Yehuda Schwartz  
George Tourlakis  

PURE ITERATION AND SUBSTITUTION AS THE BASIS OF COMPUTABILITY ∗

Abstract

It is known that, in the presence of pairing/projection functions, (pure) iteration can simulate primitive recursion [5, 6]. This fact implies that the class of primitive recursive functions, $\mathcal{PR}$, can be obtained as the closure of a small set of initial functions under substitution and pure iteration—as long as the floor of the square root is included as an initial function to bootstrap the construction of pairing/projection functions—or from just the successor and predecessor functions if we add bounded search to the a priori available operations. In Section 2 of the paper we show that neither the inclusion of square root nor of bounded search are necessary to build $\mathcal{PR}$ from the successor and predecessor. In Section 3 we show that the class of partial recursive functions, $\mathcal{P}$, can be obtained as the closure of $\mathcal{PR}$ under the operation of infinite (pure) iteration.

Keywords: Iteration, primitive recursive functions, partial recursive functions, limits of function sequences.

1. Introduction

A well known approach for obtaining the class$^1$ of primitive recursive functions, or $\mathcal{PR}$, is found in Grzegorczyk [2] and Péter [4], where one

∗This research was partially supported by NSERC grant No. 8250

$^1$As is normal in non-formalised computability, the terms “set” and “class” are synonymous.
starts with the *initial set* \( I = \{ \lambda x.x, \lambda x.x + 1 \} \) and then proceeds to obtain its *closure*\(^2\) under the operations of substitution and primitive recursion. In fact, \([2]\) stratifies the construction, using instead *bounded* primitive recursion, and builds \( \mathcal{PR} \) by stages, as a hierarchy of increasingly more inclusive classes \( \mathcal{E}^n \), for \( n \geq 0 \), where each \( \mathcal{E}^n \) is the closure of \( I^n = \{ \lambda x.x, \lambda x.x + 1, \lambda xy.g_n(x, y) \} \) under substitution and bounded primitive recursion, where \( \lambda xy.g_n(x, y) \) is a version of the Ackermann function.

It has been known as part of the folklore of recursion theory that *pure iteration*, defined below, can simulate primitive recursion as long as we have pairing functions and their projections (cf. \([5, 2, 4, 6]\)). This gave rise to characterisations of \( \mathcal{PR} \) that use iteration – rather than the “full” primitive recursion – and substitution.

In \([5]\) the initial set is \( \{ \lambda x.x + 1, \lambda xy.x + y, \lambda xy.x - y, \lambda x.\lfloor \sqrt{x} \rfloor \} \), and the pure iteration is restricted to be evaluated at 0. In \([6]\) the initial set is \( \{ \lambda x.x + 1, \lambda x.x - 1 \} \) but iteration is allowed to be evaluated on any input and, moreover, bounded search is used as a primitive operation. Both approaches rest their case once they manage to construct pairing/projection functions with the provided tools.

In the present paper we retain the trivial initial set of predecessor/successor of \([6]\) but show that the bounded search need not be primitive; it is a derived operation. Thus we obtain a partial improvement, in different directions (initial function set vs. admitted operations), over the two aforementioned sources.

In Section 3 we show that infinite iteration can supplant unbounded search and thus, along with substitution and finite iteration, can define the class of all partial recursive functions.

**Definition 1.1.** (Substitution \([2]\)) The following construction rules are called the *rules of substitution*, where we write \( \vec{x}_n \) (or \( \vec{x} \), if \( n \) is understood or unimportant) for \( x_1, \ldots, x_n \).

(a) Substitute the \( i^{th} \) variable in \( g(\vec{x}_n) \) with \( f(\vec{y}_m) \); that is, from the functions \( \lambda \vec{x}_n.g(\vec{x}_n) \) and \( \lambda \vec{y}_m.f(\vec{y}_m) \) we can obtain the following function on \( \mathbb{N}^{n+m-1} \):

\[
\lambda x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, \vec{y}_m.g(x_1, x_2, \ldots, x_{i-1}, f(\vec{y}_m), x_{i+1}, \ldots, x_n)
\]

\(^2\)That is, the smallest set that contains \( I \) and is closed under the stated operations.
Pure Iteration and Substitution as the Basis of Computability

