On a Class of Reversible Primitive Recursive Functions and Its Turing-Complete Extensions

Published version:

DOI:10.1007/s00354-018-0039-1

Noname manuscript No.
(will be inserted by the editor)

On a class of Reversible Primitive Recursive Functions and its Turing-complete extensions

Luca Paolini · Mauro Piccolo · Luca Roversi

New Generation Computing (2018) 36:233–256
https://doi.org/10.1007/s00354-018-0039-1

Abstract Reversible computing is both forward and backward deterministic. This means that a uniquely determined step exists from the previous computational configuration (backward determinism) to the next one (forward determinism) and vice-versa. We present the Reversible Primitive Recursive Functions (RPRF), a class of reversible (endo-)functions over natural numbers which allows to capture interesting extensional aspects of reversible computation in a formalism quite close to that of classical Primitive Recursive Functions. The class RPRF can express bijections over integers (not only natural numbers), is expressive enough to admit an embedding of the Primitive Recursive Functions and, of course, its evaluation is effective. We also extend RPRF to obtain a new class of functions which are effective and Turing-complete, and represent all Kleene's μ-recursive functions. Finally, we consider reversible recursion schemes that lead outside the reversible endo-functions.

Keywords Reversible computing · Recursive permutations · Primitive Recursive Functions · Reversible Pairing · Recursion Theory

1 Introduction

Reversible computing is probably the most classical among the unconventional models of computing. Origins of reversible computing trace back to the study of entropy in physical systems [5]. An introductory survey about the presence

---

This paper is an extended version of the paper "A Class of Reversible Primitive Recursive Functions" published in the Proceedings of the 16th Italian Conference on Theoretical Computer Science (Firenze 9–11 September 2015), edited by Pierluigi Crescenzi and Michele Loreti (see [25]).

L. Paolini (luca.paolini@unito.it) · M. Piccolo · L. Roversi (luca.roversi@unito.it)
Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino

Abbreviation	Class of functions	Total
PRF	Primitive Recursive Functions	✓
JPRF	Injective Primitive Recursive Functions	✓
BPRF	Bijective Primitive Recursive Functions	✓
RPRF	Reversible Primitive Recursive Functions	✓
μRPF	μ-Recursive Partial Functions	×
μRRF	μ-Reversible Recursive Partial Functions	×

Table 1 Relevant classes of functions.

of reversible aspects inside classical computing is [28]. Examples of applications of reversible computing span from software verification to programming languages, through computer architectures, operating systems, databases, artificial intelligence, as well as several non-classical models of computing, like quantum computing and other formalisms for natural computing [12,8,27,26, 37,38,39].

Foundational studies on the notion of "reversible computation" have a long tradition but they have been chiefly devoted to the thermodynamics of Turing-machine computations [2,4,15]. A reversible Turing-machine is bi-directionally deterministic, i.e. both forward-deterministic, like a classical Turing-machine, and backward-deterministic. The backward-determinism allows to reverse the computation, so to step-by-step undo what a program has done, eventually recovering former configurations [2]. Systematic surveys on recursion-theoretic aspects of the reversible computation are [2,3].

Primitive Recursive and μ-Recursive Partial Functions. The present work proposes a recursion theory for reversible functions with the aim of identifying a function algebra of numerical functions closed under specific schemes [29,34].

We start recalling the distinguishing aspects of μRPF, the class of Kleene's μ-Recursive Partial Functions [17] and of PRF, the class of Primitive Recursive Functions which μRPF is an extension of. For easy of reference, Table 1 lists classes of functions we shall deal with.

Both μRPF and PRF balance intensional and extensional aspects. Intensionally, they essentially are programming languages with an informal but unambiguous semantics. Extensionally, μRPF deals with *partial functions*[1] and PRF with *total ones.*

Goals. Let us recall that the inverse f^{-1} of a function f is its relational reverse defined by reversing its underlying relation[2], viz. $(y, x) \in f^{-1}$ if and only if

[1] A relation between two sets A, B is a subset of the cartesian product $A \times B$. A relation is *functional* when $(a, b), (a, b') \in A \times B$ implies $b = b'$. A relation is *injective* when $(a, b), (a', b) \in A \times B$ implies $a = a'$. A relation is *total* whenever $a \in A$ implies that $b \in B$ exists such that $(a, b) \in A \times B$. A relation is *surjective* whenever $b \in B$ implies that $a \in A$ exists such that $(a, b) \in A \times B$. A function is a total functional relation. A partial function is a functional relation.

[2] The inverse of a partial function may not be functional. The inverse of a total function may not be total. However, restricted (and effective) operation of inversion can be defined also in such cases, e.g. see [22].

$(x, y) \in f$. We aim at emphasizing the prominent computational status that the operation of inversion can have in the reversible models of computation. We want to set up a formalism which identifies a sufficiently large class of first-order functions whose graphs can be effectively inverted inside the formalism itself. The inversion operation that we propose in this work takes great advantage from the compositional nature of the considered recursion-theoretic model.

Motivations for RPRF. Identifying the right class of total functions acting as the extensional model of reference is not immediate. Reversible Turing Machines compute *injective* μRPF [2,3].

This suggests to consider JPRF, the class of Injective Primitive Recursive Functions as extensional model of reference. Unfortunately JPRF is not closed under inversion. A function f exists such that its inverse f^{-1} is not in JPRF. An example is the successor `succ` on natural numbers. It belongs to JPRF but its inverse $\texttt{succ}^{-1}$ is undefined on 0 and does not belong to JPRF.

Replacing the class BPRF of all Bijective Primitive Recursive Functions for JPRF is not a solution despite BPRF is strictly smaller than JPRF:

Theorem 1 (Kuznekov [18]) *There is $f \in$ BPRF whose inverse f^{-1} does not belong to* PRF.

