Computability and lambda-calculus

Jacques Garrigue, 2013.11.26

6 Recursive functions

Recursive functions, discovered by Kleene, provide a 3rd definition of computability, after Turing machines and lambda-calculus. Their main advantage is their more mathematical flavour.

6.1 Primitive recursive function

When defining what is computable, we want to start from the most primitive forms. Primitive recursive functions are not sufficient to define all computable functions, but they have an interesting property: they are always guaranteed to terminate.

Definition 1 *The following 5 rules define all* primitive recursive functions *on natural numbers.*

1. *The null function:* $0() = 0$.
2. *The successor function:* $S(x) = x + 1$.
3. *Projection functions:* $\pi_i^n(x_1, \ldots, x_n) = x_i$.
4. *For any primitive recursive functions* $f : N^n \to N$ *and* $g_i : N^m \to N$ $(1 \leq i \leq n)$, *their composition* $f\langle g_1, \ldots, g_n\rangle(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$ *is also primitive recursive.*
5. *For any primitive recursive functions* $g : N^n \to N$ *and* $h : N^{n+2} \to N$, primitive recursion *allow to define*
$$\left\{\begin{array}{rcl} f(x_1, \ldots, x_n, 0) &=& g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y+1) &=& h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{array}\right.$$
of type $f : N^{n+1} \to N$.

For instance, addition can be defined as follows:

$$\left\{\begin{array}{rcl} +(x, 0) &=& \pi_1^1(x) \\ +(x, y+1) &=& h(x, y, +(x, y)) \\ h(x, y, z) &=& S(\pi_3^3(x, y, z)) \end{array}\right.$$

The predecessor function is also easily defined:

$$\left\{\begin{array}{rcl} P(0) &=& 0() \\ P(y+1) &=& h(y, f(y)) \\ h(y, z) &=& \pi_1^2(y, z) \end{array}\right.$$

Substraction can be defined using also composition:

$$\left\{\begin{array}{rcl} -(x, 0) &=& \pi_1^1(x) \\ -(x, y+1) &=& h(x, y, -(x, y)) \\ h(x, y, z) &=& P(\pi_3^3(x, y, z)) \end{array}\right.$$

Exercise 1 *Define* $\times(x, y)$, $exp(x, y) = x^y$, $fact(x) = x!$, $max(x, y)$ *through primitive recursion.*

Theorem 1 *All primitive recursive functions can be computed in finite time.*

Lemma 1 *If $g, h : N \to N$ are primitive recursive, then the function $f : N^2 \to N$ defined as*

$$\begin{cases} f(x,0) &= g(x) \\ f(x,y+1) &= f(h(x),y) \end{cases}$$

is primitive recursive too.

Lemma 2 *If $f : N^{n+1} \to N$ is primitive recursive then the bounded fixpoint operator $\mu_< f : N^{n+1} \to N$ is primitive recursive too.*

$$\mu_< f(x_1,\ldots,x_n,z) = \begin{cases} y & \textit{the smallest } y < z \textit{ such that } f(x_1,\ldots,x_n,y) = 0 \\ z & \textit{if there is no } y < z \textit{ such that } f(x_1,\ldots,x_n,y) = 0 \end{cases}$$

Exercise 2 *Prove that the bounded fixpoint operator is primitive recursive.*

6.2 Recursive functions

Based on the example of Turing machines, the simple fact they always terminate disqualify primitive recursive functions as a model of computability. However, full expressive power can be recovered by the addition of a single construction.

Definition 2 *Functions defined in the following ways are* recursive*.*

1. *All primitive recursive functions.*
2. *The composition of recursive functions.*
3. *Primitive recursion of recursive functions.*
4. *If $f : N^{n+1} \to N$ is recursive, then its minimum fixpoint $\mu f : N^n \to N$*

$$\mu f(x_1,\ldots,x_n) = \begin{cases} y & \textit{the smallest } y < z \textit{ such that } f(x_1,\ldots,x_n,y) = 0 \\ \textit{undefined} & \textit{if there is no } y \textit{ such that } f(x_1,\ldots,x_n,y) = 0 \end{cases}$$

As can be seen in the definition, a recursive function can be undefined for some input n. In this case we say that it is a *partial* recursive function.

Theorem 2 (Kleene normal form) *All recursive functions can be defined by composing a primitive recursive function with the minimum fixpoint of a primitive recursive function.*

Proof. For any recursive function $f : N^n \to N$, we prove by induction on the definition of f the existence of primitive recursive functions $g, h : N^{n+1} \to n$ such that $h(x_1,\ldots,x_n,y) = 0 \Rightarrow h(x_1,\ldots,x_n,y+1) = 0$ and $(\forall y)\ h(x_1,\ldots,x_n,y) = 0 \Rightarrow f(x_1,\ldots,x_n) = g(x_1,\ldots,x_n,y)$, and whenever $f(x_1,\ldots,x_n)$ is defined,$\mu h(x_1,\ldots,x_n)$ is defined too.

For primitive recursive functions, just choose $g(x_1,\ldots,x_n,y) = f(x_1,\ldots,x_n)$.

For the composition of $f : N^m \to N$ and the $f_i : N^n \to N$, we build $f' = f\langle f_1,\ldots,f_m\rangle$. $h'(x_1,\ldots,x_n,y) = h(f_1(x_1,\ldots,x_n),\ldots,f_m(\ldots),y) + h_1(x_1,\ldots,x_n,y) + \ldots + h_m(x_1,\ldots,x_n,y)$ and $g'(x_1,\ldots,x_n,y) = g(g_1(x_1,\ldots,x_n,y),\ldots,g_m(x_1,\ldots,x_n,y),y)$ suffice.