(b) Substitute \( x_i \) in \( g(⃗x_n) \) with 0 to obtain the function:
\[
\lambda x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n.g(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)
\]

c) Interchange variables \( x_i \) and \( x_j \) in \( g(⃗x_n) \) to obtain the function:
\[
\lambda x_1, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_n.g(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_n)
\]

d) Identify two different variables \( x_i \) and \( x_j \) in \( g(⃗x_n) \) to obtain the following function on \( \mathbb{N}^{n-1} \):
\[
\lambda ⃗x_{j-1}, ⃗y, ⃗x_j+1, \ldots, ⃗x_n.g(⃗x_{i-1}, ⃗y, ⃗x_i+1, \ldots, ⃗x_{j-1}, ⃗y, ⃗x_i, ⃗y, ⃗x_j+1, \ldots, ⃗x_n)
\]

e) Introduce dummy variables \( ⃗y \) to \( \lambda ⃗x_n.g(⃗x_n) \) to obtain the function:
\[
\lambda ⃗x_n. ⃗y.g(⃗x_n)
\]
where \( ⃗y \) is a vector of any (fixed) length.

Definition 1.2. (The Schema of Primitive Recursion) We say that a class \( C \) of number theoretic functions is closed under \textit{primitive recursion} if whenever it contains the functions \( \lambda ⃗y.m.h(⃗y_m) \) and \( \lambda x⃗y.m.z.G(x, ⃗y_m, z) \) then it must also contain a function \( \lambda x⃗y.m.f(x, ⃗y_m) \) satisfying the following recurrence, for all \( x, ⃗y_m \):
\[
\begin{cases}
  f(0, ⃗y_m) = h(⃗y_m) \\
  f(x + 1, ⃗y_m) = G(x, ⃗y_m, f(x, ⃗y_m))
\end{cases}
\]

Remark 1.3. Notice that by rule (e) of substitution, if \( \lambda z.g(z) \in \mathcal{PR} \) then so is the function \( G = \lambda x⃗y_m.z.G(x, ⃗y_m, z) \). Therefore, given \( h = \lambda y.y \) and \( \lambda z.g(z) \), both in \( \mathcal{PR} \), we may use \( G \in \mathcal{PR} \) obtained as above to justify that the function \( f \) defined below is in \( \mathcal{PR} \), since the schema (2) can be rewritten as (1):
\[
\begin{cases}
  f(0, y) = y \\
  f(x + 1, y) = g(f(x, y))
\end{cases}
\]

This observation leads us to the next schema, called the schema of Pure Iteration.

Definition 1.4. (The Schema of Pure Iteration [5]) We say that a class \( C \) of number theoretic functions is closed under \textit{pure iteration} if whenever
it contains the function $\lambda z.g(z)$, then it must also contain a function $f$ satisfying the following:

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

The name pure iteration is due to the fact that $f(x, y) = g^x(y)$ (where, by definition, $g^0(y) = y$, for all $y$) and thus $f(x, y)$ is the “$x$th iteration of $g(y)$”.

Let us now define another class of functions called $\text{PI}$:

**Definition 1.5.** (The Class $\text{PI}$) $\text{PI}$ is the smallest class of number theoretic functions containing $\{\lambda x.x \downarrow 1, \lambda x.x + 1\}$ and closed under the operations of substitution and pure iteration.

Now, Remark 1.3, and the known fact that $\lambda x.x \downarrow 1$ is in $\text{PR}$,\(^3\) imply that $\text{PI} \subseteq \text{PR}$. On the other hand, it is also known [5, 4, 6] that in the presence of pairing functions the schema of pure iteration is as strong as that of primitive recursion. Therefore, showing that $\text{PI}$ does contain such functions is tantamount to showing that $\text{PI} = \text{PR}$.

### 2. The Existence of Pairing Functions in $\text{PI}$

The proof for the following fact can be found in [6].

