Recursion theory for realizability

Dick de Jongh

April 10, 2012

The basic principles for the theory of recursive functions $\mathbb{N}^k \to \mathbb{N}$ needed for realizability will be given. We will define the set of effectively computable functions, the recursive functions. The core of this set is formed by the primitive recursive functions, which will be defined first.

Definition 1.

1. The *constant zero function* 0 is the function defined by $0(x) = 0$ (i.e. $S = \lambda x.0$)
2. The *successor function* S is defined by $Sx = x$ (i.e. $S = \lambda x.\, x + 1$).
3. The *projection functions* U_i^n, for each $n \geq 1$, $1 \leq i \leq n$, are defined by $U_i^n(\vec{x}) = x_i$ (where $\vec{x} = x_1, \ldots x_n$, i.e. $U_i^n = \lambda \vec{x}.\, x_i$).

Definition 2. The class of *primitive recursive functions* $\mathbb{N}^k \to \mathbb{N}$ $(k \geq 1)$ is defined as the closure of the set of functions containing the constant zero function, the successor function and the projection functions under the following two operations:

1. *explicit definition*:

   h is defined from f and $g_1, \ldots g_k$ if $h(\vec{x}) = f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$, i.e. $h = \lambda\vec{x}.f(g_1(\vec{x}), \ldots, g_k(\vec{x}))$.

2. *primitive recursion*:

   h is defined from f and g if:

   $h(\vec{x}, 0) = f(\vec{x})$,

   $h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y))$.

All the standard functions like $+$, $\times$, exponentiation, !, etc. are easily seen to be primitive recursive.

Proposition 1.

1. (*pairing function*) There exists a primitive recursive bijection $\mathbf{p}$ of the natural numbers with primitive recursive inverses $\mathbf{p}_0$ and $\mathbf{p}_1$: i.e., for all $x, y, z \in \mathbb{N}$: $\mathbf{p}_0(\mathbf{p}(x, y)) = x$, $\mathbf{p}_1(\mathbf{p}(x, y)) = y$, $\mathbf{p}(\mathbf{p}_0(x), \mathbf{p}_1(x)) = x$.

2. (*sequences*) There exists a bijection $< \cdots >$ from $\mathbb{N}$ to the set of all finite sequences of natural numbers (if $x =< y_0, \ldots, y_n >$ we say that x codes the sequence $(y_1, \ldots, y_n)$) with primitive recursive inverses $\lambda\, x, i.\, x_i$ in such a way that $x_i = y_i$ if $i \leq n$, $x_i = 0$ otherwise.

To obtain all effectively computable functions (at least if one believes the Church-Turing thesis) one introduces partial functions, functions that are not defined on the whole domain $\mathbb{N}^n$. We write $f(\vec{x})\downarrow$ for $f(\vec{x})$ is defined, and $f(\vec{x})\uparrow$ for $f(\vec{x})$ is undefined, $t \simeq t'$ will mean that both t and t' are defined and equal, or both are undefined.

Definition 3. The set *partial recursive functions* is defined as the closure of the set of functions containing the constant zero function, the successor function and the projection functions under explicit definition, primitive recursion and *minimalization*: $f(\vec{x}) \simeq \mu y.\, g(\vec{x}, y) = 0$, i.e. $f(\vec{x})$ is the minimal y such that $f(\vec{x})$ is the minimal y such that $g(\vec{x}, y) = 0$ (and $g(\vec{x}, z)\downarrow$ for all z smaller than y) and is undefined otherwise. A partial recursive function is *total recursive* (or *recursive* for short) if it is defined on its whole domain.

Definition 4. A predicate on $\mathbb{N}^k$ is (primitive) recursive if its characteristic function is.

There exist various types of virtual mathematical machines, the most famous ones *Turing machines*, that can compute all (partial) recursive functions. Turing machines themselves and their computations can be coded by natural numbers in a primitive recursive manner by using the sequence coding.

This leads to the following:

Definition 5.

1. The predicate $T^n(z, \vec{x}, y)$ which is true if y codes a finished computation of the Turing machine coded by z and input $\vec{x}$ (of length n) is primitive recursive. We write T for T^1.

2. The function U that on input y determines the outcome of a computation coded by y is primitive recursive.

So, the Turing machine e is defined on input x if $\exists y\, Texy$. This predicate is typically not primitive recursive, it is in general undecidable.

This leads to *Kleene's Normal Form Theorem*:

Definition 6. 1. An n-place function f is partial recursive iff, for some e, $f(\vec{x}) \simeq U(\mu y.\, T(e, \vec{x}, y)$.

2. In such a case we write $e \bullet \vec{x}$ for $f(\vec{x})$.

Other notations in the literature for $e \bullet \vec{x}$ are $\{e\}(\vec{x})$ (Kleene) and $\varphi_e(\vec{x})$ (Rogers). Basic theorems that will be used are:

Theorem 1. (*The S^m_n-theorem*)

There exist primitive recursive functions $S^m_n(e, x_1, \ldots, x_n)$ such that

$$S^m_n(e, x_1, \ldots, x_n) \bullet (y_1, \ldots, y_m) \simeq e \bullet (x_1, \ldots, x_n, y_1, \ldots, y_m)$$

In case the S^m_n-theorem is applied to a partial recursive function $f(\vec{x}, \vec{y}) = e \bullet (\vec{x}, \vec{y})$ we may write $\Lambda_e \vec{y}.f(\vec{x}, \vec{y})$ for $S^m_n(e, \vec{x})$ and will usually drop the e. The purpose of this is that $\Lambda\vec{y}.f(\vec{x}, \vec{y}) \bullet \vec{y} = f(\vec{x}, \vec{y})$, i.e., $\Lambda\vec{y}$ acts as a λ-notation giving the code of the function instead of the function itself.

Theorem 2. (*Recursion theorem*)

For each $n+1$-place partial recursive function f there exists e such that, for all $\vec{x}$, $e \bullet \vec{x} = f(e, \vec{x})$.