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 \( \{N^k \rightarrow N \mid k \geq 0\} \).

1. Basic primitive recursive functions.

1. \( 0() \)

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

3. \( \sigma(n) \)
2. Function composition.

- Let $g$ be a function with $\ell$ arguments,

- $h_1, \ldots, h_\ell$ functions with $k$ arguments.

- $f(n) = g(h_1(n), \ldots, h_\ell(n))$ 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.

- \[
    f(n, 0) = g(n) \quad f(n, m + 1) = h(n, m, f(n, m))
\]
  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:

\[ j() = \sigma(\sigma(\ldots\sigma(0()))) \]

Addition function:

\[
\begin{align*}
\text{plus}(n_1, 0) & = \pi_1^1(n_1) \\
\text{plus}(n_1, n_2 + 1) & = \sigma(\pi_3^3(n_1, n_2, \text{plus}(n_1, n_2)))
\end{align*}
\]

Simplified notation:

\[
\begin{align*}
\text{plus}(n_1, 0) & = n_1 \\
\text{plus}(n_1, n_2 + 1) & = \sigma(\text{plus}(n_1, n_2))
\end{align*}
\]
Evaluation of $\text{plus}(4, 7)$ :

\[
\text{plus}(7, 4) = \text{plus}(7, 3 + 1) \\
= \sigma(\text{plus}(7, 3)) \\
= \sigma(\sigma(\text{plus}(7, 2))) \\
= \sigma(\sigma(\sigma(\text{plus}(7, 1)))) \\
= \sigma(\sigma(\sigma(\sigma(\text{plus}(7, 0))))) \\
= \sigma(\sigma(\sigma(\sigma(\sigma(7))))) \\
= 11
\]

Product function :

\[
\begin{align*}
n \times 0 &= 0 \\
n \times (m + 1) &= n + (n \times m)
\end{align*}
\]
Power function:

\[
\begin{align*}
n^0 & = 1 \\
n^{m+1} & = n \times n^m
\end{align*}
\]

Double power:

\[
\begin{align*}
n \uparrow\uparrow 0 & = 1 \\
n \uparrow\uparrow m + 1 & = n^{n \uparrow\uparrow m}
\end{align*}
\]

\[
n \uparrow\uparrow m = \underbrace{n^n \cdots}_m
\]
Triple power:

\[
\begin{align*}
    n \uparrow\uparrow\uparrow 0 &= 1 \\
    n \uparrow\uparrow\uparrow m + 1 &= n \uparrow\uparrow (n \uparrow\uparrow\uparrow m)
\end{align*}
\]

\( k \)-power:

\[
\begin{align*}
    n \uparrow^k 0 &= 1 \\
    n \uparrow^k m + 1 &= n \uparrow^{k-1} (n \uparrow^k m).
\end{align*}
\]

If \( k \) is an argument:

\[
    f(k + 1, n, m + 1) = f(k, n, f(k + 1, n, m)).
\]

Ackermann’s function:

\[
\begin{align*}
    Ack(0, m) &= m + 1 \\
    Ack(k + 1, 0) &= Ack(k, 1) \\
    Ack(k + 1, m + 1) &= Ack(k, Ack(k + 1, m))
\end{align*}
\]
Factorial function:

\[
\begin{align*}
0! &= 1 \\
(n + 1)! &= (n + 1).n!
\end{align*}
\]

Predecessor function:

\[
\begin{align*}
pred(0) &= 0 \\
pred(m + 1) &= m
\end{align*}
\]

Difference function:

\[
\begin{align*}
n \cdot 0 &= n \\
n \cdot (m + 1) &= pred(n \cdot m)
\end{align*}
\]
Sign function:

\[ \begin{align*}
    sg(0) &= 0 \\
    sg(m + 1) &= 1
\end{align*} \]

Bounded product:

\[ f(\bar{n}, m) = \prod_{i=0}^{m} g(\bar{n}, i) \]

\[ \begin{align*}
    f(\bar{n}, 0) &= g(\bar{n}, 0) \\
    f(\bar{n}, m + 1) &= f(\bar{n}, m) \times g(\bar{n}, m + 1)
\end{align*} \]
6.3 Primitive recursive predicates

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

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

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

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

Zero predicate:

\[
\begin{align*}
\text{zerop}(0) & = 1 \\
\text{zerop}(n + 1) & = 0
\end{align*}
\]

\(<\) predicate:

\[
\text{less}(n, m) = \text{sg}(m - n)
\]

Boolean predicates:

\[
\begin{align*}
\text{and}(g_1(n), g_2(n)) & = g_1(n) \times g_2(n) \\
\text{or}(g_1(n), g_2(n)) & = \text{sg}(g_1(n) + g_2(n)) \\
\text{not}(g_1(n)) & = 1 - g_1(n)
\end{align*}
\]