**Lemma 2.1.** The pairing function $J(x, y) = 2^{x+y+2} + 2^{y+1}$ is in $\text{PI}$.

Consequently, we need only to show that $\text{PI}$ contains the projections of $J$, i.e., it contains functions $K$ and $L$ such that $K(J(x, y)) = x$ and $L(J(x, y)) = y$, for all $x$ and $y$. However, since it can be easily ascertained from the proof in [6] that the existence of such projections hinges upon the presence of the function $\lambda x.\lfloor \log_2(x) \rfloor$ in $\text{PI}$, we need only prove the following:

**Theorem 2.2.** The function $\lambda x.\lfloor \log_2(x) \rfloor$ is in $\text{PI}$.

---

\(^3\)Indeed, $0 \downarrow 1 = 0$ and $x + 1 \downarrow 1 = x$. 
Proof. We shall derive, in $\mathcal{PI}$, a succession of functions culminating in $\left\lfloor \log_2(x) \right\rfloor$.

(a) $\lambda xy. \min(x, y) : \min(x, y) = x \div (x \div y)$
(b) $\lambda xy. \max(x, y) : \max(x, y) = x + (y \div x)$
(c) $\lambda x. 1$ and $\lambda x. 2$ : The first one from the function $s = \lambda x. x + 1$ and the rules of substitution (b). The second one by composing $s$ with $\lambda x. 1$.
(d) $\lambda x. (x \mod 2)$ : Define $t(y) = 1 \div y$. Then $t^0(0) = x \mod 2$. Indeed, $t(x)$ defines the sequence of outputs $1, 0, 1, 0, ...$. But note that $t^0(0) = 0$ and $t^{x+1}(0) = 1 \div t^x(0)$, that is, $t^x(0)$ outputs the sequence $0, 1, 0, 1, 0, ...$
(e) $\lambda x. (x \mod 3)$ : Define $g(x) = \min(x + 1, (2 \div x) + (2 \div x))$. Then, $g^0(0) = 0 \mod 3$
$g^1(0) = g(0) = \min(1, 4) = 1 = 1 \mod 3$
$g^2(0) = g(1) = \min(2, 2) = 2 = 2 \mod 3$
$g^3(0) = g(2) = \min(3, 3) = 0 = 0 \mod 3$
And so on. Hence, $g^x(0) = x \mod 3$.
(f) $\lambda x. \left\lfloor x/2 \right\rfloor$ : Define $f(x) = x + x \mod 3$.
Let us examine the iterations of $f$ starting from 1:
$1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 10 \rightarrow 10 \rightarrow ...$
We can see that the value of $f$ increases by 3 every 2 iterations. Thus, we should expect that after $x$ iterations the value of $f$ will have increased to about $\left\lfloor 3x/2 \right\rfloor$.
And indeed, let $x = 2k + r$ where $k = \lfloor x/2 \rfloor$ and $r = x \mod 2$.
Note that $(x + 1)/2 = k + r$ and $(x + 1) \mod 2 = 1 - r$, thus, $x + 1 = 2(k + r) + 1 - r$. Next, we show that $f^x(1) = 3k + 1 + r$ : It is immediate for $x = 0$, while for $x + 1, f^{x+1}(1) = f(f^x(1)) = f(3k + 1 + r) = (3k + 1 + r) + (1 + r) = 3(k + r) + 1 + (1 - r)$.
Therefore, $f^x(1) \div x = 1 = (3k + 1 + r) - (2k + r) - 1 = k = \lfloor x/2 \rfloor$.
And so, $\lambda x. \lfloor x/2 \rfloor$ is in $\mathcal{PI}$.

\footnote{We will only build functions whose presence in $\mathcal{PI}$ had not been demonstrated in [6].}
(g) \( \lambda xy \cdot x \cdot \text{sg}(y) \) : Let \( h(y) = y + \text{sg}(y) \), then \( h^2(y) \div y \) is our function.\(^5\) Indeed, if \( y = 0 \) then \( h(y) = 0 \) and \( h^2(y) \div y = 0 \); which is as it must be for emulating \( x \cdot \text{sg}(y) \) correctly. If \( y > 0 \), then \( \text{sg}(y) = 1 \), hence \( h^2(y) = x + y \), thus \( h^2(y) \div y = x \). This again tracks the expression \( x \cdot \text{sg}(y) \) well.

