DEFINITIONAL SCHEMES FOR PRIMITIVE RECURSIVE AND COMPUTABLE FUNCTIONS

Pace P. Nielsen

Abstract. The unary primitive recursive functions can be defined in terms of a finite set of initial functions together with a finite set of unary and binary operations that are primitive recursive in their inputs. We reduce arity considerations, by show that two fixed unary operations suffice, and a single initial function can be chosen arbitrarily. The method works for many other classes of functions, including the unary partial computable functions. For this class of partial functions we also show that a single unary operation (together with any finite set of initial functions) will never suffice.

1. Introduction

The set of primitive recursive functions is the smallest collection of multivariate functions on the natural numbers that is closed under substitution and recursion, and that contains a standard collection of initial functions (such as the projection functions, the multivariate zero functions, and the successor function). The partial computable functions are defined similarly, by adding another operation such as μ-minimization.

Julia Robinson [6, 7] and Raphael Robinson [8, 9] demonstrated how to eliminate multivariate considerations through the use of pairing strategies. Thus, we will likewise focus only on unary computable functions. In this setting, [7, Theorem 1] is quite striking. It says that all unary primitive recursive functions can be obtained from two certain (complicated) primitive recursive functions, F_1 and F_2, under just two operations: (1) the binary operator of function composition, and (2) the unary operator $f \mapsto f^{\square}$ where

$$f^{\square}(n) = f^n(0)$$

is the n-fold composition of f evaluated at 0; this operation is sometimes called pure iteration.

This naturally raises the question: What is the simplest possible definitional scheme for unary primitive recursive functions?

Gladstone [3, 4] and Georgieva [2] studied this question with a focus on different recursion schemes. The most recent work in this area seems to be that of Severin [10], who improved many previous results. In particular, he showed that one can take the successor function $S(n) = n + 1$ as the single initial function, if in addition to pure iteration and composition we adjoin the binary subtraction operation defined by

$$(f - g)(n) = \begin{cases} f(n) - g(n) & \text{if } f(n) \geq g(n), \\ 0 & \text{otherwise.} \end{cases}$$

By [7, Theorem 3] (also see [11, Theorem 3]), a single initial function is insufficient if our operations are limited to composition and pure iteration.

2020 *Mathematics Subject Classification.* Primary 03D20, Secondary 03D05, 03D10, 03D15.

Key words and phrases. partial computable function, unary primitive recursive function.

In this paper, we are concerned with minimizing the "arity-complexity" of any definitional scheme. We construct two fixed unary operators that generate all the unary primitive recursive functions, using any initial function. The two unary operators can be replaced by a single fixed binary operator.

The proof works by simulating other operations using an encoding of functions along congruence classes. Due to the general nature of this construction, similar results hold for many other classes of functions, including the unary partial computable functions, and in that case we show that no single unary operator will suffice (for any finite set of initial functions).

2. Notations and conventions

Throughout, we will only work with finite definitional schemes. Clearly, if we allow ourselves infinitely many initial functions, then we don't need any operations at all.

We treat each partial function $f\colon \mathbb{N} \to \mathbb{N}$ as the ordered list of its outputs, rather than as a set of ordered pairs. For instance, the successor function is identified with the sequence $(1, 2, 3, \ldots)$. For convenience, if $f(n)$ never terminates, we put the symbol ∞ as the nth term of f. Note that the unary operation $f \mapsto 0^\frown f$ can be implemented with the simple pseudo-code:

```
n:=UserInput,
If[n=0,
   0,
Else
   f(n-1)
]
```

In our definitional schemes, we will not allow arbitrary operations. Rather, an operation is allowable only if it is partial computable in its inputs. In other words, our operations can be performed by a Turing machine that can also make function calls on its inputs. (We may suppose that these function calls are instantly evaluated by an oracle, and if the symbol ∞ is ever returned, then the program immediately terminates and outputs ∞.) When working with primitive recursive functions (and elsewhere, when possible) an operation should be primitive recursive in its inputs.

