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

Assignment 1 Due: **Sep. 18, 2025**.

1. Primitive Recursion (30)

Background

We define bounded search as follows. Let $g : \mathbb{N}^{n+1} \to \mathbb{N}$ and define $f = \mathsf{BS}[g] : \mathbb{N}^{n+1} \to \mathbb{N}$ 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.

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

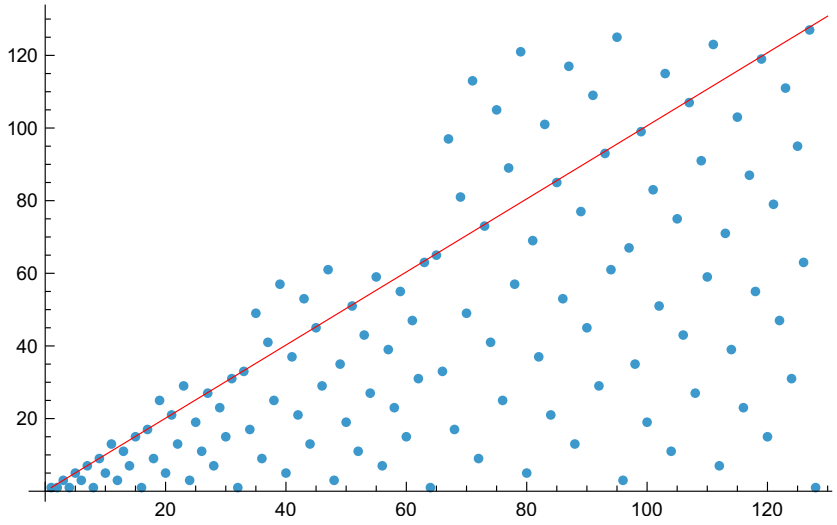
2. A Recursion (30)

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?

vs 2

Comment

It may help to implement f and experiment.

3. Primitive Recursive Word Functions (40)

Background

Models of computation typically use either the natural numbers or strings. The purpose of this problem is to show that one can reasonably define primitive recursive functions on strings, not just the naturals.

We write ε for the empty word and $\Sigma^\star$ for the set of all words over some fixed alphabet 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.

4. Loopy Loops (40)

Background

Consider a small programming language LOOP that has only one data type, natural numbers. The syntax is described in the following table:

constant	$0 \in \mathbb{N}$
variables	x, y, z, ... ranging over $\mathbb{N}$
operations	increment $x{+}{+}$
assignments	$x = 0$ and $x = y$
sequential composition	$P; Q$
control	`do` $x : P$ `od`

The semantics are obvious, except for the loop construct: `do` $x : P$ `od` is intended to mean: "Let n be the value of x before the loop is entered; then execute P exactly n times." Thus, the loop terminates after n rounds even if P changes the value of x. For example, the following LOOP program computes addition:

```
//  add : x, y --> z
        z = x;
        do y :
              z++;
        od
```

Here x and y are input variables, and the result is in z. We assume that all non-input variables are initialized to 0. So, we have a notion of a LOOP-computable function (this is entirely analogous to our definitions for register machines).

Task

A. Show how to implement multiplication and the predecessor function as LOOP programs.

B. What function does the following loop program compute?

```
// mystery : x --> x
   do x:
      do x: x++ od
   od
```

C. Show that every primitive recursive function is LOOP-computable.

D. Show that every LOOP-computable function is primitive recursive.

E. Informally, what is the key difference between LOOP and register machine programs?