(h) \( \lambda xy \cdot (1 \div \text{sg}(y)) \) : From the fact that \( x \cdot (1 \div \text{sg}(y)) = x \div x \cdot \text{sg}(y) \).

(i) \( \lambda x. \text{if } \exists n(2^n = x) \text{ then } 1 \text{ else } 0 \) :

\[
G(x) = \begin{cases} 
1 & \text{if } x = 1 \\
[x/2] & \text{if } x \text{ is even} \\
0 & \text{otherwise}
\end{cases}
\]

Thus \( G \in \mathcal{P} \mathcal{I} \) since \( G(x) = [x/2](1 \div \text{sg}(x(\mod \ 2))) + \min(x, 2 \div x) \).

What is \( G^2(x) ? \)

If \( x \geq 2 \) and \( x = 2^n \) for some \( n \) \( \neq x \) then after \( n \) iterations we reach 1 where we remain for the last \( x - n \) iterations. i.e.

\[
2^n \overset{n \text{ iterations}}{\longrightarrow} G \overset{2^{n-1}}{\longrightarrow} \cdots \overset{2}{\longrightarrow} G_2 \overset{1}{\longrightarrow} G_1 \overset{1}{\longrightarrow} \cdots \overset{1}{\longrightarrow}
\]

If, on the other hand, \( x = 2^n \cdot m \) for some \( m \) \( > 1 \) odd, then after \( n \) \( \neq x \) iterations we reach \( m \) and then move to 0 where we remain for the last \( x - n - 1 \) iterations. i.e.

\[
2^n \cdot m \overset{n \text{ iterations}}{\longrightarrow} G \overset{2^{n-1}}{\longrightarrow} \cdots \overset{2 \cdot m}{\longrightarrow} G_2 \overset{m}{\longrightarrow} 0 \overset{0}{\longrightarrow} 0 \overset{0}{\longrightarrow} \cdots \overset{0}{\longrightarrow}
\]

Therefore, \( G^2(x) = \text{if } \exists n(2^n = x) \text{ then } 1 \text{ else } 0 \), that is, \( \lambda x. G^2(x) \) is the characteristic function of the predicate “\( x \) is a power of 2”.

(j) \( \lambda x. [\log_2(x)] \) : Define

\[
H(y) = y + \begin{cases} 
1 & \text{if } y \text{ is not a power of } 2 \\
2 & \text{otherwise}
\end{cases}
\]

that is, \( H(y) = y + \max(1, 2 \cdot G^m(y)) \), hence \( H \in \mathcal{P} \mathcal{I} \).

---

\(^5\) \( \text{sg}(y) = 1 \div (1 \div y) \).
Here are few iterations of $H$ starting from 0.

$0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 13 \rightarrow \ldots$

What is the relation between $H^x(0)$ and $\lfloor \log_2(x) \rfloor$? To begin with, $\lfloor \log_2(x) \rfloor$ represents the number of integers less than or equal to $x$ that are positive powers of 2. Now, when starting from 0, $H$ will increase its value by 1 if its input is not a power of 2 and by 2 if it is. Thus, the $\lfloor \log_2(x) \rfloor$ powers of 2 less or equal to $x$ contribute twice their share and we should expect $H^x(0)$ to overshoot $x$ by about $\lfloor \log_2(x) \rfloor$, which would mean that $\lfloor \log_2(x) \rfloor$ is approximately $H^x(0) - x$.

The precise relationship between $\lfloor \log_2(x) \rfloor$ and $H^x(0)$ is uncovered by the following lemmata:

**Lemma 2.3.** $H^{2^n}(0) = 2^n + n$ for all $n$.

**Proof.** Induction on $n$.

$n = 0 : H^{2^0}(0) = 1 = 2^0 + 0$.

$n + 1 : H^{2^{n+1}}(0) = H^{2^n + 2^n}(0) \overset{\text{I.H.}}{=} H^{2^n}(2^n + n) = H^{2^n - 1}(2^n + n + 1) = H^{2^{n-2}}(2^n + n + 2) = \ldots = H^{2^{n-(2^n - n)}}(2^n + n + 2^n - n) = H^n(2^{n+1})$.

