1 Recursiveness and computability

Definition 0.1 (Primitive recursive function)**.**

Proposition 0.1 (Primitive recursive functions)**.**
The following are primitive recursive:

- *Addition:* $x + 0 = x, x + S(y) = S(x + y)$
- *Predecessor:* $pd(0) = 0, pd(x + 1) = x$
- *Integer difference:* $x - 0 = x, x - (y + 1) = pd(x - y)$

Proof. • Define $\begin{cases} f(x,0) = I_1^1(x) \\ f(x,y+1) = h(x,y,f(x,y)) \\ h(x,y,z) = S(I_3^3(x,y,z)) \end{cases}$ □

Definition 0.2 (Recursive μ-operator)**.**
content...

Definition 0.3 (Recursive function)**.**
A function f is defined from a relation R by μ-recursion if:

1. *R is a regular predicate, i.e.* $\forall \overrightarrow{x}.\exists y.R(\overrightarrow{x}, y)$
2. $f(\overrightarrow{x}) = \mu y.R(\overrightarrow{x}, y)$

Theorem 1 (Kleene's Normal Form Theorem)**.**
There exist:

- *U primitive recursive function*
- *for each n, primitive recursive predicates T_n*

such that for any recursive function $f^{(n)}$ there is a number e for which:

- $\forall \overrightarrow{x}.\exists y.T_n(e, \overrightarrow{x}, y)$
- $f(\overrightarrow{x}) = U(\mu y.T_n(e, \overrightarrow{x}, y))$

Proof. Let $f^{(n)}$ be a recursive function. This function can be obtained algorithmically in different forms according to the inductive procedure that we use to generate recursive functions. Once we fix this procedure, we have fixed an algorithmic description of every recursive function. This description is in essence a sequence of elementary steps or computations. The theorem says that for any input $\overrightarrow{x}$, there is a computation giving the value of f at $\overrightarrow{x}$ and that the final result can be obtained through primitive recursive operations using at most one minimization. The function U plays the role of an interpreter, extracting the result value from the computation code. Note also that programs are treated as data, i.e. recursive functions are also reduced to numbers via indexes. This is done so that we can define numeric functions U and T_n.

Recursive function coding:

$O \mapsto \langle 0 \rangle$

$S \mapsto \langle 1 \rangle$

$I_i^n \mapsto \langle 2, n, i \rangle$

$g(h_1(\cdot), \ldots, h_m(\cdot)) \mapsto \langle 3, b_1, \ldots, b_m, a \rangle$

prim. recursion$(g, h) \mapsto \langle 4, a, b \rangle$

bounded recusion$(g) \mapsto \langle 5, a \rangle$ □

2 Basic recursion theory

Definition 1.1 (Partial recursive μ-operator)**.**
A function f is defined from a relation R by μ-recursion if $f(\overrightarrow{x}) \simeq \mu y.R(\overrightarrow{x}, y)$.

Definition 1.2 (Partial recursive functions)**.**
The class of partial recursive functions is the smallest class of functions
1. Containing the inital functions O, S, I_i^n
2. Closed under composition: $\varphi(\overrightarrow{x}) \simeq \psi(\gamma_1(\overrightarrow{x}), \ldots, \gamma_m(\overrightarrow{x}))$
3. Closed under primitive recursion:

$$\varphi(\overrightarrow{x}, 0) \simeq \psi(\overrightarrow{x}), \varphi(\overrightarrow{x}, y+1) = \gamma(\overrightarrow{x}, y, \varphi(\overrightarrow{x}, y))$$

4. Closed under unrestricted μ-recursion:

$$\varphi(\overrightarrow{x}) \simeq \mu y[\forall z \leq y.\psi(\overrightarrow{x}, z) \downarrow \wedge \psi(\overrightarrow{x}, y) \simeq 0]$$

Recall the normal form expression $U(\mu y.T_n(e, \overrightarrow{x}, y))$. In the total setting, it is shown that for e an index of a total recursive function, this gives a value. In particular, the minimized predicate must be regular. But what if e does not correspond to any recursive function? The predicate may not be minimizable, i.e. the corresponding search is not terminating. This problem is removed with the particular μ-operator. In that setting, $\varphi_e^n(\overrightarrow{x}) \simeq U(\mu y.T_n(e, \overrightarrow{x}, y))$ always defines a partial function.

Theorem 2 (Enumeration theorem)**.**
For each e, φ_e^n is a partial recursive function of n variables.

If ψ is a partial recursive function of n variables, then there is e such that $\psi \simeq \varphi_e^n$

There is a partial recursive function $\varphi^{(n+1)}(e, \overrightarrow{x}) \simeq \varphi_e^n(\overrightarrow{x})$

In summary, φ is a partial recursive function that enumerates all partial recursive functions with its parameter e.

Theorem 3 (Padding lemma)**.**
Let e be an index for a partial recursive function φ.

We can effectively generate infinitely many indices of the same function.

Proof. It would suffice to apply the identity function to the coding of φ or to add and subtract the same quantity. Note that the computation trees appearing in the proof of the normal form theorem have equations as nodes. So the proof is phrased in terms of "adding redundant equations" □

Theorem 4 (S_n^m/parametrization theorem)**.**
Let $m, n \in \omega$. There is a primitive recursive, injective function S_n^m such that:

$$\varphi_{S_n^m(e,x_1,\ldots,x_n)}(y_1,\ldots,y_m) \simeq \varphi_e(x_1,\ldots,x_n,y_1,\ldots,y_m)$$

Proof. We take a partial recursive function coded by a program e and we assume that some of its parameters $x_1,\ldots,x_n$ are fixed. The result is some function γ. The theorem says that we can compute a program $S_n^m(e,x_1,\ldots,x_n)$ using only primitive recursive means.

The computation of γ can be described by the function coded by e followed by the equation $\gamma(y_1,\ldots,y_m) \simeq \psi(C_{x_1},\ldots,C_{x_n},y_1,\ldots,y_m)$ (recall the computation trees). This function is clearly partial recursive and its index depends on the code e and the fixed constant, call it $S_n^m(e,x_1,\ldots,x_n)$. By construction the index function is injective and its arithmetization is clearly primitive recursive. □

Note that the enumeration and parametrization theorems play an inverse role translating arguments to indices and viceversa. This is the essence of partial evaluation and supercompilation.

Definition 4.1 (Static complexity measure)**.**
Given an acceptable system of indices $\{\varphi_e\}_{e\in\omega}$ for the partial recursive functions.

We call a static complexity measure any total recursive function m, and call complexity or size of e the number $m(e)$. Then we can naturally talk of the minimal complexity of a partial recursive function as the least complexity of its programs (this notion obviously depends on both to and the acceptable system).