Proof Consider a total computable function whose rate of growth is too fast to be primitive recursive, e.g. consider the Ackermann-like function $A_0(x) = 2^x$ and $A_{n+1}(x) = \underbrace{A_n \ldots A_n}_{x}$ (1) (see [10, Exercise 3.2. p.57]). The Normal Form Theorem [17] ensures that a predicate T and a function U exist which are primitive recursive and such that $\phi_i(x) = U(\mu y[T(i, x, y)])$ for all program index i. We follow the suggestion of [33, Exercise 5.7, p.25]. We assume that e is the program index such that $\phi_e(x) = A_x(x)$. Hence, we can set $g(x) = \mu y[T(e, x, y)]$. Certainly g grows faster than ϕ_e because $U(y) \leq y$ for all y. Therefore $g(x)$ cannot be primitive recursive, but $g^{-1}(y)$ is because the set of primitive recursive functions is closed under bounded minimization. Let $g^{-1}(y)$ be defined as $(\mu x \leq y)[T(e, x, y)]$. The following primitive recursive function on $\mathbb{N}$ is a permutation:

$$f(y) = \begin{cases} 2g^{-1}(y) & \text{if } (\exists x \leq y)[T(e, x, y)], \\ 1 + 2(k-1) & \text{otherwise, where } k = \sum_{j=0}^{y} (\exists x \leq j)[1 - T(e, x, j)] \end{cases}$$

assuming that true is represented by 0. However its inverse is not primitive recursive because $f^{-1}(2x) = g(x)$ is not. $\square$

Corollary 1 BPRF *and recursive permutations are two different classes of functions.*

Since we cannot effectively enumerate BPRF, we might wonder if we can enumerate the whole set of recursive permutations. Two negative results are in

[31, Exercise 4-6, p.55]: (i) no effective and complete procedure can list a set of Gödel numbers that represent; and, (ii) the group of recursive permutations is not finitely generated. In addition, we prove a stronger negative result: no functional language can characterize all and only the recursive permutations.

Theorem 2 *Recursive permutations cannot be recursively enumerated.*

Proof Assume $\phi_0, \ldots, \phi_n, \ldots$ be a recursive enumeration of permutations. We aim to find a recursive permutation Ψ which is not in the list.

- Let us assume that there is $m \in \mathbb{N} \setminus \{0\}$ such that $\phi_0(0) \neq \phi_m(m)$. We apply a classical diagonalization argument and define Ψ by induction: $\Psi(0) = \phi_m(m)$ and $\Psi(n+1) = \min(\mathbb{N} \setminus \{\Psi(0), \ldots, \Psi(n), \phi_{n+1}(n+1)\})$. By definition: (i) $\Psi(0) \neq \phi_0(0)$ thus $\Psi \neq \phi_0$; and, (ii) for all $i \in \mathbb{N} \setminus \{0\}$, $\Psi(i) \neq \phi_i(i)$ thus $\Psi \neq \phi_i$.
- Otherwise, there is $d \in \mathbb{N}$ such that for all $i \in \mathbb{N}$, $\phi_i(i) = d$. The above diagonalization argument cannot apply. We define Ψ by induction: $\Psi(0) = \phi_0(0) = d$, $\Psi(1) = \phi_0(2)$ and $\Psi(m) = \min(\mathbb{N} \setminus \{\Psi(0), \ldots, \Psi(m-1)\})$, for every $m \geq 2$. Since ϕ_0 is a bijection, $\phi_0(2) \neq \phi_0(1)$; thus, $\Psi(1) \neq \phi_0(1)$ ensures that $\Psi \neq \phi_0$. Since ϕ_i is a bijection, $\phi_i(0) \neq d$ for each $i \geq 1$; thus, $\Psi(0) = d \neq \phi_i(0)$ ensures that $\Psi \neq \phi_i$ $(i \geq 1)$. □

Outline. The above observations led us to synthesize the class Reversible Primitive Recursive Functions (RPRF) [3] (Section 2) which includes bijections only, is closed under the effective meta-operation of inversion and which is PRF-complete, i.e. every $f \in$ PRF has a faithful counterpart in RPRF (Section 3). In fact, RPRF is also PRF-sound, i.e. every $f \in$ F has a faithful counterpart in RPRF. The interested reader can refer to [25].

The new parts of this work, as compared to [25], are Sections 4 and 5. The first one extends RPRF to the class μRRF which we prove is Turing-complete. I.e. every function of μRPF has its counterpart in μRRF. The second one discusses how to relax the constraint that forces every element in RPRF to have identical input and output arity. Section 6 concludes the work.

Acknowledgements. We must thank editors, reviewers and Felice Cardone who carefully read the preliminary work both commenting about its critical parts and rising stimulating questions.

2 Reversible Primitive Recursive Functions

We introduce *Reversible Primitive Recursive Functions* (RPRF), a class of total functions. RPRF operates on integers and not on natural numbers like primitive recursive functions do. The reason is that natural numbers do not form a group (endowed by inverses) with standard operations, as noted in

[3] The name "Reversible Primitive Recursive Functions" comes from [25] in order to underline its close correspondence between RPRF itself and Primitive Recursive Functions.

[21]. Another peculiar aspect of RPRF is that it is closed under inversion in an effective way.

Some preliminary steps are worth giving before formally introducing RPRF.

We use $\mathbb{Z}$ to denote the set of integers and $\mathbb{N}$ to denote the set of natural numbers[4]. Consider a function $f : X^n \to Y^m$ where X, Y are sets and $n, m \in \mathbb{N}$; we say that f is arity-respecting if $X = Y$ and $n = m$. For the sake of simplicity, we restrict RPRF to arity-respecting functions so to include permutations only. Finally, let $«\_,\_» : \mathbb{Z}^2 \to \mathbb{Z}$ be a given computable bijection whose definition details are irrelevant to our purposes[5].

Definition 1 (Base Reversible Primitive Recursive Functions) The class of *Base Reversible Primitive Recursive Functions* contains the following functions.

