15-354: CDM	K. Sutner
Assignment 1	Due: **Feb. 12, 2021**.

1. Historical Ackermann (30)

Background

There is a famous example of a computable function defined by recursion that fails to be primitive recursive, the so-called Ackermann function. What is nowadays called Ackermann function is actually a variation (due to Rózsa Péter) of the original function $B : \mathbb{N}^3 \to \mathbb{N}$:

$$\begin{aligned} B(0,0,z) &= z \\ B(0,y^+,z) &= B(0,y,z)^+ \\ B(1,0,z) &= 0 \\ B(x,0,z) &= 1 \\ B(x^+,y^+,z) &= B(x,B(x^+,y,z),z) \end{aligned}$$

Here u^+ is shorthand for $u+1$. This definition is to be interpreted like a switch statement: always use the first applicable rule (so there is no contradiction between, say, equation 3 and 4). It is useful to think of the ternary function B as a family of binary functions B_x:

$$B_x(y,z) := B(x,y,z).$$

Task

1. Explain the first three levels B_0, B_1 and B_2.
2. Show that each of the functions B_x is primitive recursive.
3. Compute $B_3(3,3)$.
4. What is the problem in computing $B_4(4,4)$? How about $\log B_4(4,4)$?

Comment

The first three levels are all well-known and you should recognize the fourth level.

Solution: Historical Ackermann

Part A: Low Levels

For the first 3 levels, the equations look like so:

$$\begin{aligned} B_0(0,z) &= z \\ B_0(y^+,z) &= B_0(y,z)^+ \end{aligned}$$

$$\begin{aligned} B_1(0,z) &= 0 \\ B_1(y^+,z) &= B_0(B_1(y,z),z) \end{aligned}$$

$$\begin{aligned} B_2(0,z) &= 1 \\ B_2(y^+,z) &= B_1(B_2(y,z),z) \end{aligned}$$

It follows that B_0 is addition, B_1 is multiplication, and B_2 is exponentiation ($B_2(y,z) = z^y$).

Part B: Primitive Recursiveness

We have already shown that the first 3 levels are p.r. For $x \geq 3$, we have the equations

$$\begin{aligned} B_x(0,z) &= 1 \\ B_x(y^+,z) &= B_{x-1}(B_x(y,z),z) \end{aligned}$$

Hence $B_x(y^+,z) = B_{x-1}^y(1,z)$. By induction on x, B_x is p.r.

Part C: B_3

A similar argument shows that the next level is super-exponentiation, $B_3(y,z) = z \uparrow\uparrow y$.

$B_3(3,3) = 3^{3^3} = 3^{27} = 7625597484987$.

Part D: Next Step

Let $\beta = 4^{4^4}$, so we are trying to compute 4^β. OK, but

$$\beta = 13407807929942597099574024998205846127479365820592393377723561443721764030073$$
$$546976801874298166903427690031858186486050853753882811946569946433649006084096$$

a number with 155 decimal digits. Less cryptically, $\beta = 2^{512}$. So the decimal expansion of $B_3(4,4) = 4^\beta$ cannot be written down in this universe, and you can throw in a couple of others, if you like.

2. Lists via Coding (30)

Background

We have seen how to define a coding function, an injective map

$$\langle . \rangle : \mathbb{N}^\star \to \mathbb{N}$$

with "inverse" functions $\mathsf{len} : \mathbb{N} \to \mathbb{N}$ and $\mathsf{dec} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$\begin{aligned} \mathsf{len}(\langle a_0, a_1, \ldots, a_{n-1} \rangle) &= n, \\ \mathsf{dec}(\langle a_0, a_1, \ldots, a_{n-1} \rangle, i\,) &= a_i, \quad i < n \end{aligned}$$

len and dec are primitive recursive and $\langle . \rangle$ is primitive recursive for a fixed number of arguments.

Our claims is that coding allows us to express discrete, finite structures as natural numbers, and the corresponding operations as arithmetic functions. In most cases, the functions will turn out to be primitive recursive. We will verify this claim for lists of natural numbers.

Assume the coding function from class so that

$$\begin{aligned} \langle \mathsf{nil} \rangle &:= 0 \\ \langle a_0, \ldots, a_{n-1} \rangle &:= \pi(a_0, \langle a_1, \ldots, a_{n-1} \rangle) \end{aligned}$$

Recall that with this setup all of $\mathbb{N}$ is the code of some list.