For the minimum ficpoint of $f : N^{n+1} \to N$, we build $f' = \mu f$.
$h'(x_1,\ldots,x_n,y) = \sum_{i=0}^{y-1} h(x_1,\ldots,x_n,i,y) + \mathrm{Z}(y - \mu_{z<y} g(x_1,\ldots,x_n,z,y))$ and $g'(x_1,\ldots,x_n,y) = \mu_{z<y} g(x_1,\ldots,x_n,z,y)$ suffice. Here we use $\mathrm{Z}(0) = 1$, $\mathrm{Z}(y+1) = 0$.

Other cases are easy. □

Since recursive functions are defined in a structural way, we can build the list of all recursive functions. We call f_n the n^{th} 1-parameter recursive function. Similarly, we call p_n then n^{th} primitive recursive function. $f_n(x)$ and $p_n(x)$ can be defined as 2-parameter recursive functions of input (n, x).

Theorem 3 *There exists a total recursive function that is not primitive recursive.*

Proof. $F(x) = p_x(x) + 1$ is total, but it differs from all p_n. □

Theorem 4 *There is a total recursive function that grows faster than any primitive recursive function.*

Theorem 5 *There is a partial recursive function that is not included in any total recursive function. Moreover, when given a list of total recursive functions, one can create a total recursive function that does not belong to this list.*

Proof. We define $F(x) = f_x(x) + 1$ whenever $f_x(x)$ is defined. Since this is a recursive function, it is equal to some f_n, however $F(n)$ must be undefined.

If T_x is a list of total functions, the function $F(x) = T_x(x) + 1$ does not belong to this list. □

6.3 Recursive functions and Turing machines

Theorem 6 *The step function of a Turing machine is primitive recursive.*

To prove this theorem we need to define the behavior of a Turing machine as a function on natural numbers. For simplicity we assume that there are only two symbols including B. On the tape

...	b_2	b_1	b_0	s	c_0	c_1	c_2	...

the head in front of s, in state q, is represented by the following 4-uple

$$q, \quad s, \quad m = \sum_0^\infty b_i 2^i, \quad n = \sum_0^\infty c_i 2^i$$

First, we define the step function as a function on 4-uples. For instance, let us consider a right move. If $\delta(q, s) = (q', s', \leftarrow)$, then the new 4-uple is as follows:

$$q', \quad P(n), \quad s' + 2m, \quad H(n)$$

Here $H(n)$ and $P(n)$ are respectively the result and remainder of dividing n by 2. That is, $P(n) \in \{0, 1\}$ and $P(n) + 2H(n) = n$. They can be defined as follows.

$$\begin{cases} \mathrm{Z}(0) = 1 \\ \mathrm{Z}(y+1) = 0 \end{cases} \qquad \begin{cases} P(0) = 0 \\ P(y+1) = \mathrm{Z}(P(y)) \end{cases} \qquad \begin{cases} H(0) = 0 \\ H(y+1) = H(y) + P(y) \end{cases}$$

We introduce the following 3 functions.

$$\delta(q, s) = (q', s', d) \Leftrightarrow \begin{cases} Q(q, s) = q' & \textit{sttate} \\ R(q, s) = s' & \textit{symbol} \\ D(q, s) = d & \textit{direction} \end{cases}$$

For the direction, 0 is right and 1 is left.

The step functions are as follows.

$$\begin{array}{rcl} q^* & = & Q(q,s) \\ s^* & = & P(m) \times \overline{D}(q,s) + P(n) \times D(q,s) \\ m^* & = & (2 \times m + R(q,s)) \times D(q,s) + H(m) \times \overline{D}(q,s) \\ n^* & = & (2 \times n + R(q,s)) \times \overline{D}(q,s) + H(n) \times (q,s) \\ \overline{D}(q,s) & = & 1 - D(q,s) \end{array}$$

Theorem 7 *Turing computable functions are recursive.*

We just need to repeat the step function until we reach a halting state. However, we would end up appling recursion to multiple arguments. We need to combine them in a single argument.

We define 3 functions $Cons$, Hd and Tl such that $Hd(Cons(x,y)) = x$ and $Tl(Cons(x,y)) = y$. A simple approach is to use $Cons(x,y) = 2^x(2y+1)$.

Hd and Tl are a bit involved. First we define division of even numbers: $J(x) = H(x) \times \mathrm{Z}(P(x))$.

$Div(x,n)$ is 1 if x is a multiple of 2^n, 0 otherwise: $Div(x,0) = \mathrm{Z}(\mathrm{Z}(x)), Div(x,n{+}1) = Div(J(x),n)$

We count the number of times we can divide: $Count(x,0) = 0, Count(x,n+1) = Div(x,n+1) + Count(x,n)$

Finally, $Hd(x) = Count(x,x)$ and $Tl(x) = H^{Hd(x)+1}(x)$. Here we use lemma 1 to define Div and Tl.

The step-function can now be defined as a 1-argument function:

$$\begin{array}{l} Step(x) = Cons(q^*(Hd(x), Hd(Tl(x)), Hd(Tl(Tl(x))), Tl(Tl(Tl(x))), \\ \qquad Cons(s^*(Hd(x),\ldots), Cons(m^*(Hd(x),\ldots), n^*(Hd(x),\ldots))))) \end{array}$$

The primitive recursive function computing n steps is $T_0(x,n) = Step^n(x)$.

Finally, assuming that the final state is 0, T computes until the final state is reached:

$$\begin{array}{l} T(x) = T_0(x, \mu E(x)) \\ E(x,n) = Hd(T_0(x,n)) \end{array}$$

Theorem 8 *Any recursive function can be represented by a Turing machine.*

Proof. Any recursive function can be represented by a $|lambda$-term, and in turn this λ-term can be normalized by a Turing machine. □