- Successor functions $\mathsf{S}_i(x_1, \ldots, x_i, \ldots, x_k) = (x_1, \ldots, x_i + 1, \ldots, x_k)$ and predecessor functions $\mathsf{P}_i(x_1, \ldots, x_i, \ldots, x_k) = (x_1, \ldots, x_i - 1, \ldots, x_k)$, for every $1 \leq i \leq k$ and for every $k \geq 1$.
- A *finite permutation* $\mathsf{fP}_\ell(x_1, \ldots, x_k) = (x_{i_1}, \ldots, x_{i_k})$, for every $k \geq 2$, where $\ell = i_1, \ldots, i_k$ is a permutation of $1, \ldots, k$.
- For every $k \geq 2$, the pairing functions $\mathsf{addPair}_h^{(i,j)}, \mathsf{subPair}_h^{(i,j)} : \mathbb{Z}^k \to \mathbb{Z}^k$ such that $1 \leq i < j \leq k$, $1 \leq h \leq k$ and $h \neq i, j$. The function $\mathsf{addPair}_h^{(i,j)}$ is the identity on all its arguments but the one in position h which is incremented by $«x_i, x_j»$. The function $\mathsf{subPair}_h^{(i,j)}$ is the identity on all its arguments but for the one in position h which is decremented by $«x_i, x_j»$. For example, $\mathsf{addPair}_1^{(2,3)}(n, x, y, \ldots) = (n + «x, y», x, y, \ldots)$ and $\mathsf{subPair}_1^{(2,3)}(n, x, y, \ldots) = (n - «x, y», x, y, \ldots)$.
- For every $k \geq 2$, the un-pairing functions $\mathsf{addUnPair}_h^{(i,j)}, \mathsf{subUnPair}_h^{(i,j)} : \mathbb{Z}^k \to \mathbb{Z}^k$ such that $1 \leq i < j \leq k$, $1 \leq h \leq k$ and $h \neq i, j$. The function $\mathsf{addUnPair}_h^{(i,j)}$ is the identity on all its arguments but those ones in positions i and j. They are *incremented* by x and y, respectively, if $«x, y»$ is the argument of position h. The function $\mathsf{subUnPair}_h^{(i,j)}$ is the identity on all arguments but those ones in positions i and j. They are *decremented* by x and y, respectively, if $«x, y»$ is the argument of position h. For instance, $\mathsf{addUnPair}_1^{(2,3)}(«x', y'», x, y, \ldots) = («x', y'», x + x', y + y', \ldots)$ and $\mathsf{subUnPair}_1^{(2,3)}(«x', y'», x, y, \ldots) = («x', y'», x - x', y - y', \ldots)$.

In this work the un-pairing functions $\mathsf{addUnPair}_h^{(i,j)}, \mathsf{subUnPair}_h^{(i,j)}$: are crucial to represent the class PRF inside RPRF[6]. The main motivation to supply un-pairing is to pack information into a single argument which becomes a store

[4] We recall that $\mathbb{N}$ and $\mathbb{Z}$ are in bijection, see [6, Example 5.1].

[5] The bijection $«\_,\_»$ can be defined by composing the bijection between $\mathbb{N} \leftrightarrow \mathbb{Z}$ as in the Example 5.1 of [6] and the variant of Cantor pairing as defined in [13] which is a bijection as well.

[6] In fact, by anticipating the forthcoming [24], the un-pairing functions are admissible in RPRF, i.e. we can encode them by means of the built-in functions and schemes of RPRF.

like the following example shows:

$$
\begin{aligned}
&\mathsf{fP}_{2,1,3,4} \circ \mathsf{subUnPair}_2^{(1,4)} \circ \mathsf{addPair}_2^{(1,4)} \circ \mathsf{subUnPair}_1^{(2,3)} \circ \mathsf{addPair}_1^{(2,3)}(0, x_2, x_3, x_4) = \\
&= \mathsf{fP}_{2,1,3,4} \circ \mathsf{subUnPair}_2^{(1,4)} \circ \mathsf{addPair}_2^{(1,4)} \circ \mathsf{subUnPair}_1^{(2,3)}(«x_2, x_3», x_2, x_3, x_4) \\
&= \mathsf{fP}_{2,1,3,4} \circ \mathsf{subUnPair}_2^{(1,4)} \circ \mathsf{addPair}_2^{(1,4)}(«x_2, x_3», 0, 0, x_4) \\
&= \mathsf{fP}_{2,1,3,4} \circ \mathsf{subUnPair}_2^{(1,4)}(«x_2, x_3», ««x_2, x_3», x_4», 0, x_4) \\
&= \mathsf{fP}_{2,1,3,4}(0, ««x_2, x_3», x_4», 0, 0) = (««x_2, x_3», x_4», 0, 0, 0)
\end{aligned}
$$

Definition 2 (Sequential Composition Scheme) Let $j, k \geq 1$ and let $k_1, \ldots, k_j \in \mathbb{N}$ be such that $k = \sum_{i=1}^{j} k_i$. Let $g_i : \mathbb{Z}^{k_i} \longrightarrow \mathbb{Z}^{k_i}$ and $f : \mathbb{Z}^k \longrightarrow \mathbb{Z}^k$ be functional relations, where $1 \leq i \leq j$. The *sequential composition* of f with $g_1, \ldots, g_j$ that is $\circ[f; g_1, \ldots, g_j](\vec{x}_1, \ldots, \vec{x}_j) = f(g_1(\vec{x}_1), \ldots, g_j(\vec{x}_j))$ and yields a functional relation from $\mathbb{Z}^k$ to $\mathbb{Z}^k$. Of course, for every $1 \leq i \leq n$, we assume that $\vec{x}_i$ contains k_i elements.

The composition among elements of RPRF and PRF have no major differences. Given two functions $f, g : \mathbb{Z}^k \to \mathbb{Z}^k$, we abbreviate $\circ[f; g]$ by means of the more standard $f \circ g$. Moreover, let $f^{\stackrel{n}{\cdots}}$ denote the instance of the *sequential composition* that composes n occurrences of f, for any $n \geq 0$.

Definition 3 (Recursion Scheme) Let $k \geq 1$ and let $f, g, h : \mathbb{Z}^k \longrightarrow \mathbb{Z}^k$ be functional relations. The function $\mathsf{Rec}^i[f, g, h] : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ on f, g and h is:

$$
\mathsf{Rec}^i[f, g, h](\vec{x_0}, y, \vec{z_0}) = \begin{cases} (\vec{x_1}, y, \vec{z_1}) & \text{if } y > 0 \text{ and } f^{\stackrel{y}{\cdots}}(\vec{x_0}, \vec{z_0}) = (\vec{x_1}, \vec{z_1}) \\ (\vec{x_1}, 0, \vec{z_1}) & \text{if } y = 0 \text{ and } g(\vec{x_0}, \vec{z_0}) = (\vec{x_1}, \vec{z_1}) \\ (\vec{x_1}, y, \vec{z_1}) & \text{if } y < 0 \text{ and } h^{\stackrel{-y}{\cdots}}(\vec{x_0}, \vec{z_0}) = (\vec{x_1}, \vec{z_1}) \end{cases}
$$

for every $1 \leq i \leq k + 1$. Of course, we assume that $\vec{x_0}$ and $\vec{x_1}$ contain $i - 1$ elements, while $\vec{z_0}$ and $\vec{z_1}$ contain $(k + 1) - i$ elements.

