Chapter 6
Recursive functions

6.1 Introduction

- Other formalization of the concept of effective procedure: computable functions over the natural numbers.

- Computable functions?
  - Basic functions.
  - Function composition.
  - Recursion mechanism.

6.2 Primitive recursive functions

Functions in the set $\{N^k \to N \mid k \geq 0\}$.

1. Basic primitive recursive functions.

   1. $\mathbf{0}()$

   2. $\pi_i^k(n_1, \ldots, n_k)$

   3. $\sigma(n)$

2. Function composition.

- Let g be a function with ℓ arguments,
- $h_1, \ldots, h_\ell$ functions with k arguments.
- $f(\overline{n}) = g(h_1(\overline{n}), \ldots, h_\ell(\overline{n}))$ is the composition of g and of the functions h_i.

3. Primitive recursion.

- Let g be a function with k arguments and h a function with $k+2$ arguments.

- $$\begin{array}{lcl} f(\overline{n},0) & = & g(\overline{n}) \\ f(\overline{n},m+1) & = & h(\overline{n},m,f(\overline{n},m)) \end{array}$$

  is the function defined from g and h by primitive recursion.

- Remark: f is computable if g and h are computable.

Definition

The *Primitive recursive functions* are :

- the basic primitive recursive functions ;
- all functions that can be obtained from the basic primitive recursive functions by using composition and primitive recursion any number of times.

Examples

Constant functions :

$$\mathbf{j}() \;=\; \overbrace{\sigma(\sigma(\ldots\sigma(}^{j}\mathbf{0}()))$$

Addition function:

$$\begin{array}{lcl} plus(n_1, 0) & = & \pi_1^1(n_1) \\ plus(n_1, n_2 + 1) & = & \sigma(\pi_3^3(n_1, n_2, plus(n_1, n_2))) \end{array}$$

Simplified notation :

$$\begin{array}{lcl} plus(n_1, 0) & = & n_1 \\ plus(n_1, n_2 + 1) & = & \sigma(plus(n_1, n_2)) \end{array}$$

Evaluation of $plus(4,7)$:

$$\begin{aligned} plus(7,4) &= plus(7,3+1) \\ &= \sigma(plus(7,3)) \\ &= \sigma(\sigma(plus(7,2))) \\ &= \sigma(\sigma(\sigma(plus(7,1)))) \\ &= \sigma(\sigma(\sigma(\sigma(plus(7,0))))) \\ &= \sigma(\sigma(\sigma(\sigma(7)))) \\ &= 11 \end{aligned}$$

Product function :

$$\begin{aligned} n \times 0 &= 0 \\ n \times (m+1) &= n + (n \times m) \end{aligned}$$

Power function:

$$\begin{aligned} n^0 &= 1 \\ n^{m+1} &= n \times n^m \end{aligned}$$

Double power :

$$\begin{aligned} n \uparrow\uparrow 0 &= 1 \\ n \uparrow\uparrow m+1 &= n^{n \uparrow\uparrow m} \end{aligned}$$

$$n \uparrow\uparrow m = n^{n^{n^{\cdot^{\cdot^{\cdot^{n}}}}}} \Big\}m$$

Triple power:

$$\begin{array}{lcl} n \uparrow\uparrow\uparrow 0 & = & 1 \\ n \uparrow\uparrow\uparrow m+1 & = & n \uparrow\uparrow (n \uparrow\uparrow\uparrow m) \end{array}$$

k-power :

$$\begin{array}{lcl} n \uparrow^k 0 & = & 1 \\ n \uparrow^k m+1 & = & n \uparrow^{k-1} (n \uparrow^k m). \end{array}$$

If k is an argument:

$$f(k+1, n, m+1) = f(k, n, f(k+1, n, m)).$$

Ackermann's function:

$$\begin{array}{lcl} Ack(0, m) & = & m+1 \\ Ack(k+1, 0) & = & Ack(k, 1) \\ Ack(k+1, m+1) & = & Ack(k, Ack(k+1, m)) \end{array}$$

