15-354: CDM **K. Sutner**

Assignment 1 Due: **Sep. 6, 2024**.

1. Primitive Recursion (30)

Background

In a function based environment we can define bounded search as follows. Let $g : \mathbb{N}^{n+1} \to \mathbb{N}$. Then $f = \mathsf{BS}[g] : \mathbb{N}^{n+1} \to \mathbb{N}$ is the function defined by

$$f(x, \boldsymbol{y}) = \begin{cases} \min\big(\, z < x \mid g(z, \boldsymbol{y}) = 0 \,\big) & \text{if } z \text{ exists,} \\ x & \text{otherwise.} \end{cases}$$

Hence $f(x, \boldsymbol{y}) = x$ indicates failure and $f(x, \boldsymbol{y}) = z < x$ means that z is the least example found.

Task

A. Show that $f(x, \boldsymbol{y}) = \sum_{z<h(x)} g(z, \boldsymbol{y})$ is primitive recursive when h is primitive recursive and strictly monotonic.

B. What if h is not monotonic?

C. Show that $\mathsf{BS}[g]$ is primitive recursive whenever g is.

Comment

Do not simply argue in terms of closure but give explicit primitive recursive definitions of these summation operations.

Solution: Primitive Recursion

We will ignore the parameters $\boldsymbol{y}$ throughout.

Part A: Bounded Sum

Define a summation function $S(a, d) = \sum_{x=a}^{a+d-1} g(x)$. We claim that S is primitive recursive.

$$S(a, 0) = 0$$
$$S(a, d^+) = S(a, d) + g(a + d)$$

Now define

$$f(0) = S(0, h(0))$$
$$f(x^+) = f(x) + S(h(x), h(x^+) - h(x))$$

By monotonicity, the subtraction is actually proper subtraction, so f is primitive recursive. But $f(x)$ is none other than $\sum_{z<h(x)} g(z)$.

Part B: Non Monotonic

The summation is still primitive recursive, but the definition of f is now by cases, depending on whether $h(x) \le h(x^+)$ or not.

Part C: Bounded Search

Define a predicate

$$P(z,x) \equiv \big(z = x \vee (z < x \wedge g(z) = 0)\big) \wedge \forall\, u < z\, (g(u) \neq 0)$$

From the definition, for any x there is a unique $z \leq x$ such that $P(z,x)$ holds, either the first witness for a root of g, or the default value x. Now let

$$f(x) = \sum_{z \leq x} z \cdot \mathsf{char}_P(z,x)$$

using part (A) for the sum. Then f shows that bounded search is primitive recursive.

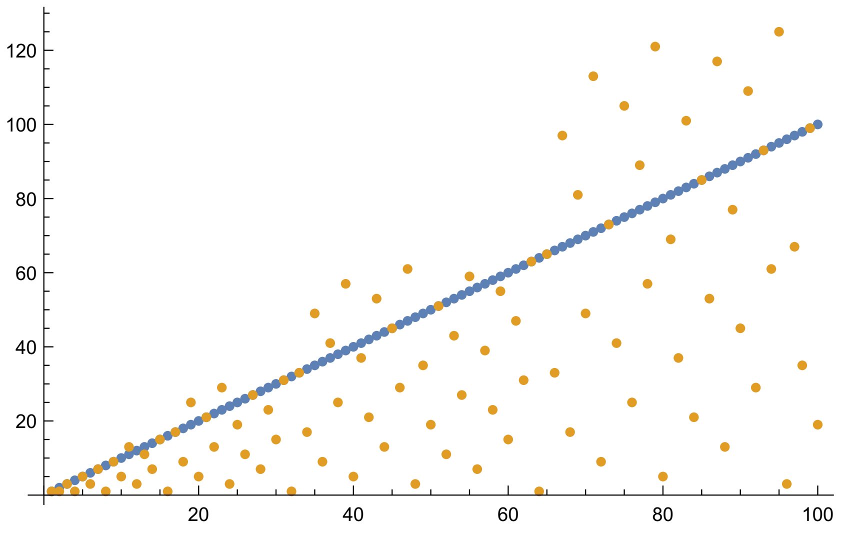
2. A Recursion (30)

Background

Consider the following function f, presumably defined on the positive integers.

$$\begin{aligned} f(1) &= 1 \\ f(3) &= 3 \\ f(2n) &= f(n) \\ f(4n+1) &= 2f(2n+1) - f(n) \\ f(4n+3) &= 3f(2n+1) - 2f(n) \end{aligned}$$

For what it's worth, here is a plot of the first few values.



Task

A. Consider small values of f and conjecture an explicit, non-recursive definition of f.

B. Prove that your definition is correct and conclude that f is indeed a function from $\mathbb{N}_+$ to $\mathbb{N}_+$.

C. Is f primitive recursive?

Comment

Implement f and experiment.