Despite we call it "recursive", $\mathsf{Rec}^i[f, g, h]$ is not defined in terms of itself. We call it that way for two reasons. One is that "recursive" refers to effective computational processes. The other is that RPRF and PRF turns out to be equivalent The above scheme $\mathsf{Rec}^i[f, g, h]$ iteratively applies one of the three parameters $f, g, h \in$ RPRF as many times as the value of the argument in position i if $x_i \neq 0$. Otherwise, if $x_i = 0$, it applies g once. The termination of the scheme is thus immediate. The value of x_i cannot be argument of the iterated function: it reappears untouched as part of the result. Of course the i-th argument can be negative. We take into account this case by using its absolute value for driving the iteration.

Definition 4 (*Reversible Primitive Recursive functions*) The set RPRF of Reversible Primitive Recursive functions is the least class of functions which contains the *Base Primitive Recursive Functions* (Definition 1) and is closed under the *sequential composition* scheme (Definition 2) and the *recursion* scheme (Definition 3.)

Lemma 1 *All functions in* RPRF *are total.*

Proof Every function which belongs to the Base Reversible Primitive Recursive Functions is total. The composition of total functions is total. The recursion scheme is also defined as compositions of total functions. So, the claim follows. □

It is worth introducing some notation to simplify Definition 5 here below which gives an effective inversion maps from RPRF to RPRF.

- For every $k \in \mathbb{N}$, the *identity* $\mathsf{Id}^k(\vec{x}) = \vec{x}$ is the permutation that does not exchange any of its k arguments. When clear, we omit the superscript.
- Let $f_i : \mathbb{Z}^{k_i} \longrightarrow \mathbb{Z}^{k_i}$ and let $\vec{x}_i$ contain k_i elements for every $1 \leq i \leq n$. The *parallel composition* of $f_1, \ldots, f_n$ from $\mathbb{Z}^{k_1+\cdots+k_n}$ to $\mathbb{Z}^{k_1+\cdots+k_n}$ is $(f_1 \parallel \ldots \parallel f_n)(\vec{x}_1, \ldots, \vec{x}_n) = \mathsf{Id}^{k_1+\cdots+k_n}(f_1(\vec{x}_1), \ldots, f_n(\vec{x}_n))$.

Definition 5 The map $\circledR : \mathsf{RPRF} \rightarrow \mathsf{RPRF}$ is defined inductively as follows:

- $\circledR(\mathsf{S}_i) = \mathsf{P}_i$ and $\circledR(\mathsf{P}_i) = \mathsf{S}_i$;
- $\circledR(\mathsf{fP}_\ell)$ is the unique finite permutation $\mathsf{fP}_{\ell'}$ that inverts fP_ℓ, for any permutation fP_ℓ;
- $\circledR(\circ[f; g_1, \ldots, g_j]) = \circ[(\circledR(g_1) \parallel \ldots \parallel \circledR(g_n)); \circledR(f)]$;
- $\circledR(\mathsf{addPair}_h^{(i,j)}) = \mathsf{subPair}_h^{(i,j)}$ and $\circledR(\mathsf{subPair}_h^{(i,j)}) = \mathsf{addPair}_h^{(i,j)}$;
- $\circledR(\mathsf{addUnPair}_h^{(i,j)}) = \mathsf{subUnPair}_h^{(i,j)}$ and $\circledR(\mathsf{subUnPair}_h^{(i,j)}) = \mathsf{addUnPair}_h^{(i,j)}$;
- $\circledR(\mathsf{Rec}^i[f, g, h]) = \mathsf{Rec}^i[\circledR(f), \circledR(g), \circledR(h)]$.

The next theorem show that the $\circledR$ is actually an inverter in the sense of [11], namely a map associating each function to its inverse.

Theorem 3 **(RPRF is closed under inversion)** *If $f : \mathbb{Z}^k \rightarrow \mathbb{Z}^k$ is a* RPRF *then, $f(\vec{x}) = \vec{y}$ if and only if $\circledR(f)(\vec{y}) = \vec{x}$.*

Proof The proof is by induction on the Definition 5. □

Corollary 2 *Each* RPRF *is a bijective function on $\mathbb{Z}^k$, for some $k \in \mathbb{N}$.*

Theorem 3 technically justifies why RPRF works on $\mathbb{Z}$ instead of $\mathbb{N}$. If the issue is to define a theory of computable reversible functions, the restriction to $\mathbb{N}$ and to a class of functions where the predecessor cannot be a primitive function looks artificial. This position, which we share with [21], will be reinforced by the coming sections, where we show that RPRF is complete with respect to PRF which means that we are developing a theory of computable functions which are reversible.

2.1 Expressiveness of RPRF

We show the expressiveness of RPRF and how Bennett's method [4] fits into it. Let $k \in \mathbb{N}$.

- Let $\mathsf{inc} : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ be defined as $\mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{P}_1]$, that is $\mathsf{inc}(n, x, \ldots) = (n+x, x, \ldots)$. The function $\mathsf{inc}^i_j : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ generalizes inc by involving the values of the arguments in position i and j, provided that $i \neq j$ and $1 \leq i, j \leq 2+k$. The first one drives the iteration. The value of the latter gets added to the value of the first as follows:

$$\mathsf{inc}^i_j(\overbrace{\underbrace{\ldots}_{i-1}, n, \ldots}^{j-1}, x, \ldots) = (\overbrace{\underbrace{\ldots}_{i-1}, n, \ldots}^{j-1}, x+n, \ldots) \ .$$

  If $j < i$ then we can define inc^i_j as $\mathsf{Rec}^i[\mathsf{S}_j, \mathsf{Id}, \mathsf{P}_j]$. If $i < j$ then we can define inc^i_j as $\mathsf{Rec}^i[\mathsf{S}_{j-1}, \mathsf{Id}, \mathsf{P}_{j-1}]$ because x_i is hidden by recursion (cf. Definition 4). We remark that if x_i is negative then we subtract it from x.
- The function $\mathsf{dec}^i_j : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ involves the values of the arguments in position i and j, provided that $i \neq j$ and $1 \leq i, j \leq 2+k$. The first one drives the iteration. The value of the latter gets subtracted from the value of the first as follows:

$$\mathsf{dec}^i_j(\overbrace{\underbrace{\ldots}_{i-1}, n, \ldots}^{j-1}, x, \ldots) = (\overbrace{\underbrace{\ldots}_{i-1}, n, \ldots}^{j-1}, x-n, \ldots) \ .$$