Task

1. Show that the append operation app is primitive recursive:

$$\mathsf{app}(\langle a_0, \ldots, a_{n-1} \rangle, x) = \langle a_0, \ldots, a_{n-1}, x \rangle$$

2. Show that the reversal operation rev is primitive recursive:

$$\mathsf{rev}(\langle a_0, \ldots, a_{n-1} \rangle) = \langle a_{n-1}, \ldots, a_0 \rangle$$

3. Show that the join operation join is primitive recursive:

$$\mathsf{join}(\langle a_0, \ldots, a_{n-1} \rangle, \langle b_0, \ldots, b_{m-1} \rangle) = \langle a_0, \ldots, a_{n-1}, b_0, \ldots, b_{m-1} \rangle$$

Solution: Lists via Coding

Part A: Append

Let $n = \mathsf{len}\, a$. It suffices to define a function

$$F(k, a, x) = \langle a_{n-k}, \ldots, a_{n-1}, x \rangle,$$

since $\mathsf{app}(a, x) = F(n, a, x)$. We use primitive recursion:

$$\begin{aligned} F(0, a, x) &= \langle x \rangle = \pi(x, 0) \\ F(k+1, a, x) &= \pi((a)_{\max(n-k-1,0)}, F(k, a, x)) \end{aligned}$$

where we have written $(a)_i$ instead of $\mathsf{dec}(a, i)$. We assume $(a)_i = 0$ for $i \geq \mathsf{len}\, a$.

Another solution is to exploit bounded search. We have

$$\mathsf{app}(a, x) = \min\big(z \mid \mathsf{len}\, z = \mathsf{len}\, a + 1 \wedge \forall\, i < \mathsf{len}\, a((z)_i = (a)_{\mathsf{len}\, a - i - 1}) \wedge (z)_{\mathsf{len}\, a} = x \big)$$

But the search can be bounded by $2^{\mathsf{len}\, a + 1 + \sum a_i + x + 1} \leq 2^{2a+x+2}$ (this bound depends heavily on our choice of coding function, but similar arguments can be made for other coding functions).

Also note that we could attack this problem by combining prepend (which is easy) with reversal (which is handled in the next subproblem).

Part B: Reverse

Again we can use an auxiliary function

$$F(k, a) = \langle a_{k-1}, \ldots, a_1, a_0 \rangle,$$

since $\mathsf{rev}(a) = F(\mathsf{len}\, a, a)$. We use primitive recursion:

$$F(0, a) = \langle \mathsf{nil} \rangle = 0$$
$$F(k + 1, a) = \pi((a)_k, F(k, a))$$

Of course, there is also a solution base on bounded search. We have

$$\mathsf{rev}(a) = \min\big(z \mid \mathsf{len}\, z = \mathsf{len}\, a \wedge \forall\, i < \mathsf{len}\, a((z)_i = (a)_{\mathsf{len}\, a - i - 1}) \big)$$

Bounds are similar to part (A).

Part C: Join

Once again we can use an auxiliary function

$$F(k, a, b) = \langle a_0, \ldots, a_{n-1}, b_0, \ldots, b_{k-1} \rangle$$

so that $\mathsf{join}(a, b) = F(\mathsf{len}\, b, a, b)$. We use primitive recursion:

$$F(0, a, b) = a$$
$$F(k + 1, a, b) = \mathsf{app}(F(k, a, b), (b)_k)$$

The bounded search solution is left to the reader.

3. Primitive Recursive Word Functions (40)

Background

We defined primitive recursive functions on the naturals. A similar definition would also work for words over some alphabet Σ. We write ε for the empty word and $\Sigma^\star$ for the set of all words over Σ. Consider the clone of word functions generated by the basic functions

- Constant empty word $E : (\Sigma^\star)^0 \to \Sigma^\star$, $E() = \varepsilon$,
- Append functions $S_a : \Sigma^\star \to \Sigma^\star$, $S(x) = x\,a$ where $a \in \Sigma$.

and closed under primitive recursion over words: suppose we have a function $g : (\Sigma^\star)^n \to \Sigma^\star$ and a family of functions $h_a : (\Sigma^\star)^{n+2} \to \Sigma^\star$, where $a \in \Sigma$. We can then define a new function $f : (\Sigma^\star)^{n+1} \to \Sigma^\star$ by

$$\begin{aligned} f(\varepsilon, \boldsymbol{y}) &= g(\boldsymbol{y}) \\ f(xa, \boldsymbol{y}) &= h_a(x, f(x, \boldsymbol{y}), \boldsymbol{y}) \qquad a \in \Sigma \end{aligned}$$

We will call the members of this clone the word primitive recursive (w.p.r.) functions.

