Advanced Topics in Theoretical Computer Science

Part 3: Recursive functions (3)

12.12.2013

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

Until now

Primitive recursive functions

Primitive recursion. If the functions $g : \mathbb{N}^k \to \mathbb{N}$ and $h : \mathbb{N}^{k+2} \to \mathbb{N} (k \geq 0)$ are primitive recursive, then the following function is also primitive recursive:

$$f : \mathbb{N}^{k+1} \to \mathbb{N} \text{ with } \quad \begin{aligned} f(\mathbf{n}, 0) &= g(\mathbf{n}) \\ f(\mathbf{n}, m+1) &= h(\mathbf{n}, m, f(\mathbf{n}, m)) \end{aligned}$$

Notation without arguments: $f = \mathcal{PR}[g, h]$

Definition (Primitive recursive functions)

- **Atomic functions:** The functions null (0), successor (+1) and projection ($\pi_i^k \ (1 \leq i \leq k)$) are primitive recursive.
- **Composition:** The functions obtained by composition from primitive recursive functions are primitive recursive.
- **Primitive recursion:** The functions obtained by primitive recursion from primitive recursive functions are primitive recursive.

Examples of primitive recursive functions

$f(n) = n + c, \quad \text{for } c \in \mathbb{N}, c \geq 0 \qquad +(n, m) = n + m$

$(-1)(n) = n - 1 \qquad -(n, m) = n - m$

$*(n, m) = n * m$

Lemma The set of primitive recursive functions is closed under re-ordering, omitting and repeating of arguments when composing functions.

Lemma. Assume $f : \mathbb{N}^k \to \mathbb{N}$ is primitive recursive.
Then, for every $l \in \mathbb{N}$, the function $f' : \mathbb{N}^k \times \mathbb{N}^l \to \mathbb{N}$ defined for every $\mathbf{n} \in \mathbb{N}^k$ and every $\mathbf{m} \in \mathbb{N}^l$ by:

$$f'(\mathbf{n}, \mathbf{m}) = f(\mathbf{n})$$

is primitive recursive.

Primitive recursive functions

Lemma (Case distinction). If g_i, h_i $(1 \leq i \leq r)$ are primitive recursive functions, and for every n there exists a unique i with $h_i(n) = 0$, then the function f defined by:

$$f(n) = \begin{cases} g_1(n) & \text{if } h_1(n) = 0 \\ \dots & \\ g_r(n) & \text{if } h_r(n) = 0 \end{cases}$$

is primitive recursive.

Theorem (Sums and products)
If $g : \mathbb{N}^k \times \mathbb{N} \to \mathbb{N}$ is a primitive recursive function then the following functions $f_1, f_2 : \mathbb{N}^k \times \mathbb{N} \to \mathbb{N}$ are also primitive recursive:

$$f_1(\mathbf{n}, m) = \begin{cases} 0 & \text{if } m = 0 \\ \sum_{i<m} g(\mathbf{n}, i) & \text{if } m > 0 \end{cases} \qquad f_2(\mathbf{n}, m) = \begin{cases} 1 & \text{if } m = 0 \\ \prod_{i<m} g(\mathbf{n}, i) & \text{if } m > 0 \end{cases}$$

Bounded μ operator

Definition. Let $g : \mathbb{N}^{k+1} \to \mathbb{N}$ be a function. The bounded μ operator is defined by:

$$\mu_{i<m}\ i\ (g(\mathbf{n}, i) = 0) := \begin{cases} i_0 & \text{if } g(\mathbf{n}, i_0) = 0 \text{ and for all } j < i_0 \quad g(\mathbf{n}, j) \neq 0 \\ 0 & \text{if } g(\mathbf{n}, j) \neq 0 \text{ for all } 0 \leq j < m \text{ or } m = 0 \end{cases}$$

$\mu_{i<m}\ i\ (g(\mathbf{n}, i) = 0)$ is the smallest $i < m$ such that $g(\mathbf{n}, i) = 0$

Theorem. If $g : \mathbb{N}^{k+1} \to \mathbb{N}$ is a primitive recursive function then the function $f : \mathbb{N}^{k+1} \to \mathbb{N}$ defined by:

$$f(\mathbf{n}, m) = \mu_{i<m}\ i\ (g(\mathbf{n}, i) = 0)$$

is also primitive recursive

Gödelisation

Bijection between $\mathbb{N}^r$ and $\mathbb{N}$:

$K^r : \mathbb{N}^r \to \mathbb{N}$, defined by $K^r(n_1, \ldots, n_r) = \prod_{i=1}^{r} p(i)^{n_i}$.