  If $j < i$ then we define dec^i_j as $\mathsf{Rec}^i[\mathsf{P}_j, \mathsf{Id}, \mathsf{S}_j]$, otherwise $\mathsf{Rec}^i[\mathsf{P}_{j-1}, \mathsf{Id}, \mathsf{S}_{j-1}]$. Remark that $Ⓡ(\mathsf{dec}^i_j) = \mathsf{inc}^i_j$.
- The function $\mathsf{neg}^i_j : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ involves the values of the arguments in position i and j, provided that $i \neq j$, $1 \leq i$ and $j \leq 2+k$. The function inverts the sign of the argument i, while the j argument serves as an ancilla. If $i < j$ then $\mathsf{fP}_{\ldots,j,\ldots,i,\ldots} \circ \mathsf{inc}^i_j \circ \mathsf{dec}^j_i \circ \mathsf{inc}^i_j$ so that:

$$\mathsf{neg}^j_i(\overbrace{\underbrace{\ldots}_{i-1}, x_i, \ldots}^{j-1}, x_j, \ldots) = (\overbrace{\underbrace{\ldots}_{i-1}, -x_i, \ldots}^{j-1}, x_j, \ldots) \ .$$

  The case $j < i$ is similar. We note that $\mathsf{subPair}^{(i,j)}_h$ and $\mathsf{addPair}^{(i,j)}_h$ are interdefinable; for instance, $\mathsf{addPair}^{(i,j)}_h = \mathsf{neg}^i_h \circ \mathsf{subPair}^{(i,j)}_h \circ \mathsf{neg}^i_h$. Likewise, $\mathsf{subUnPair}^{(i,j)}_h$ and $\mathsf{addUnPair}^{(i,j)}_h$ are interdefinable.
- If $\mathsf{sum} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ is defined as $\mathsf{inc}^3_1 \circ \mathsf{inc}^2_1$ then $\mathsf{sum}(n, x_1, x_2, \ldots) = (n + x_1 + x_2, x_1, x_2, \ldots)$. Moreover, $Ⓡ(\mathsf{sum})(n, x_1, x_2, \ldots) = (n - x_1 - x_2, x_1, x_2, \ldots)$. The natural generalization under the same pattern as inc^i_j and dec^i_j is $\mathsf{sum}^{(i,j)}_h : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ which adds the arguments of position i and j to the one of position h, provided that i, j, h are pairwise distinct and $1 \leq i, j, h \leq 2+k$. For example, $\mathsf{sum}(9, 5, -3) = (11, 5, -3)$. Generally speaking, the sum of two numbers needs an argument initialized to zero. This is the typical side-effect of representing an inherently non-reversible function by a reversible one. To avoid such a side effect, [4] uses a third tape and [36] uses some input-constant.

– By definition, $\mathsf{mult} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ is $\mathsf{Rec}^3[\mathsf{inc}_1^2, \mathsf{Id}, \mathsf{dec}_1^2]$ such that:

$$\mathsf{mult}(n, x_1, x_2, \ldots) = (n + \underbrace{x_1 + \ldots + x_1}_{x_2}, x_1, x_2, \ldots)$$
$$\text{®}(\mathsf{mult})(n, x_1, x_2, \ldots) = (n - (\underbrace{x_1 + \ldots + x_1}_{x_2}), x_1, x_2, \ldots) \ .$$

Its generalization $\mathsf{mult}_h^{(i,j)} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ is:

$$\mathsf{mult}_h^{(i,j)}(\overbrace{\overbrace{\underbrace{\ldots}_{h-1}, n, \ldots}^{i-1}, x_1, \ldots}^{j-1}, x_2, \ldots) = (\overbrace{\overbrace{\underbrace{\ldots}_{h-1}, n + \underbrace{x_1 + \ldots + x_1}_{x_2}, \ldots}^{i-1}, x_1, \ldots}^{j-1}, x_2, \ldots) \ .$$

It adds the product between the i-th and the j arguments to the argument of position h. We define it as $\mathsf{mult}_h^{(i,j)} = \mathsf{Rec}^j[\mathsf{inc}_h^i, \mathsf{Id}, \mathsf{dec}_h^i]$, provided that $1 \leq h < i < j \leq 3 + k$.

– Let $\mathsf{square} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ be defined as $\mathsf{dec}_3^2 \circ \mathsf{mult}_1^{(2,3)} \circ \mathsf{inc}_3^2$, therefore $\mathsf{square}(0, x, 0, \ldots) = (x^2, x, 0, \ldots)$. We emphasize that the square operator rests on the assumption that a zero-valued argument (the third one) is available.

The *following examples* introduce functions deliberately defined to behave like the identity on negative inputs. This simplifies their definition but leave them general enough to represent various interesting functions. We can obtain such a behavioral asymmetry by exploiting the branching mechanism of $\mathsf{Rec}^-[\_, \_, \_]$ that allows to determine the sign of one of its arguments.

– The (total) predecessor restricted to positive numbers $\mathsf{totalNatPred} : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ is defined as $\mathsf{S}_2 \circ \mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{P}_2$. The sub-term $\mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{Id}]$ modifies the first argument only if the result of P_2 is negative. Finally, S_2 restores the second argument. The defined function ensures that if $x \geq 0$, then $\mathsf{totalNatPred}(0, x, \ldots) = ((x \dot{-} 1), x, \ldots)$, otherwise $\mathsf{totalNatPred}(0, x, \ldots) = (0, x, \ldots)$.
– The (total) subtraction restricted to positive numbers $\mathsf{totalNatMinus} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ is $\mathsf{inc}_2^3 \circ \mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{dec}_2^3$ (with dec defined as above), that is $\mathsf{totalNatMinus}(0, x_1, x_2, \ldots) = ((x_1 \dot{-} x_2), x_1, x_2, \ldots)$.
– The factorial is $\mathsf{fact} : \mathbb{Z}^{7+k} \to \mathbb{Z}^{7+k}$ such that $\mathsf{fact}(0, x, 0, 0, 0, z, 0, \ldots) = (x!, x, 0, 0, z', 0 \ldots)$, for every $x \geq 0$. The 6th argument can be thought of as a sort of trash-bin that assures we obtain an injective function. We proceed as follows:

  – Let $\ell_{1,3}$ swap its first and third arguments. Let $\ell_{4,5}$ swap its fourth and fifth arguments. Let $\ell_{2,7}$ swap its second and seventh arguments. They are the identity elsewhere.