For instance, the operation $f \mapsto 0^\frown f$ is primitive recursive in its input, as we see using the pseudo-code above, and hence it is allowed. The operations used in the definitional schemes of J. Robinson and Severin are also, clearly, allowed.

Not all operations are allowed. For instance, let $(f_0, f_1, \ldots)$ be a uniformly computable list of all the unary partial computable functions. That such a list can be made, without repetitions, is due to work of Friedberg [1], but also see Kummer's paper [5]. The operation A that takes $f_m \mapsto f_{m+1}$ will not be allowed, as the following argument shows.

Proof. Suppose, by way of contradiction, that A is partial computable in its input. First, consider the case that there exists some $n \in \mathbb{N}$ such that to evaluate $A(f)(n)$ we make no function calls to f. This would mean that $A(f_0)(n) = A(f_1)(n) = \cdots$, which is clearly nonsense, as the nth term of a computable function can become arbitrarily large. Thus, for each $n \in \mathbb{N}$, when evaluating $A(f)(n)$ we must make at least one function call to f. Now letting k be the index such that $f_k = (\infty, \infty, \ldots)$, we have $A(f_k) = f_k \neq f_{k+1}$. □

In the proof above, we leveraged the fact that the empty partial function $(\infty, \infty, \ldots)$ has very specific behavior under unary operations that are partial computable in their inputs. Similar ideas lead to the following strict lower bound on the arity-complexity of any definition of the unary partial computable functions.

Theorem 2.1. *Given finitely many unary partial computable functions, $F_0, F_1, \ldots, F_{m-1}$, and any unary operation, A, that is partial computable in its input, there is some unary partial computable function not expressible in those symbols.*

Proof. First, consider the case when there exists some $n \in \mathbb{N}$ such that $A(f)(n)$ is computed without making any function calls to f. This means that $A(f)(n)$ does not depend on the function f (but could possibly equal ∞). The well-formed expressions from the symbols $\{F_0, F_1, \ldots, F_{m-1}, A\}$ are exactly of the form $A^i(F_j)$ for $i, j \in \mathbb{N}$ with $j \leq m-1$. Thus, their values at n are among the finite set

$$\{F_0(n), F_1(n), \ldots, F_{m-1}(n), A(F_0)(n)\}.$$

It is easy to construct a unary (total) computable function whose value at n differs from these finitely many options.

We may now assume that, for each $n \in \mathbb{N}$, to compute $A(f)(n)$ there must be at least one function call to f. Let $m_n \in \mathbb{N}$ be the number such that the first function call when evaluating $A(f)(n)$ is exactly $f(m_n)$. (The function $n \mapsto m_n$ is computable, since A is computable in its inputs, but we won't need the full power of this fact.) Without loss of generality, expanding and renumbering our initial functions if necessary, we may assume that $F_0 = (\infty, \infty, \ldots)$ is the empty partial function. Note that $A(F_0) = F_0$.

Now, consider the case when $M = \{m_n : n \in \mathbb{N}\}$ is a proper subset of $\mathbb{N}$. Fixing some $k \in \mathbb{N} - M$, for each $\ell \in \mathbb{N}$ we define

$$g_\ell(n) = \begin{cases} \infty & \text{if } n \neq k, \\ \ell & \text{otherwise.} \end{cases}$$

Each g_ℓ is a unary partial computable function. Further, from the fact that $k \notin M$, we have that $A(g_\ell) = F_0$. Thus, for each of the finitely many j in the range $0 \leq j \leq m-1$, the sequence of functions

$$F_j, A(F_j), A^2(F_j), \ldots$$

contains at most one of the g_ℓ (since, immediately after g_ℓ appears, the sequence is constantly F_0). Thus, at least one of the g_ℓ is not expressible in the symbols $\{F_0, F_1, \ldots, F_{m-1}, A\}$, as desired.