$D_i : \mathbb{N} \to \mathbb{N}$, $1 \leq i \leq r$, defined by $D_i(n) = D(n, i)$

Theorem. K^r and $D_1, \ldots, D_r$ are primitive recursive.

Lemma.
(1) $D_i(K^r(n_1, \ldots, n_r)) = n_i \quad$ for all $1 \leq i \leq r$.
(2) $K^r(n_1, \ldots, n_r) = K^{r+1}(n_1, \ldots, n_r, 0)$

In general, $D_i(K^r(n_1, \ldots, n_r)) = 0$ if $i > r$.

Notation:

$$K^r(n_1, \ldots, n_r) = \langle n_1, \ldots, n_r \rangle$$

$$D_i(n) = (n)_i$$

For $r = 0$:
$\langle\rangle = 1$
$(\langle\rangle)_i = 0$

Today

- Simultaneous recursion: proof
- $\mathcal{P} = \text{LOOP}$

Gödelisation: Applications

Theorem (Simultaneous Recursion)

If

$$\begin{aligned} f_1(\mathbf{n}, 0) &= g_1(\mathbf{n}) \\ &\cdots \\ f_r(\mathbf{n}, 0) &= g_r(\mathbf{n}) \\ f_1(\mathbf{n}, m+1) &= h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \ldots, f_r(\mathbf{n}, m)) \\ &\cdots \\ f_r(\mathbf{n}, m+1) &= h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \ldots, f_r(\mathbf{n}, m)) \end{aligned}$$

and if $g_1, \ldots, g_r, h_1, \ldots, h_r$ are primitive recursive

then $f_1, \ldots, f_r$ are primitive recursive.

Example

Let f_1 and f_2 be defined by simultaneous recursion as follows:

$f_1(0) = 0$	$g_1 = 0$	
$f_2(0) = 1$	$g_2 = 1$	
$f_1(n+1) = f_2(n)$	$h_1(n, f_1(n), f_2(n)) = f_2(n)$	$h_1 = \pi_3^3$
$f_2(n+1) = f_1(n) + f_2(n)$	$h_2(n, f_1(n), f_2(n)) = f_1(n) + f_2(n)$	$h_2 = + \circ (\pi_2^3, \pi_3^3)$

Gödelisation: Applications

Theorem (Simultaneous Recursion)

If

$$
\begin{aligned}
f_1(\mathbf{n}, 0) &= g_1(\mathbf{n}) \\
&\cdots \\
f_r(\mathbf{n}, 0) &= g_r(\mathbf{n}) \\
f_1(\mathbf{n}, m+1) &= h_1(\mathbf{n}, m, f_1(\mathbf{n}, m), \ldots, f_r(\mathbf{n}, m)) \\
&\cdots \\
f_r(\mathbf{n}, m+1) &= h_r(\mathbf{n}, m, f_1(\mathbf{n}, m), \ldots, f_r(\mathbf{n}, m))
\end{aligned}
$$

and if $g_1, \ldots, g_r, h_1, \ldots, h_r$ are primitive recursive

then $f_1, \ldots, f_r$ are primitive recursive.

Gödelisation: Applications

Proof: We define a new function f by:

$$f(\mathbf{n}, m) = \langle f_1(\mathbf{n}, m), \ldots, f_r(\mathbf{n}, m)\rangle$$

f can be computed by primitive recursion as follows:

$$\begin{aligned} f(\mathbf{n}, 0) &= \langle g_1(\mathbf{n}), \ldots, g_r(\mathbf{n})\rangle \\ f(\mathbf{n}, m+1) &= \langle h_1(\mathbf{n}, m, (f(\mathbf{n}, m))_1, \ldots, (f(\mathbf{n}, m))_r), \ldots, \\ &\quad\; h_r(\mathbf{n}, m, (f(\mathbf{n}, m))_1, \ldots, (f(\mathbf{n}, m))_r)\rangle \end{aligned}$$

$K^r \circ (g_1, \ldots, g_r)$ and $K^r \circ (h_1, \ldots, h_r)$ are primitive recursive.

For all $1 \leq i \leq r$, $f_i(\mathbf{n}, m) = D_i(f(\mathbf{n}, m))$.

Since $f_i = D_i \circ f$ is primitive recursive, it follows that f_i is primitive recursive for all $1 \leq i \leq r$.