– Then clean_3 is $\mathsf{fP}_{\ell_{4,5}} \circ \mathsf{subUnPair}_4^{(3,5)} \circ \mathsf{addPair}_4^{(3,5)}$ such that:

$$\begin{aligned}
&\mathsf{clean}_3(x_1, x_2, x_3, 0, x_5, \ldots) = \\
&= \mathsf{fP}_{\ell_{4,5}} \circ \mathsf{subUnPair}_4^{(3,5)}(x_1, x_2, x_3, «x_3, x_5», x_5, \ldots) \\
&= \mathsf{fP}_{\ell_{4,5}}(x_1, x_2, 0, «x_3, x_5», 0, \ldots) \\
&= (x_1, x_2, 0, 0, «x_3, x_5», \ldots) \ .
\end{aligned}$$

Recall that the recursion hides an argument. So, for the sake of simplicity, we use the seventh argument to drive the iteration.

– So, we can conclude that fact is $\mathsf{Rec}^7[\mathsf{P}_2 \circ \mathsf{clean}_3 \circ \mathsf{mult}_1^{(2,3)} \circ \mathsf{fP}_{\ell_{1,3}}, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{inc}_7^2 \circ \mathsf{S}_1$.

3 Primitive Recursive functions and RPRF

We recall a possible definition of the class of Primitive Recursive Functions (PRF) [6, 23].

Definition 6 (Basic Primitive Recursive Functions) The function constantly equal to 0, i.e. such that $\mathsf{0}(\vec{x}) = 0$, the successor such that $\mathsf{S}(x) = x+1$ and the projections such that $\pi_i^k(x_1, \ldots, x_k) = x_i$ for all $k \geq i \geq 1$ are the *basic Primitive Recursive Functions.*

Definition 7 (Composition scheme) Let $g_1, \ldots, g_m : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^m \to \mathbb{N}$ be total functions. The *composition scheme* is $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$ from $\mathbb{N}^n$ to $\mathbb{N}$.

Definition 8 (Recursion scheme) Let $g : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^{n+2} \to \mathbb{N}$ be total functions. The *recursion scheme* is:

$$\begin{aligned}
f(x_1, \ldots, x_n, 0) &= g(x_1, \ldots, x_n) \\
f(x_1, \ldots, x_n, y+1) &= h(f(x_1, \ldots, x_n, y), x_1, \ldots, x_n, y)
\end{aligned}$$

from $\mathbb{N}^{n+1}$ to $\mathbb{N}$.

The above scheme is often called *primitive recursion schema*, since it catches a limited form of recursion [29].

Definition 9 (Primitive Recursive Functions) The Primitive Recursive Functions is the least class of functions which contains the basic Primitive Recursive Functions of Definition 6 and which is closed under the composition scheme of Definition 7 and the recursion scheme of Definition 8.

We are interested to prove that PRF and RPRF are reciprocally interdefinable. We refer the reader to [25] for the detailed definition of a map from every function of RPRF to a function of PRF that simulates it.

3.1 From PRF to RPRF

For any $f \in \mathsf{PRF}$, we show how to define a corresponding function in RPRF which, suitably restricted in domain and range, extensionally behaves as f, of course, exploiting that $\mathbb{N} \subseteq \mathbb{Z}$. Using the terminology that [3,38] advocate we obtain the *injectivization* of $f \in \mathsf{PRF}$ into RPRF by forwarding its input as part of the output.

Definition 10 (RPRF-*definable functions*) A function $f : \mathbb{N}^k \to \mathbb{N}$ is RPRF^k_h-definable (for $h \geq 3$) whenever a function $in_{\mathsf{RPRF}}(f) : \mathbb{Z}^{k+h} \to \mathbb{Z}^{k+h}$ in RPRF exists such that, for all $x_1, \ldots, x_k, z \in \mathbb{N}$, if $f(x_1, \ldots, x_k) = y$ then:

$$in_{\mathsf{RPRF}}(f)(0, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z) = (y, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z') \ ,$$

for some $z' \in \mathbb{N}$.

We write $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^k_h$ to denote that $in_{\mathsf{RPRF}}(f)$ represents f which is RPRF^k_h-definable. Some remarks on Definition 10 are in order. Extensionally, every $in_{\mathsf{RPRF}}(f)$ behaves as an identity on all its arguments, but on the first and the last ones. This means that every argument with position $2 \leq i \leq k+h-1$ reappears as part of the output even though its value can be altered in the course of the computation. The last argument, with position $k+h$, plays the role of a waste bin that we shall operate on, as it was a stack. The first argument, which conventionally carries the value 0, balances the presence of the output $f(x_1, \ldots, x_k)$ of the function we encode. The first argument makes the input and the output arities equal. We observe that whenever the arguments $x_1, \ldots, x_k$ of Definition 10 are non-negative the value of y in $in_{\mathsf{RPRF}}(f)$ is non-negative.

Lemma 2 (Weakening) *Let $h \geq 3$. For every $f : \mathbb{N}^k \to \mathbb{N}$, if f is RPRF^k_h-definable, then f is also RPRF^k_{h+1}-definable.*

Lemma 2 holds because, if $f : \mathbb{N}^k \to \mathbb{N}$ is defined by $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^k_h$ for some $h \geq 3$ then $Ⓡ(\mathsf{fP}_\ell) \circ (in_{\mathsf{RPRF}}(f) \parallel \mathsf{Id}) \circ \mathsf{fP}_\ell$, where $\ell = 1, \ldots, k+h+1, k+h$, represents f in RPRF^k_{h+1}. We remark that $Ⓡ(\mathsf{fP}_\ell) = \mathsf{fP}_\ell$.

The two following functions of RPRF show why we consider the last argument, and the last output of a given $in_{\mathsf{RPRF}}(\ )$ a sort of waste bin which we use as a stack.