Task

1. Show that the reversal operation $\mathsf{rev}(x) = x_n x_{n-1} \ldots x_1$ is w.p.r.
2. Show that the prepend operations $\mathsf{pre}_a(x) = a\,x$ are w.p.r.
3. Show that the concatenation operation $\mathsf{cat}(x, y) = x\,y$ is w.p.r.
4. Prove that every primitive recursive function is also a word primitive recursive function. By this we mean that for every p.r. function $f : \mathbb{N}^k \to \mathbb{N}$ there is a w.p.r. function $F : (\Sigma^\star)^k \to \Sigma^\star$ so that $f(\boldsymbol{x}) = D(F(C(\boldsymbol{x})))$ where C and D are simple coding and decoding functions (between numbers and words).
5. Prove the opposite direction: every w.p.r. is already p.r., using coding and decoding as in the last problem.

Comment

For the last part, don't get bogged down in tons of technical details, just explain how one would go about proving this.

Solution: PR Words Functions

Part A: Prepend

$$\begin{aligned} \mathsf{pre}_a(\varepsilon) &= a \\ \mathsf{pre}_a(xb) &= S_b(\mathsf{pre}_a(x)) \end{aligned}$$

Part B: Reversal

$$\begin{aligned} \mathsf{rev}(\varepsilon) &= \varepsilon \\ \mathsf{rev}(xa) &= \mathsf{pre}_a(\mathsf{rev}(x)) \end{aligned}$$

Part C: Concatenation

$$\begin{aligned}\mathsf{cat}(\varepsilon, y) &= y\\ \mathsf{cat}(xa, y) &= \mathsf{cat}(x, \mathsf{pre}_a(y))\end{aligned}$$

Part D: Translation

We first need to fix the coding functions. Choose the unary alphabet $\Sigma = \{a\}$. To keep things simple, let $C(n) = a^n$, and $D = C^{-1}$.

Then all the basic p.r. functions are obviously w.p.r. (and projections don't even require any coding). Similarly, we have closure in both classes. It's not hard to check that primitive recursion over $\mathbb{N}$ translates directly into primitive recursion for words over $a^\star$: append corresponds to successor (and ε to 0).

The key point here is that we don't need the power of words over larger alphabets. It builds character to show that we could also use, say, the alphabet $\Sigma = \{0, 1\}$, encode natural numbers in binary, and then define successor, addition, multiplication and so on in a w.p.r. fashion.

Part E: Back-Translation

The opposite direction is a little more tedious: we need to represent every w.p.r. function f as a p.r. function $\widehat{f}$.

First, we may safely assume that our alphabet is a digit alphabet of the form $\Sigma = \{1, 2, \ldots, k\}$. We can code words $w = a_1 a_2 \ldots a_n$ over Σ as sequence numbers $\widehat{w} = \langle a_1, \ldots, a_n\rangle$ where $a_i \in [k]$. Write $W \subseteq \mathbb{N}$ for the collection of all such sequence numbers, so W is p.r.. It is clear that in W we can check for the empty word, extract the last letter a_n, drop the last letter $\widehat{v} = \langle a_1, \ldots, a_{n-1}\rangle$ and so on, all in a primitive recursive fashion. For $x \in W$ we write $\widetilde{x}$ for the corresponding word. So we need

$$f(w_1, \ldots, w_n) = \widetilde{z} \quad \text{where} \quad z = \widehat{f}(\widehat{w_1}, \ldots, \widehat{w_n})$$

The w.p.r. basic functions are easy: $\widehat{E} = 0$, the sequence number of nil. The arithmetic version of the successor S_a is given by

$$\widehat{S_a}(x) = \begin{cases} 0 & \text{if } x \notin W\\ \mathsf{app}(x, a) & \text{otherwise.}\end{cases}$$

Here app is the append operation, see the previous problem. The real problem is to deal with primitive recursion over words. For simplicity, let's only handle the case where there are no parameters:

$$\begin{aligned}f(\varepsilon) &= u\\ f(wa) &= h_a(w, f(w)) \qquad a \in \Sigma, w \in \Sigma^\star\end{aligned}$$

For the recursion, note that $\widehat{wa} > \widehat{w}$, so we can use course-of-value recursion. More precisely, define an arithmetic function F as follows:

$$F(x) = \begin{cases} 0 & \text{if } x \notin W,\\ \widehat{u} & \text{if } x = 0,\\ \widehat{h_a}(\widehat{w}, F(\widehat{w})) & \text{if } x = \mathsf{app}(\widehat{w}, a).\end{cases}$$

This shows that F is p.r., so we can set $\widehat{f} = F$.

Comment: It is tempting to say that the coding and decoding maps $w \mapsto \widehat{w}$ and $x \mapsto \widetilde{x}$ are p.r., but that does not typecheck. To fix this problem one can set up a notion of primitive recursive functions over sets, and then our p.r. and w.p.r. functions are just special cases, and related by maps that are set primitive recursive.