Now,

$H^n(2^{n+1}) \overset{\text{I.H.}}{=} H^{n-1}(2^{n+1} + 2) = H^{n-2}(2^{n+1} + 3) = H^{n-3}(2^{n+1} + 4) = \ldots = H(2^{n+1} + n) = 2^{n+1} + n + 1$. ■

**Lemma 2.4.** For all $n$ and $x$, if $2^n \leq x < 2^{n+1} - 1$, then $H^x(0) = x + n$.

**Proof.** Induction on $x$. The case of $x = 1 = 2^0$ is by Lemma 2.3.

On the obvious induction hypothesis, consider $x + 1$ such that, for some $n$, $2^n \leq x + 1 \leq 2^{n+1} - 1$.

If $x + 1 = 2^n$, then Lemma 2.3 rests the case without the I.H.

If not — and thus $x + 1 \geq 3$ — then $H^{x+1}(0) = H(H^x(0)) \overset{\text{I.H.}}{=} H(x + n)$, the I.H. being applicable since $2^n < x + 1 \leq 2^{n+1} - n$ implies $2^n \leq x < 2^{n+1} - n$. But we have also $2^n < x + n < 2^{n+1} - n$ — the left “<” is due to $x + 1 \geq 3$, which guarantees $n > 0$. Thus $H(x + n) = (x + 1) + n$. ■

**Lemma 2.5.** For all $n$ and $x$, if $2^n \leq x < 2^{n+1}$ then $2^n + x \leq 2^{n+1} - n - x$. 

Proof. Since $2^n \leq x \leq 2^{n+1} - 1$ then $2^n + x \leq 2^x \leq 2^{n+1} + x - 2^x$. And since it can easily be proved that $x + \lfloor \log_2(x) \rfloor \leq 2^x$ for all $x$, we get that $2^{n+1} + x - 2^x \leq 2^{n+1} + x - n - x$.

Lemma 2.6. For all $x$, $\lfloor \log_2(x) \rfloor = (H_2^x(0) \div 2^x) \div x$.

Proof. It is immediate for $x = 0$. If $x \geq 1$ then there exists an $n$ such that $2^n \leq x < 2^{n+1}$ and so, by (2.5), $2^n + x \leq 2^x \leq 2^{n+1} + x - n - x$, and by (2.4), $H_2^x(0) = 2^x + x + \lfloor \log_2(x) \rfloor$ or $\lfloor \log_2(x) \rfloor = (H_2^x(0) \div 2^x) \div x$.

These lemmata conclude the proof that $\lambda x. \lfloor \log_2(x) \rfloor \in PI$, and thus $PI = PR$.

3. Infinite Iteration and The Class $PI_\infty$

Let us define the notion of closure under infinite iteration on total functions. Recall that a sequence $x_n$ of numbers from $\mathbb{N}$ converges to $a$, in symbols $\lim_{n \to \infty} x_n = a$, if the set $\{n : x_n \neq a\}$ is finite.

Given a sequence of total functions $\lambda x. f(n, x)$ we can define a partial function $h$ by $h = \lambda x. \lim_{n \to \infty} f(n, x)$. Of course, $h(x)$ is undefined whenever the limit $\lim_{n \to \infty} f(n, x)$ does not exist.

Definition 3.1.[Infinite Iteration] We say that a class $C$ of number theoretic functions is closed under infinite iteration if whenever it contains a total $\lambda y. g(y)$, then it must also contain the function $\lambda x. \lim_{x \to \infty} g^x(y)$.

Recall now that one of the ways to define the class of partial recursive functions, $P$, is the following:

Definition 3.2. $P$ is the closure of $\{\lambda x. x + 1, \lambda x. x \div 1\}$ under substitution, pure iteration and unbounded search ($\tilde{\mu}$) on total functions.

The above characterisation of $P$ is not surprising as it readily follows from $PI = PR$ of Section 2, via Kleene’s Normal Form theorem. It appears in a slightly different form in [1, 3, 6] (cf. [6], p. 116 and note that full

---

6$x \leq 2^x$ (for all $x$) implies $2^x 2^x \leq 2^{2x}$. Thus also $2^x x \leq 2^{2x}$ and hence $x + \log_2 x \leq 2^x$. But $[\log_2(x)] \leq \log_2 x$.