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 use Gödelisation. We showed 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

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

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

Proof (Idea)

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

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

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

1c: We show that LOOP is closed under 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: All atomic primitive recursive functions are LOOP computable

$0:$	$\mathtt{x}_1 := \mathtt{x}_1 - 1$	$//NOP$
$+1:$	$\mathtt{x}_2 := \mathtt{x}_1 + 1$	
π_j^k	$\mathtt{x}_{\mathtt{k}+1} := \mathtt{x}_\mathtt{j}$	

$\mathcal{P}$ = LOOP

Proof (ctd) 1b: LOOP is closed under composition of functions

Let $f : \mathbb{N}^k \to \mathbb{N}$ with $f(\mathbf{n}) = h(g_1(\mathbf{n}), \ldots, g_r(\mathbf{n}))$

Assume that:

- P_h computes h
- P_{g_j} computes g_j $(1 \leq j \leq r)$

Idea: f is computed by the program P_f:

$$P'_{g_1}; \ldots; P'_{g_r}; P'_h$$

where P'_{g_i} differs from P_{g_i} (and P'_h from P_h) only up to the fact that registers have been renamed/the contents stored in them copied.

$\mathcal{P}$ = LOOP

Proof (ctd) 1b: LOOP is closed under composition of functions

Let $f : \mathbb{N}^k \rightarrow \mathbb{N}$ with $f(\mathbf{n}) = h(g_1(\mathbf{n}), \dots, g_r(\mathbf{n}))$

Assume that:

- P_h computes h
- P_{g_j} computes g_j $(1 \leq j \leq r)$

More precisely: P'_{g_i}: obtained from P_{g_i} by renaming register x_{k+i} to x_{k+r+i}.

$\mapsto$ keep free registers $x_{k+1}, \dots, x_{k+r}$ for writing result of $P_{g_1}, \dots, P_{g_r}$

P'_h: obtained from P_h by renaming x_j to x_{k+j}.

$$
\begin{aligned}
P_f : \quad & P'_{g_1}; x_{k+1} := x_{k+r+1}; x_{k+r+1} := 0; \dots \\
& P'_{g_r}; x_{k+r} := x_{k+r+1}; x_{k+r+1} := 0; \\
& P'_h; x_{k+1} := x_{k+r+1}; x_{k+2} := 0; \dots ; x_{k+r+1} := 0
\end{aligned}
$$

$\mathcal{P}$ = LOOP

Proof (ctd) 1c: LOOP is closed under primitive recursion

Assume that $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is such that:

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

$$f(\mathbf{n}, m+1) = h(\mathbf{n}, m, f(\mathbf{n}, m))$$

Then f is computed by the following LOOP Program:

$x_{\mathrm{store_m}} := x_{k+1}$; // Number of loops (m)
$x_{k+1} := 0$; //Actual value of m (at the beginning 0)
P'_g; // Computes $f(n, 0)$; result in x_{k+2}
loop $x_{\mathrm{store_m}}$ do
  P_h; // Computes $f(n, x_{k+1} + 1) = h(n, m, f(n, m))$
  $x_{k+2} := x_{k+2+1}$; // $x_{k+2} = f(n, x_{k+1} + 1)$
  $x_{k+2+1} := 0$;
  $x_{k+1} := x_{k+1} + 1$ // $m = m + 1$
end;
$x_{\mathrm{store_m}} := 0$

$\mathcal{P}$ = LOOP

Proof (ctd) 1c: LOOP is closed under primitive recursion

Assume that $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ is such that:

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

$$f(\mathbf{n}, m+1) = h(\mathbf{n}, m, f(\mathbf{n}, m))$$

Then f is computed by the following LOOP Program:

$x_{\text{store}_m} := x_{k+1};$ // Number of loops (m)
$x_{k+1} := 0;$ //Actual value of m (at the beginning 0)
$P'_g;$ // Computes $f(n, 0)$; result in x_{k+2}
loop x_{store_m} do
  $P_h;$
  $x_{k+2} := x_{k+2+1};$
  $x_{k+2+1} := 0;$
  $x_{k+1} := x_{k+1} + 1$
end;
$x_{\text{store}_m} := 0$

where P'_g differs from P_g only in the fact that some registers have been renamed (e.g. output in x_{k+2}, not in x_{k+1})

$\mathcal{P}$ = LOOP

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

Proof (Idea)

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

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

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. $f_P(\langle n_1, \ldots, n_l, h_1, \ldots, h_m \rangle) = \left\langle n'_1, \ldots, n'_l, h_1, \ldots, h_m \right\rangle$ 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 $x_i := x_i - 1$

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

$\mathcal{P}$ = LOOP

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

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

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

P is $x_i := x_i - 1$

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

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:

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

Let the j-th loop run:

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

Then:

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

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

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:

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

$f_1(n) = n * p(l+j)^{(n)_i}$. 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 $\mapsto \mathcal{P}$
- $\mathcal{P} = \text{LOOP}$
- μ-recursive functions $\mapsto F_\mu$
- $F_\mu = \text{WHILE}$
- Summary

Next time

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