\(=\) predicate:

\[
\text{equal}(n, m) = 1 - (\text{sg}(m - n) + \text{sg}(n - m))
\]
Bounded quantification:

\[ \forall i \leq m \ p(n, i) \]

is true if \( p(n, i) \) is true for all \( i \leq m \).

\[ \exists i \leq m \ p(n, i) \]

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

\[ \forall i \leq mp(n, i) : \]

\[ \prod_{i=0}^{m} p(n, i) \]

\[ \exists i \leq mp(n, i) : \]

\[ 1 - \prod_{i=0}^{m} (1 - p(n, i)) \]
Definition by case:

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

\[
f(n) = g_1(n) \times p_1(n) + \ldots + g_\ell(n) \times p_\ell(n).
\]

Bounded minimization:

\[
\mu_i \leq m \quad q(n, i) =
\begin{cases} 
  \text{the smallest } i \leq m \text{ such that } q(n, i) = 1, \\
  0 & \text{if there is no such } i
\end{cases}
\]

\[
\begin{align*}
\mu_i \leq 0 & \quad q(n, i) = 0 \\
\mu_i \leq m + 1 & \quad q(n, i) = \\
\end{align*}
\]

\[
\begin{cases} 
  0 & \text{if } -\exists i \leq m + 1 \quad q(n, i) \\
  \mu_i \leq m \quad q(n, i) & \text{if } \exists i \leq m \quad q(n, i) \\
  m + 1 & \text{if } q(n, m + 1) \quad \text{and } -\exists i \leq m \quad q(n, i)
\end{cases}
\]
### 6.4 Beyond primitive recursive functions

**Theorem**

There exist computable functions that are not primitive recursive.

\[
\begin{array}{c|cccccc}
A & 0 & 1 & 2 & \ldots & j & \ldots \\
\hline
f_0 & f_0(0) & f_0(1) & f_0(2) & \ldots & f_0(j) & \ldots \\
f_1 & f_1(0) & f_1(1) & f_1(2) & \ldots & f_1(j) & \ldots \\
f_2 & f_2(0) & f_2(1) & f_2(2) & \ldots & f_2(j) & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
f_i & f_i(0) & f_i(1) & f_i(2) & \ldots & f_i(j) & \ldots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\
\end{array}
\]

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

is not primitive recursive, but is computable.
6.4 The $\mu$-recursive functions

Unbounded minimization:

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

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

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

The $\mu$-recursive functions and predicates are those obtained from the basic primitive recursive functions by:

- composition, primitive recursion, and

- unbounded minimization of safe predicates (safe unbounded minimization).
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 $\Sigma$ of size $k$. Each symbol of $\Sigma$ is represented by an integer between 0 and $k - 1$. The representation of a string $w = w_0 \ldots w_l$ is thus:

$$gd(w) = \sum_{i=0}^{l} k^{l-i} gd(w_i)$$
Example: \( \Sigma = \{abcdefgij\} \).

\[
\begin{align*}
gd(a) &= 0 \\
gd(b) &= 1 \\
&\vdots \\
gd(i) &= 8 \\
gd(j) &= 9
\end{align*}
\]

\[gd(aaabaaafgj) = 00100569.\]

This encoding is ambiguous:

\[gd(aaabaaafgj) = 000100569 = 00100569 = gd(aaabaaafgj)\]

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 \( \mu \)-recursive functions
To Turing machines

Theorem
Every \( \mu \)-recursive function is computable by a Turing machine.

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

2. Composition, primitive recursion and safe unbounded minimization applied to Turing computable functions yield Turing computable functions.
**Theorem**
Every Turing computable functions is $\mu$-recursive.

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

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

Useful predicates:

1. $\text{init}(x)$ initial configuration of $M$.

2. $\text{next\_config}(x)$
3. \[ \text{config}(x, n) \begin{cases} \text{config}(x, 0) = x \\ \text{config}(x, n + 1) = \text{next\_config}(\text{config}(x, n)) \end{cases} \]

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

5. \text{output}(x)

We then have:

\[ f(x) = \text{output}(\text{config}(\text{init}(x), \text{nb\_of\_steps}(x))) \]

where

\[ \text{nb\_of\_steps}(x) = \mu_i \text{ stop}(\text{config}(\text{init}(x), i)). \]
Partial functions

A partial function \( f : \Sigma^* \rightarrow \Sigma^* \) 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 $f : \mathbb{N} \to \mathbb{N}$ is $\mu$-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 $\mu$-recursive if and only if it is Turing computable.