1}(f_2(n \; DIV \; p(l+j))) & \text{otherwise} \end{cases}$$

Then:

$$f_P = f_2 \circ f_1$$

$\mathcal{P} = \textbf{LOOP}$

Proof (ctd) We show that f_2 is primitive recursive.

Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be defined by:

$F(n, 0) = n$

$F(n, m + 1) = f_{P_1}(F(n, m))$

Then $F \in \mathcal{P}$.

It can be checked that $f_2(n) = F(n, D(n, l + j))$. Therefore, $f_2 \in \mathcal{P}$.

Since f_1, f_2 are primitive recursive, so is $f_P = f_2 \circ f_1$.

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 $\mapsto \mathcal{P}$
- $\mathcal{P} =$ LOOP
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu =$ 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 $\quad \pi_i^k \quad (1 \leq i \leq k)$

  b 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 recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive 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 recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive 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 recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive 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 recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive recursive functions are μ-recursive.

μ-recursive Functions

Definition (μ-recursive Functions)

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

  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 recursive functions are μ-recursive.
- **μ Operator:** The functions obtained by applying the μ operator from μ-recursive recursive functions are μ-recursive.

μ-recursive Functions

Notation:

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

μ-recursive Functions

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

μ-recursive Functions

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

Proof (Idea)

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

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

μ-recursive Functions

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

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

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. $F_\mu \subseteq \text{WHILE}$ 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

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

$$
\begin{aligned}
A_0(x) &= \begin{cases} 1 & \text{is } x = 0 \\ 2 & \text{is } x = 1 \\ x + 2 & \text{otherwise} \end{cases} \\
A_{n+1}(0) &= A_n(1) \\
A_{n+1}(x+1) &= A_n(A_{n+1}(x)) \\
A(x) &= A_x(x)
\end{aligned}
$$

μ-recursive Functions

Definition: Ackermann function A

$$
\begin{aligned}
A_0(x) &= \begin{cases} 1 & \text{is } x = 0 \\ 2 & \text{is } x = 1 \\ x+2 & \text{otherwise} \end{cases} \\
A_{n+1}(0) &= A_n(1) \\
A_{n+1}(x+1) &= A_n(A_{n+1}(x)) \\
A(x) &= A_x(x)
\end{aligned}
$$

$A_1(x) \geq 2 * x;$ $\quad A_2(x) \geq 2^x;$ $\quad A_3(x) \geq \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{x \text{ times}}$

μ-recursive Functions

Definition: Ackermann function A

$$
\begin{aligned}
A_0(x) &= \begin{cases} 1 & \text{is } x = 0 \\ 2 & \text{is } x = 1 \\ x+2 & \text{otherwise} \end{cases} \\
A_{n+1}(0) &= A_n(1) \\
A_{n+1}(x+1) &= A_n(A_{n+1}(x)) \\
A(x) &= A_x(x)
\end{aligned}
$$

$A_1(x) \geq 2 * x$; $\quad A_2(x) \geq 2^x$; $\quad A_3(x) \geq \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{x \text{ times}}$

$A_4(3) \geq 2^{2^{2^2}} = 65536$; $\quad A_4(4) \geq \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{A_4(3) \text{ times}} = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536 \text{ times}}$;

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 functions A_n are total. (In every recursion step one of the arguments is smaller.)

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

A 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 A is μ-recursive.

μ-recursive Functions

Theorem. The Ackermann function is:

- total
- μ-recursive
- not primitive recursive

Proof: A 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 A 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: A 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 A 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 $\mapsto \mathcal{P}$
- $\mathcal{P} = \text{LOOP}$
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu = \text{WHILE}$
- 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:

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

In this section we proved:

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

Still to show:

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

TM revisited

In what follows we will need the following results:

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

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

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

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

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.

$$f(n_1, \dots, n_k) = 0 \text{ iff } s, \#\underbrace{|\dots|}_{n_1}\#\dots\#\underbrace{|\dots|}_{n_k}\underline{\#} \quad \vdash_M \quad h, \underbrace{|\dots|}_{n_k}\underline{\#}$$

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 $\mu i((g(\mathbf{n}, i) - h(\mathbf{n}, i)) + (h(\mathbf{n}, i) - g(\mathbf{n}, i)) = 0)$
(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

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

Summary

Classes of computable functions:

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