Factorial function:

$$\begin{array}{lcl} 0! & = & 1 \\ (n+1)! & = & (n+1).n! \end{array}$$

Predecessor function:

$$\begin{array}{lcl} pred(0) & = & 0 \\ pred(m+1) & = & m \end{array}$$

Difference function:

$$\begin{array}{lcl} n \dot{-} 0 & = & n \\ n \dot{-} (m+1) & = & pred(n \dot{-} m) \end{array}$$

Sign function:

$$\begin{array}{lcl} sg(0) & = & 0 \\ sg(m+1) & = & 1 \end{array}$$

Bounded product:

$$f(\overline{n}, m) = \prod_{i=0}^{m} g(\overline{n}, i)$$

$$\begin{array}{lcl} f(\overline{n}, 0) & = & g(\overline{n}, 0) \\ f(\overline{n}, m+1) & = & f(\overline{n}, m) \times g(\overline{n}, m+1) \end{array}$$

6.3 Primitive recursive predicates

A predicate P with k arguments is a subset of N^k (the elements of N^k for which P is true).

The *characteristic function* of a predicate $P \subseteq N^k$ is the function $f : N^k \to \{0, 1\}$such that

$$f(\overline{n}) = \begin{cases} 0 & \text{si } \overline{n} \notin P \\ 1 & \text{si } \overline{n} \in P \end{cases}$$

A predicate is *primitive recursive* if its characteristic function is primitive recursive.

Examples

Zero predicate :

$$\begin{array}{lcl} zerop(0) & = & 1 \\ zerop(n+1) & = & 0 \end{array}$$

$<$ predicate :

$$less(n,m) \;=\; sg(m \dot{-} n)$$

Boolean predicates :

$$\begin{array}{lcl} and(g_1(\overline{n}), g_2(\overline{n})) & = & g_1(\overline{n}) \times g_2(\overline{n}) \\ or(g_1(\overline{n}), g_2(\overline{n})) & = & sg(g_1(\overline{n}) + g_2(\overline{n})) \\ not(g_1(\overline{n})) & = & 1 \dot{-} g_1(\overline{n}) \end{array}$$

$=$ predicate :

$$equal(n,m) \;=\; 1 \dot{-} (sg(m \dot{-} n) + sg(n \dot{-} m))$$

Bounded quantification :

$$\forall i \leq m \ p(\overline{n}, i)$$

is true if $p(\overline{n}, i)$ is true for all $i \leq m$.

$$\exists i \leq m \ p(\overline{n}, i)$$

is true if $p(\overline{n}, i)$ is true for at least one $i \leq m$.

$\forall i \leq m p(\overline{n}, i)$:

$$\prod_{i=0}^{m} p(\overline{n}, i)$$

$\exists i \leq m p(\overline{n}, i)$:

$$1 \dot{-} \prod_{i=0}^{m} (1 \dot{-} p(\overline{n}, i)).$$

Definition by case :

$$f(\overline{n}) = \begin{cases} g_1(\overline{n}) & \text{if } p_1(\overline{n}) \\ \vdots & \\ g_\ell(\overline{n}) & \text{if } p_\ell(\overline{n}) \end{cases}$$

$$f(\overline{n}) = g_1(\overline{n}) \times p_1(\overline{n}) + \ldots + g_\ell(\overline{n}) \times p_\ell(\overline{n}).$$

Bounded minimization :

$$\mu i \leq m \ q(\overline{n}, i) = \begin{cases} \text{the smallest } i \leq m \text{ such that } q(\overline{n}, i) = 1, \\ 0 \text{ if there is no such } i \end{cases}$$

$$\begin{aligned} \mu i \leq 0 \ q(\overline{n}, i) &= 0 \\ \mu i \leq m + 1 \ q(\overline{n}, i) &= \end{aligned}$$

$$\begin{cases} 0 & \text{if } \neg\exists i \leq m + 1 \ q(\overline{n}, i) \\ \mu i \leq m \ q(\overline{n}, i) & \text{if } \exists i \leq m \ q(\overline{n}, i) \\ m + 1 & \text{if } q(\overline{n}, m + 1) \\ & \text{and } \neg\exists i \leq m \ q(\overline{n}, i) \end{cases}$$

6.4 Beyond primitive recursive functions

Theorem

There exist computable functions that are not primitive recursive.