All that remains is to handle when $M = \mathbb{N}$. In that case, we see that for each function f we have

$$|\{n \in \mathbb{N} : f(n) = \infty\}| \leq |\{n \in \mathbb{N} : A(f)(n) = \infty\}|.$$

Write $C_f = |\{n \in \mathbb{N} : f(n) = \infty\}|$. Thus, for each j, the cardinalities

$$C_{F_j} \leq C_{A(F_j)} \leq C_{A^2(F_j)} \leq \ldots$$

either each have a finite upper bound k_j, or not, in which case we set $k_j = 0$. Letting $k = \max(k_0, k_1, \ldots, k_{m-1}) + 1$, we see that there are only finitely many expressions G in $\{F_0, F_1, \ldots, F_{m-1}, A\}$ such that $C_G = k$. But there are infinitely many unary partial computable functions with that same property. □

As we will see in the next section, we have very different behavior when allowing two unary operations, or a single binary operation.

3. Simulating higher arity operations using two unary operations

As mentioned in the introduction, J. Robinson proved that every unary primitive recursive function can be expressed in the symbols $\{F_1, F_2, {}^{\square}, \circ\}$, where F_1 and F_2 are specific initial functions. With access to a binary operator, one can define operations of higher arity; for instance, $(f, g, h) \mapsto f \circ (g \circ h)$ is a 3-ary operation. However, if we limit ourselves to unary operations, there is no way to define derived operations of higher arity. In particular, we have no way to define composition.

Instead, we will simulate composition in the following manner. Take $f = (a_0, a_1, \ldots)$ and $g = (b_0, b_1, \ldots)$ to be two arbitrary functions. If we can somehow construct the function $h = (a_0, b_0, a_1, b_1, \ldots)$, then we can simulate the composition of f and g by using computations on the single function h.

More generally, for any modulus $m \geq 2$, and functions $f_0, f_1, \ldots f_{m-1}$, we write the m-tuple $(f_0, f_1, \ldots, f_{m-1})$ to denote the function

$$(f_0(0), f_1(0), \ldots, f_{m-1}(0), f_0(1), f_1(1), \ldots, f_{m-1}(1), \ldots).$$

In other words, this is the function where each f_i has been encoded along the congruence class $i \pmod m$.

Given an m-tuple of functions $(f_0, f_1, \ldots, f_{m-1})$ we will make use of the following *unary* operations.

Operation 1: Right rotation:

$$(f_0, f_1, \ldots, f_{m-2}, f_{m-1}) \mapsto (f_{m-1}, f_0, f_1, \ldots, f_{m-2}).$$

Operation 2: Switching the first two functions:

$$(f_0, f_1, *, \ldots, *) \mapsto (f_1, f_0, *, \ldots, *).$$

(Here, and hereafter, we use $*$ to denote an entry that is left unchanged.)

The symmetric group on m letters is always generated by the m-cycle $(1\,2\,\ldots\,m)$ and the transposition $(1\,2)$. Thus, with these two operations in place, we can permute the entries of any m-tuple arbitrarily.

Operation 3: Replace the first function with F_1:

$$(f_0, *, \ldots, *) \mapsto (F_1, *, \ldots, *).$$

Operation 4: Replace the first function with F_2:

$$(f_0, *, \ldots, *) \mapsto (F_2, *, \ldots, *).$$

Operation 5: Apply pure iteration to the first function:

$$(f_0, *, \ldots, *) \mapsto (f_0^{\square}, *, \ldots, *).$$

Operation 6: Replace the first function with the composition of the first two functions:

$$(f_0, f_1, *, \ldots, *) \mapsto (f_0 \circ f_1, f_1, *, \ldots, *).$$

Starting with the constant zero function $\overline{0}$, we use these operations to simulate the construction of any unary primitive recursive function. We illustrate this process by constructing $G = (F_1F_2)^{\square}(F_1(F_2F_1)^{\square})^{\square}$. (We write compositions as concatenations to ease notation.)