Definition 11 (Push and pop) For any $n \in \mathbb{N}$ such that $n \geq 3$, let $\ell = 1, \ldots, n, n-1$. We denote push_i the term $\mathsf{fP}_\ell \circ \mathsf{subUnPair}^{(i,n)}_{n-1} \circ \mathsf{addPair}^{(i,n)}_{n-1}$ that defines the following map:

$$\mathsf{push}_i(\ldots, x_{i-1}, x_i, x_{i+1}, \ldots, 0, x_n) = (\ldots, x_{i-1}, 0, x_{i+1}, \ldots, 0, «x_i, x_n») \ ,$$

i.e. push_i is the identity everywhere but on both its i-th and last arguments. Symmetrically, we denote pop_i the term $\mathsf{subPair}^{(i,n)}_{n-1} \circ \mathsf{addUnPair}^{(i,n)}_{n-1} \circ \mathsf{fP}_\ell$ that defines the following map:

$$\mathsf{pop}_i(\ldots, x_{i-1}, 0, x_{i+1}, \ldots, 0, «x_i, x_n») = (\ldots, x_{i-1}, x_i, x_{i+1}, \ldots, 0, x_n) \ ,$$

i.e. pop_i is the identity everywhere but on both its i-th and last arguments.

The proof of the next theorem proposes a reversibilization (in accordance with the terminology in [3, 38]) of primitive recursive functions.

Theorem 4 *Every $f \in \mathsf{PRF}$ is RPRF-definable.*

The proof of Theorem 4 is by induction on the definition of $f \in \mathsf{PRF}$. For the sake of simplicity, we present some of its cases in an exemplified form.

- Let f be $\mathsf{0} : \mathbb{N}^k \to \mathbb{N}$ for some fixed k. We define $in_{\mathsf{RPRF}}(\mathsf{0}) = \mathsf{Id}^{k+3}$, whose arity is $k+3$, such that $in_{\mathsf{RPRF}}(\mathsf{0})(0, x_1, \ldots, x_k, 0, y) = (0, x_1, \ldots, x_k, 0, y)$. This means that $\mathsf{0}$ is RPRF^k_3-definable.
- Let f be $\pi^k_i : \mathbb{N}^k \to \mathbb{N}$ for some fixed k. We define $in_{\mathsf{RPRF}}(\pi^k_i) = \mathsf{inc}^{i+1}_1$ such that $in_{\mathsf{RPRF}}(\pi^k_i)(0, x_1, \ldots, x_k, 0, y) = (0 + x_i, x_1, \ldots, x_k, 0, y)$, where inc^{i+1}_1 is the one defined in Section 2.1. So projections are RPRF^k_3-definable.
- Let f be $\mathsf{S} : \mathbb{N} \to \mathbb{N}$. We define $in_{\mathsf{RPRF}}(\mathsf{S}) = \mathsf{S}_1 \circ \mathsf{inc}^2_1$ with arity $1+3$ such that $in_{\mathsf{RPRF}}(\mathsf{S})(0, x_1, 0, y) = \mathsf{S}_1(0 + x_1, x_1, 0, y) = (x_1 + 1, x_1, 0, y)$. So, the PRF-successor is RPRF^1_3-definable.
- Let $f : \mathbb{N}^k \to \mathbb{N}$ be a PRF defined as $f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_m(\vec{x}))$ where $h : \mathbb{N}^m \to \mathbb{N}$, $g_i : \mathbb{N}^k \to \mathbb{N}$ are PRF. By induction hypothesis: (i) h is $\mathsf{RPRF}^m_{l_h}$-definable, for some l_h; and (ii) g_i is $\mathsf{RPRF}^k_{l_i}$-definable for some l_i, with $1 \le i \le m$. Let $l = \max\{l_1, l_2, l_3, l_h\}$ and $l' = 3 + m \cdot (k + l)$ so that both $in_{\mathsf{RPRF}}(g_1), in_{\mathsf{RPRF}}(g_2), in_{\mathsf{RPRF}}(g_3) \in \mathsf{RPRF}^k_l$ and $in_{\mathsf{RPRF}}(h) \in \mathsf{RPRF}^m_{l'-m}$ exist.
  Our goal is to define $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^k_{l'}$ that represents f.
  For the sake of simplicity, we focus on the details of the case with $m = 3$, $k = 2$ and $l = 3$. It shows all the technical problems.
  1. We are looking for $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^2_{18}$. By Definition 10 we expect an input $0, x_1, x_2, \underbrace{0, \ldots, 0}_{15}, 0, z$ with 20 arguments.
  2. We want to arrange the arguments for $\mathsf{Id}^3 \parallel in_{\mathsf{RPRF}}(g_1) \parallel in_{\mathsf{RPRF}}(g_2) \parallel in_{\mathsf{RPRF}}(g_3) \parallel \mathsf{Id}^2$ in $\mathbb{Z}^{20} \to \mathbb{Z}^{20}$ because $in_{\mathsf{RPRF}}(g_1), in_{\mathsf{RPRF}}(g_2), in_{\mathsf{RPRF}}(g_3) \in \mathsf{RPRF}^2_3$. So, we apply the next RPRF-functions. This means that $\mathsf{inc}^2_{15} \circ \mathsf{inc}^2_{10} \circ \mathsf{inc}^2_5$ produces:
  $$0, x_1, x_2, \underbrace{0, x_1, 0, 0, 0}_{5}, \underbrace{0, x_1, 0, 0, 0}_{5}, \underbrace{0, x_1, 0, 0, 0}_{5}, 0, z$$
  while $\mathsf{inc}^3_{16} \circ \mathsf{inc}^3_{11} \circ \mathsf{inc}^3_6$ produces:
  $$0, x_1, x_2, \underbrace{0, x_1, x_2, 0, 0}_{5}, \underbrace{0, x_1, x_2, 0, 0}_{5}, \underbrace{0, x_1, x_2, 0, 0}_{5}, 0, z \quad .$$
  3. The application of $\mathsf{Id}^3 \parallel in_{\mathsf{RPRF}}(g_1) \parallel in_{\mathsf{RPRF}}(g_2) \parallel in_{\mathsf{RPRF}}(g_3) \parallel \mathsf{Id}^2$ yields the following tuple with twenty elements:
  $$\begin{aligned} &0, x_1, x_2, \underbrace{g_1(x_1, x_2), x_1, x_2, 0, z_1}_{5}, \\ &\quad \underbrace{g_2(x_1, x_2), x_1, x_2, 0, z_2}_{5}, \underbrace{g_3(x_1, x_2), x_1, x_2, 0, z_3}_{5}, 0, z \quad . \end{aligned}$$