A	0	1	2	$\dots$	j	$\dots$
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$\dots$	$f_0(j)$	$\dots$
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$\dots$	$f_1(j)$	$\dots$
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$\dots$	$f_2(j)$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
f_i	$f_i(0)$	$f_i(1)$	$f_i(2)$	$\dots$	$f_i(j)$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$

$$g(n) = f_n(n) + 1 = A[n, n] + 1.$$

is not primitive recursive, but is computable.

6.4 The μ-recursive functions

Unbounded minimization :

$$\mu i \; q(\overline{n}, i) = \begin{cases} \text{the smallest } i \text{ such that } q(\overline{n}, i) = 1 \\ 0 \text{ if such an } i \text{ does not exist} \end{cases}$$

A predicate $q(\overline{n}, i)$ is said to be *safe* if

$$\forall \overline{n} \; \exists i \; q(\overline{n}, i) = 1.$$

The μ-recursive functions and predicates are those obtained from the basic primitive recursive functions by :

- composition, primitive recursion, and
- unbounded minimization of safe predicates.

μ-recursive functions and computable functions

Numbers and character strings :

Lemma
There exists an effective representation of numbers by character strings.

Lemma
There exists an effective representation of character strings by natural numbers.

Alphabet Σ of size k. Each symbol of Σ is represented by an integer between 0 and $k-1$. The representation of a string $w = w_0 \dots w_l$ is thus:

$$gd(w) = \sum_{i=0}^{l} k^{l-i} gd(w_i)$$

Example : $\Sigma = \{abcdefghij\}$.

$$\begin{array}{rcl} gd(a) & = & 0 \\ gd(b) & = & 1 \\ & \vdots & \\ gd(i) & = & 8 \\ gd(j) & = & 9 \end{array}$$

$$gd(aabaafgj) = 00100569.$$

This encoding is ambiguous :

$$gd(aaabaafgj) = 000100569 = \\ 00100569 = gd(aabaafgj)$$

Solution: use an alphabet of size $k+1$ and do not encode any symbol by 0.

$$gd(w) = \sum_{i=0}^{l} (k+1)^{l-i} gd(w_i).$$

From μ-recursive functions To Turing machines

Theorem

Every μ-recursive function is computable by a Turing machine..

1. The basic primitive recursive functions are Turing machine computable;

2. Composition, primitive recursion and bounded minimization applied to Turing computable functions yield Turing computable functions.

From Turing machines to μ-recursive functions

Theorem
Every Turing computable functions is μ-recursive.

Let M be a Turing machine. One proves that there exists a μ-recursive function f_M such that

$$f_M(w) = gd^{-1}(f(gd(w))).$$

Useful predicates :

1. $init(x)$ initial configuration of M.

2. $next\_config(x)$

3.

$$config(x,n)\begin{cases} config(x,0) = x \\ config(x,n+1) = \\ next\_config(config(x,n)) \end{cases}$$

4. $stop(x) = \begin{cases} 1 & \text{if } x \text{ final} \\ 0 & \text{if not} \end{cases}$

5. $output(x)$

We then have :

$$f(x) = output(config(init(x), nb\_of\_steps(x)))$$

where

$$nb\_of\_steps(x) = \mu i \ \ stop(config(init(x), i)).$$

Partial functions

A partial function $f : \Sigma^* \to \Sigma^*$ is computed by a Turing machine M if,

- for every input word w for which f is defined, M stops in a configuration in which $f(w)$ is on the tape,

- for every input word w for which f is not defined, M does not stop or stops indicating that the function is not defined by writing a special value on the tape.

A partial function $f : N \to N$ is μ-recursive if it can be defined from basic primitive recursive functions by

- composition,
- primitive recursion,
- unbounded minimization.

Unbounded minimization can be applied to unsafe predicates. The function $\mu i \; p(\overline{n}, i)$ is undefined when there is no i such that $p(\overline{n}, i) = 1$.

Theorem
A partial function is μ-recursive if and only if it is Turing computable.