Solution: A Recursion

Part A: Conjecture

It is clear that $f(2^k) = 1$. A little induction shows that Mersenne numbers $n = 2^k - 1$ are fixed points. Similarly Fermat numbers $n = 2^k + 1$ are fixed points. Likewise $f(3\,2^k) = 3$. This suggests that one should probably write the arguments in binary. Write $\mathsf{bin}(x)$ for the binary expansion of x (MSD first, no leading zeros), $\mathsf{val}(b)$ for the numerical value of the binary digit sequence b, and $\mathsf{rev}(b)$ for the reverse of a digit sequence. A few more experiments indicate that

$$f(x) = \mathsf{val}(\mathsf{rev}(\mathsf{bin}(x)))$$

Part B: Proof

Below we write the arguments in binary, but the coefficients are still written in decimal. From the definition of f, we have

$$\begin{aligned} f(1) &= 1 \\ f(11) &= 11 \\ f(x0) &= f(x) \\ f(x01) &= 2f(x1) - f(x) \\ f(x11) &= 3f(x1) - 2f(x) \end{aligned}$$

A modular induction shows that the conjecture really holds.

The claims about domain and codomain follow immediately.

Part C: PR

To show that the characterization from part (A) implies that f is primitive recursive, define a few auxiliary operations:

$$\begin{aligned} M_2(x) &= x \bmod 2 \\ D_2(x) &= x \operatorname{div} 2 \\ K_2(x) &= \max\big(\, z < x \mid 2^z \leq x < 2^{z+1} \,\big) \end{aligned}$$

It is easy to check that these are all primitive recursive.

Now suppose $\mathsf{bin}(x) = d_k d_{k-1} \dots d_1 d_0$ where $d_i \in \mathbf{2}$, $d_k \neq 0$, so that

$$x = \sum_{i=0}^{k} d_i \, 2^i \qquad\qquad f(x) = \sum_{i=0}^{k} d_i \, 2^{k-i}$$

Here $k = K_2(x)$ and $d_i = M_2(D_2^i(x))$, so the sum is over a p.r. function and the bound is also p.r. Hence f is also p.r.

One could also argue more directly about the digit sequence operations as shown in the next problem on p.r. word functions.

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 $\mathsf{rev}(x) = x_n x_{n-1} \dots x_1$ 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: Reversal

We will use a prepend operation, justified in part (B).

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

Part B: Prepend

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

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 $\widehat{f}$. We can ignore the case of one-letter alphabets.

First, we may safely assume that our alphabet is a digit alphabet of the form $\Sigma = \{0, 1, \ldots, k-1\}$, $k > 1$, so we can associate every word w over Σ with a numerical value $\mathsf{val}(w)$ (think of writing numbers in base k). Since we have to go back and forth between $\Sigma^\star$ and $\mathbb{N}$, we need to find a convenient way to do so. There are many possibilities, the cheapest is probably to enumerate all words in length-lex order. For $k = 3$ this would mean

$$\varepsilon, 0, 1, 2, 00, 01, 02, 10, 11, 12, 20, 21, 22, 000, 0001, \ldots$$

The position $\mathsf{pos}(w)$ of w in this list is $(k^{|w|} - 1)/(k-1) + \mathsf{val}(w)$. The inverse function $\mathsf{wrd}(n)$ is slightly more tedious: given n, let $\ell = \lfloor \log_k(n(k-1)+1) \rfloor$; then $\mathsf{wrd}(n)$ is the base k expansion of n, padded to ℓ digits.

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 to show that for any w.p.r. function f there is a p.r. function $\widehat{f}$ such that

$$f(w_1, \ldots, w_n) = \mathsf{wrd}\big(\widehat{f}(\mathsf{pos}(w_1), \ldots, \mathsf{pos}(w_n))\big)$$

The w.p.r. basic functions are easy: $\widehat{E} = 0$. For the arithmetic version of the successor S_a we need to compute $\mathsf{pos}(wa)$ given $\mathsf{pos}(w)$ and a. Letting $\ell = |w|$ we have

$$\mathsf{pos}(wa) = (k^{\ell+1} - 1)/(k-1) + \mathsf{val}(wa) = (k^{\ell+1} - 1)/(k-1) + \mathsf{val}(w) + \mathsf{val}(a)$$

But ℓ and $\mathsf{val}(w)$ can easily be recovered in a p.r. manner from $\mathsf{pos}(w)$, done. Note that one can also go in the opposite direction, given $\mathsf{pos}(wa)$, we can determine $\mathsf{pos}(w)$ and a.

The real problem is to deal with primitive recursion over words. For simplicity, let's only handle the case where there are no parameters. We have a word function

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

For the recursion, note that $\mathsf{pos}(w) < \mathsf{pos}(wa)$, so the corresponding arithmetic function $\widehat{h_a}$ will be called on a smaller argument. This is slightly more complicated than the usual step $x^+ \mapsto x$, corresponding to course-of-value recursion. But that is not really a problem, since we have our sequence number machinery: we can just code up all previous values as a sequence number (in other words, p.r. functions are also closed under course-of-value recursion).

Comment: It is tempting to say that the coding and decoding maps pos and wrd are p.r., but that does not typecheck. To fix this problem one could set up a notion of primitive recursive functions over a larger universe that includes $\mathbb{N}$ and $\Sigma^\star$. Hereditarily finite sets would work, and then our pos and wrd functions indeed turn out to be "primitive recursive."

A less high-powered approach would be to identify a word w with its sequence number $\langle w_1, \ldots, w_\ell \rangle$; one can then check that all our functions are p.r.