Take $m = 5$; any modulus at least as large as the number of instances of F_1 and F_2 will suffice. We then have

$$\begin{aligned}
&(\overline{0},\overline{0},\overline{0},\overline{0},\overline{0}) \overset{3}{\mapsto} (F_1,\overline{0},\overline{0},\overline{0},\overline{0}) \overset{1}{\mapsto} (\overline{0},F_1,\overline{0},\overline{0},\overline{0}) \overset{4}{\mapsto} (F_2,F_1,\overline{0},\overline{0},\overline{0}) \overset{6}{\mapsto} (F_2F_1,F_1,\overline{0},\overline{0},\overline{0})\\
\overset{5}{\mapsto}\ &((F_2F_1)^{\square},F_1,\overline{0},\overline{0},\overline{0}) \overset{1}{\mapsto} (\overline{0},(F_2F_1)^{\square},F_1,\overline{0},\overline{0}) \overset{3}{\mapsto} (F_1,(F_2F_1)^{\square},F_1,\overline{0},\overline{0})\\
\overset{6}{\mapsto}\ &(F_1(F_2F_1)^{\square},(F_2F_1)^{\square},F_1,\overline{0},\overline{0}) \overset{5}{\mapsto} ((F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1,\overline{0},\overline{0})\\
\overset{1}{\mapsto}\ &(\overline{0},(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1,\overline{0}) \overset{4}{\mapsto} (F_2,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1,\overline{0})\\
\overset{1}{\mapsto}\ &(\overline{0},F_2,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1) \overset{3}{\mapsto} (F_1,F_2,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1)\\
\overset{6}{\mapsto}\ &(F_1F_2,F_2,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1) \overset{5}{\mapsto} ((F_1F_2)^{\square},F_2,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1)\\
\overset{2}{\mapsto}\ &(F_2,(F_1F_2)^{\square},(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1) \overset{1\circ 4}{\longmapsto} ((F_1F_2)^{\square},(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1,F_2)\\
\overset{6}{\mapsto}\ &(G,(F_1(F_2F_1)^{\square})^{\square},(F_2F_1)^{\square},F_1,F_2)
\end{aligned}$$

Of course, we also need a seventh operation, $(f_0, f_1, \ldots, f_{m-1}) \mapsto f_0$, that extracts our final answer from the class $0 \pmod m$. All that remains are the technical details.

Theorem 3.1. *There exist two unary operations, A_1 and A_2, each of which is primitive recursive in its input, and the closure under these operators using* any *single unary primitive recursive function, F_0, is the set of all unary primitive recursive functions.*

Proof. Let A_1 be the operation $f \mapsto 0^\frown f$. When $f(0) < 2$, define A_2 by the rule

$$A_2(f)(n) = \begin{cases} f(1)+1 & \text{if } f(0) = 0 \text{ and } n = 0,\\ f(n+1) & \text{if } f(0) = 0 \text{ and } n \neq 0,\\ 0 & \text{if } f(0) = 1. \end{cases}$$

Note that $A_2^2A_1^2(F_0)$ is the constant function $\overline{0}$.

Now, given any constructed function f, we claim that we can prepend any natural number. Clearly, $A_1(f) = 0^\frown f$ gives us our base case. Supposing, inductively, that $m^\frown f$ has been constructed, then $A_2A_1(m^\frown f) = (m+1)^\frown f$ is also constructible. When plugging a function into A_2, we will treat the first entry as a code which tells us how to proceed.

Let $f = (a_0, a_1, \ldots)$ be an arbitrary function. For any $c \geq 2$, we define $A_2(c^\frown f) = (b_0, b_1, \ldots,)$ as follows. First, fix

$$m = \left\lfloor \frac{c-2}{7} \right\rfloor + 2.$$

The "-2" comes from the fact that A_2 has special cases when $c = 0, 1$. The "$+2$" comes from the fact that we want our modulus to satisfy $m \geq 2$. The "7" comes from the fact that we have seven operations to encode. Thus, there are multiple cases to consider according to the value of c; the cases will naturally correspond to the operations discussed in the paragraphs preceding the statement of Theorem 3.1.

Case 1: $c \equiv 0 \pmod 7$. In this case, set

$$b_n = \begin{cases} a_{n+m-1} & \text{if } n \equiv 0 \pmod m,\\ a_{n-1} & \text{otherwise.} \end{cases}$$

In other words, after dropping the initial coding number c from $c^\frown f$, we treat f as an m-tuple of functions, and apply the right shift operation.