7By definition, $[\log_2(0)] = 0$; cf. [6].
addition and subtraction are defined by iteration from the successor and predecessor functions.)

We are reminded here of the three main versions of search: For any total \( \lambda y \vec{x}.g(y, \vec{x}) \), the expression \((\mu y)g(y, \vec{x})\) denotes \(\min\{y : g(y, \vec{x}) = 0\}\) and is undefined when the minimum does not exist. That is, it denotes the operation of unbounded search. Kleene’s unbounded search can be applied on a partial recursive \( \lambda y \vec{x}.g(y, \vec{x}) \) without prior knowledge that it is total (knowledge that, as is well known, cannot be obtained algorithmically from a program of \(g\)): \((\mu y)g(y, \vec{x})\) stands for \(\min\{y : g(y, \vec{x}) = 0 \land (\forall z < y)(g(z, \vec{x}) \text{ is defined})\}\). It is undefined if the minimum is. It is trivial that for total \(g\), \((\tilde{\mu} y) g(y, \vec{x}) = (\mu y) g(y, \vec{x})\) since the part after “\(\land\)” is true for all \(\vec{x}\). Lastly, bounded search on total functions is denoted by the expression \((\mu y < z) g(y, \vec{x})\), given by

\[
(\mu y < z) g(y, \vec{x}) = \begin{cases} 
\min\{y : y < z \land g(y, \vec{x}) = 0\} & \text{if } (\exists y < z) g(y, \vec{x}) = 0 \\
\text{z otherwise}
\end{cases}
\]

It is clear that \((\tilde{\mu} y) g(y, \vec{x}) = \lim_{z \to \infty} (\mu y \leq z) g(y, \vec{x})\). Since \(\mathcal{P}I\) is closed under bounded search, this raises the question of what is the relation between two different extensions of \(\mathcal{P}I\): One, where we add closure under the operation of unbounded search on total functions (at once seen to result in \(\mathcal{P}\)), two, where we add closure under the operation of infinite iteration.

To explore this question, let us first introduce the class \(\mathcal{P}I_\infty\).

**Definition 3.3.** The class \(\mathcal{P}I_\infty\) is the closure of \(\{\lambda x.x + 1, \lambda x.x - 1\}\) under substitution, pure iteration and infinite iteration (on total functions).

We claim that \(\mathcal{P} = \mathcal{P}I_\infty\).

**Lemma 3.4.** \(\mathcal{P}I_\infty \subseteq \mathcal{P}\)

**Proof.** Given the above remarks we only need to show that \(\mathcal{P}\) is closed under infinite iteration.\(^8\) Let \(\lambda y.g(y) \in \mathcal{R}\), and set \(f(x, y) = g^x(y)\). Then

\[
\lim_{x \to \infty} g^x(y) = f(\tilde{x}(g^{x+1}(y) = g^x(y)), y).
\]

\(^8\)Definition 3.1, applied to the context of \(\mathcal{P}\), requires that infinite iteration be applied to functions of its subclass \(\mathcal{R}\) of total (recursive) functions.
The next lemma uses coding. Since $\mathcal{PI} = \mathcal{PR}$ we have (in $\mathcal{PI}_\infty$) onto pairing functions,\(^9\) from which, for any fixed $n$, we can define primitive recursive sequence-coding, $\lambda\vec{x}_n.\langle\vec{x}_n\rangle$, and decoding, $\lambda z.(z)_i$, for $i = 0, 1, \ldots, n - 1$, functions. The ambiguity in the notation “$(z)_i$” is intentional for readability and is removed by the context (which sets the value of $n$). Ontoness allows the identity $\langle(\langle z \rangle)_0, \ldots, (\langle z \rangle)_{n-1}\rangle = z$, which along with the (implied by 1-1-ness) identities $(\langle \vec{x}_n \rangle)_i = x_{i+1}$, for $i = 0, \ldots, n - 1$ allow us to say things such as “if the functions $g$ and $f$—of two and $n + 1$ variables respectively— satisfy, for all $z,x$, $g(z,\langle \vec{x}_n \rangle) = f(z,\vec{x}_n)$, then $f$ is in $\mathcal{PI}_\infty$ iff $g$ is”, and also, “define the single-variable function, $f$, for all $x,y,z$ by $f(\langle x, y, z \rangle) = \ldots$”.