Case 2: $c \equiv 1 \pmod 7$. Set

$$b_n = \begin{cases} a_{n+1} & \text{if } n \equiv 0 \pmod m, \\ a_{n-1} & \text{if } n \equiv 1 \pmod m, \\ a_n & \text{otherwise.} \end{cases}$$

Case 3: $c \equiv 2 \pmod 7$. Set

$$b_n = \begin{cases} F_1(n/m) & \text{if } n \equiv 0 \pmod m, \\ a_n & \text{otherwise.} \end{cases}$$

Case 4: $c \equiv 3 \pmod 7$. Do exactly as in the previous case, except replacing F_1 with F_2.

Case 5: $c \equiv 4 \pmod 7$. Set

$$b_n = \begin{cases} g^{\square}(n/m) & \text{if } n \equiv 0 \pmod m, \\ a_n & \text{otherwise,} \end{cases}$$

where g is the function defined by the rule $g(n) = a_{mn}$. (In other words, g is the function currently residing in the $0 \pmod m$ entries of f.)

Case 6: $c \equiv 5 \pmod 7$. Set

$$b_n = \begin{cases} a_{m \cdot a_{n+1}} & \text{if } n \equiv 0 \pmod m, \\ a_n & \text{otherwise.} \end{cases}$$

Case 7: $c \equiv 6 \pmod 7$. Set

$$b_n = a_{mn}.$$

We have now completely defined A_2, on an arbitrary input, and it is primitive recursive in its input. Starting with the zero sequence, then by repeatedly prepending the proper number c, we can pass through these cases and simulate the formation of any expression in the symbols $\{F_1, F_2, {}^{\square}, \circ\}$. Thus, we can form every unary primitive recursive function. Furthermore, since A_1 is primitive recursive in its input, as is A_2, then starting with a primitive recursive function, F_0, we can only construct primitive recursive functions this way. □

This theorem applies to many other classes, C, of unary computable functions, besides the primitive recursive functions. Indeed, as long as (1) the operation $f \mapsto 0^\frown f$ is C-computable in its input, (2) the class C allows for case analysis, (3) standard arithmetic operations are allowed, to handle passage in and out of congruence classes, and (4) the class C has some finite definitional scheme (using operations of any arity), then minor modifications of the proof above gives us a definitional scheme for C using two unary operations. In particular, adding a μ-minimization operation to the list of cases defining A_2, we have:

Corollary 3.2. *There exist two unary operations, A_1 and A_2, the first is primitive recursive in its input, the second is partial computable in its input, and the closure under these operators using* any *single unary partial computable function, F_0, is the set of all unary partial computable functions.*

Not surprisingly, replacing the two unary operations with a single binary operation is extremely easy.

Proposition 3.3. *There is a binary operation, A, primitive recursive in its inputs, such that the closure under this operator using any unary primitive recursive initial function, F_0, is the set of all unary primitive recursive functions.*

Proof sketch. Define $A(f,g)$ by the rule

$$A(f,g)(n) = \begin{cases} g(n)+1 & \text{if } f(0) \leq g(0), \\ F_1(n) & \text{if } f(0) = g(0)+1, \\ F_2(n) & \text{if } f(0) = g(0)+2, \\ g^{\square}(n) & \text{if } f(0) = g(0)+3, \\ \max(f(g(n)) - g(0) - 4, 0) & \text{otherwise.} \end{cases}$$