**Lemma 3.5.** $\mathcal{P} \subseteq \mathcal{PI}_\infty$.

**Proof.** We will show that $\mathcal{PI}_\infty$ is closed under unbounded search on total functions. So let $\lambda y \vec{x}_m.f(y,\vec{x}_m) \in \mathcal{PI}_\infty$ be total. First, since the function $\vec{f}$, defined by $\vec{f}(y,\langle \vec{x}_n \rangle) = f(y,\vec{x}_n)$, is in $\mathcal{PI}_\infty$ we can assume without loss of generality that $m = 1$. For every $n,x,y$ define the single-variable function $U$ by:

$$U(\langle n, x, y \rangle) = \langle (n+1) \text{sg}(n) \text{sg}(f(n-1,x)), x, y + \text{sg}(n) \text{sg}(f(n-1,x)) \rangle$$

It readily follows that $U \in \mathcal{PR} \subseteq \mathcal{PI}_\infty$. By inspection, for all $n$ and all $x$, we have

$$U(\langle 0, x, n \rangle) = \langle 0, x, n \rangle \quad \text{and, for } n > 0,$$

$$U(\langle n, x, n - 1 \rangle) = \begin{cases} 
(\langle n+1, x, n \rangle & f(n-1,x) \neq 0 \\
(0, x, n - 1) & f(n-1,x) = 0 
\end{cases}$$

Let us set $r(x,z) = \begin{cases} 
0 & (\mu y < z)f(y,x) < z \\
z + 1 & (\mu y < z)f(y,x) = z 
\end{cases}$

We now claim that

$$U^z(\langle 1, x, 0 \rangle) = \langle r(x,z), x, (\mu y < z)f(y,x) \rangle \quad \text{(1)}$$

**Proof of (1).** For $z = 0$, we know that $(\mu y < 0)f(y,x) = 0$ and so $r(x,0) = 1$ and $U^0(\langle 1, x, 0 \rangle) = \langle 1, x, 0 \rangle = \langle r(x,0), x, (\mu y < 0)f(y,x) \rangle$.

\(^9\)E.g., $\lambda xy.\lfloor(x+y)(x+y+1)/2\rfloor + x.$
For \( z+1, U^{z+1}(⟨1, x, 0⟩) = U((r(x, z), x, (μy < z)f(y, x))). \) There are three cases:

(a) \( r(x, z) = 0. \) Then \( (μy < z + 1)f(y, x) = (μy < z)f(y, x) \) and \( r(x, z+1) = r(x, z). \) Thus, \( U((r(x, z), x, (μy < z)f(y, x))) = U((0, x, (μy < z)f(y, x))) = 0, x, (μy < z)f(y, x) = 0, x, (μy < z + 1)f(y, x)). \)

(b) \( r(x, z) = z + 1 \& f(z, x) = 0. \) Then \( r(x, z + 1) = 0 \) and \( (μy < z + 1)f(y, x) = z. \) So, \( U((r(x, z), x, (μy < z)f(y, x))) = U((0, x, z)) = (r(x, z + 1), x, (μy < z + 1)f(y, x)). \)

(c) \( r(x, z) = z + 1 \& f(z, x) \neq 0. \) Then \( r(x, z + 1) = z + 2 \) and \( (μy < z + 1)f(y, x) = z + 1. \) So, \( U((r(x, z), x, (μy < z)f(y, x))) = U((z + 1, x, z)) = (z + 2, x, z + 1) = (r(x, z + 1), x, (μy < z + 1)f(y, x)). \)

End of proof of (1).

Therefore, \( \lim_{z \to \infty} U^z(⟨1, x, 0⟩) = \big\langle \lim_{z \to \infty} r(x, z), x, \lim_{z \to \infty} (μy < z)f(y, x) \big\rangle \)

and \( (\check{μ}y)f(y, x) = \big( \lim_{z \to \infty} U^z(⟨1, x, 0⟩) 3 \big) \)

References