The remaining details are left to the reader. □

Some care has to be taken when generalizing this proposition to the unary partial computable functions. An arbitrary initial function no longer suffices, due to issues with the output ∞. In particular, to currently evaluate $A(F_0, F_0)$ we must be able to compute $F_0(0)$. One workaround is to take $F_0 = \overline{0}$ and redefine A by the rule

$$A(f,g)(n) = \begin{cases} g(0)+1 & \text{if } f(0) = 0 \text{ and } n = 0, \\ g(n) & \text{if } f(0) = 0 \text{ and } n \neq 0, \\ (0^\frown g)(n) & \text{if } f(0) = 1, \\ F_1(n) & \text{if } f(0) = 2, \\ F_2(n) & \text{if } f(0) = 3, \\ g^{\square}(n) & \text{if } f(0) = 4, \\ \text{minimum } m \in \mathbb{N} \text{ such that } g(m) = n & \text{if } f(0) = 5, \\ f(g(n)+1) & \text{if } f(0) = 6, \\ \infty & \text{otherwise.} \end{cases}$$

The "+1" in the next-to-last case lets us avoid noncomputability issues when defining compositions, as follows. First, we can construct the function $1^\frown\overline{0} = A(\overline{0},\overline{0})$. Thus, from any constructed function g we can construct $0^\frown g = A(1^\frown\overline{0}, g)$, and then we can define $m^\frown g$ for each $m \in \mathbb{N}$, using the recursive formula $(m+1)^\frown g = A(\overline{0}, m^\frown g)$. Thus, given a pair of constructed functions f and g, we can form $f \circ g = A(6^\frown f, g)$. Note that this equality works even if f is undefined at 0.

(A more general collection of initial functions can be made to work in this case, by modifying A accordingly. Essentially, any function F_0—except the empty partial function $(\infty, \infty, \ldots)$—will work, together with an appropriate binary operation, but that operation may now depend on F_0.)

While the arity-complexity for unary partial computable functions is now fully understood, the case for unary primitive recursive functions is not quite settled. In particular, one cannot use the special properties of partial functions, as was done in Theorem 2.1. Thus, we ask:

Question 3.4. Is there a definitional scheme for the unary primitive recursive functions that has only a single unary operation?

Another avenue for future research would be to see if the definitional schemes of J. Robinson or Severin could be simplified by adjoining the unary operation $f \mapsto 0^\frown f$, as this operation proved very useful in the arguments of this paper.

4. Acknowledgements

I thank Andreas Blass and Noah Schweber for answering questions related to this work. I also thank the two anonymous referees for very careful reading of the manuscript, and for suggestions that improved the quality of this paper.

References

1. Richard M. Friedberg, *Three theorems on recursive enumeration. I. Decomposition. II. Maximal set. III. Enumeration without duplication*, J. Symbolic Logic **23**(3):309–316 (1958). MR 109125
2. N. Georgieva, *Another simplification of the recursion scheme*, Arch. Math. Logik Grundlag. **18**(1):1–3 (1976/1977). MR 485271
3. M. D. Gladstone, *A reduction of the recursion scheme*, J. Symbolic Logic **32**(4):505–508 (1967). MR 224460
4. M. D. Gladstone, *Simplifications of the recursion scheme*, J. Symbolic Logic **36**(4):653–665 (1971). MR 305993
5. Martin Kummer, *An easy priority-free proof of a theorem of Friedberg*, Theoret. Comput. Sci. **74**(2):249–251 (1990). MR 1067521
6. Julia Robinson, *General recursive functions*, Proc. Amer. Math. Soc. **1**(6):703–718 (1950). MR 38912
7. Julia Robinson, *A note on primitive recursive functions*, Proc. Amer. Math. Soc. **6**(4):667–670 (1955). MR 73536
8. Raphael M. Robinson, *Primitive recursive functions*, Bull. Amer. Math. Soc. **53**(10):925–942 (1947). MR 22536
9. Raphael M. Robinson, *Primitive recursive functions. II*, Proc. Amer. Math. Soc. **6**(4):663-666 (1955). MR 73535
10. Daniel E. Severin, *Unary primitive recursive functions*, J. Symbolic Logic **73**(4):1122-1138 (2008). MR 2467207
11. István Szalkai, *On the algebraic structure of primitive recursive functions*, Z. Math. Logik Grundlag. Math. **31**(35–36):551-556 (1985). MR 813895

Department of Mathematics, Brigham Young University, Provo, UT 84602, USA
Email address, Corresponding author: pace@math.byu.edu