4. Now, we arrange the arguments which we apply $\mathsf{Id}^2 \parallel in_{\mathsf{RPRF}}(h)$ where $in_{\mathsf{RPRF}}(h) \in \mathsf{RPRF}^3_{15}$ to. We push the useless values on the "stack", we erase copies of x_1, x_2 and we suitably permute arguments. For doing this, let z^* denote $«z_1, «z_2, «z_3, z»»»$. Then, the composition $\mathsf{push}_{18} \circ \mathsf{push}_{13} \circ \mathsf{push}_8$ produces:
$$0, x_1, x_2, \underbrace{g_1(x_1,x_2), x_1, x_2, 0, 0}_{5},$$
$$\underbrace{g_2(x_1,x_2), x_1, x_2, 0, 0}_{5}, \underbrace{g_3(x_1,x_2), x_1, x_2, 0, 0}_{5}, 0, z^*$$
and $\mathsf{dec}^3_{16} \circ \mathsf{dec}^2_{15} \circ \mathsf{dec}^3_{11} \circ \mathsf{dec}^2_{10} \circ \mathsf{dec}^3_{6} \circ \mathsf{dec}^2_{5}$ produces:
$$0, x_1, x_2, \underbrace{g_1(x_1,x_2), 0, 0, 0, 0}_{5},$$
$$\underbrace{g_2(x_1,x_2), 0, 0, 0, 0}_{5}, \underbrace{g_3(x_1,x_2), 0, 0, 0, 0}_{5}, 0, z^*$$
and a suitable finite permutation fP_{ℓ_2} produces:
$$x_1, x_2, 0, g_1(x_1,x_2), g_2(x_1,x_2), g_3(x_1,x_2),$$
$$\underbrace{0, \dots, 0}_{12}, 0, «z_1, «z_2, «z_3, z»»» \enspace .$$
5. Applying $\mathsf{Id}^2 \parallel in_{\mathsf{RPRF}}(h)$ we get:
$$x_1, x_2, f(x_1,x_2), g_1(x_1,x_2), g_2(x_1,x_2), g_3(x_1,x_2), \underbrace{0, \dots, 0}_{12}, 0, z_4$$
because $f(x_1, x_2) = h(g_1(x_1,x_2), g_2(x_1,x_2), g_3(x_1,x_2))$.
6. Applying $\mathsf{push}_4 \circ \mathsf{push}_5 \circ \mathsf{push}_6$ pushes $g_1(x_1,x_2), g_2(x_1,x_2), g_3(x_1,x_2)$ on the "stack".
7. The last step is permuting $f(x_1, x_2)$ with the first two arguments, getting to $f(x_1,x_2), x_1, x_2, \underbrace{0, \dots, 0}_{15}, 0, z_5$ which satisfies Definition 10. This means that f is RPRF^2_{18}-definable.

– Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ be the PRF defined as $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$ and $f(\overrightarrow{x}, y+1) = h(f(\overrightarrow{x}, y), \overrightarrow{x}, y)$, where $h : \mathbb{N}^{k+2} \to \mathbb{N}$ and $g : \mathbb{N}^k \to \mathbb{N}$.
By the inductive hypothesis, h is $\mathsf{RPRF}^{k+2}_{l_h}$-definable and g is $\mathsf{RPRF}^{k}_{l_g}$-definable. Let $l = \max\{l_g, 3+l_h\}$ so that $in_{\mathsf{RPRF}}(g) \in \mathsf{RPRF}^k_l$ and $in_{\mathsf{RPRF}}(h) \in \mathsf{RPRF}^{k+2}_{l-3}$ exist.
We aim at building a $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^{k+1}_l$.
For the sake of simplicity, we discuss the details of the case with $k = 2$, $l_g = 3$ and $l_h = 5$ which shows all the technical problems. We remind that the evaluation of the PRF function $f(\overrightarrow{x}, n)$ starts by evaluating $g(\overrightarrow{x})$ and proceeds by iteratively applying h as many times as n.

1. We are looking for $in_{\mathsf{RPRF}}(f) \in \mathsf{RPRF}^3_8$ thus, by Definition 10, we would expect an input of the shape $0, x_1, x_2, y, 0, 0, 0, 0, 0, 0, z$ containing 11 arguments.
2. We want to arrange the arguments for $\mathsf{Id}^1 \parallel in_{\mathsf{RPRF}}(g)$ that, belongs to $\mathbb{Z}^{11} \to \mathbb{Z}^{11}$ because $in_{\mathsf{RPRF}}(g) \in \mathsf{RPRF}^2_8$. Thus, we apply a suitable finite permutation prefixing y to the remaining argument-list.
3. The application of $\mathsf{Id}^1 \parallel in_{\mathsf{RPRF}}(g)$ produces $y, g(x_1, x_2), x_1, x_2, 0, 0, 0, 0, 0, 0, z$.
4. The more tricky point is the simulation of the primitive-recursion by means of the reversible-recursion.
   We move an argument from position 5 to position 2 (by means of a finite permutation), obtaining $y, 0, g(x_1, x_2), x_1, x_2, 0, 0, 0, 0, 0, z$. Since we want to use y to drive the recursion, we need to define an auxiliary function $h^* : \mathbb{Z}^{10} \to \mathbb{Z}^{10}$ making $\mathsf{Rec}^1[h^*, \mathsf{Id}^{10}, \mathsf{Id}^{10}]$ our recursive block. We remark that y (i.e. the first argument) is excluded by the argument-list provided to h^* by Definition 4. Thus, the argument-list supplied to h^* is $0, g(x_1, x_2), x_1, x_2, 0, 0, 0, 0, 0, z$ containing 10 values.
   The main issue for getting to the definition of $in_{\mathsf{RPRF}}(f)$ is that each application of h^* requires an argument-list which carries the information about how many times h^* has already been applied. The value zero of position 5, which we increment at each step, provides such an information to h^*. Additionally, at each recursive step, we push the previous result (in position 2) and, finally, we permute the first two positions of the argument-list (i.e. we put a zero in the position 1 and we make the new intermediary result available) by using a suitable finite permutation fP_{ℓ_3}. Formally, we define h^* as $\mathsf{fP}_{\ell_3} \circ \mathsf{push}_2 \circ \mathsf{S}_5 \circ in_{\mathsf{RPRF}}(h)$, so that

$$
\begin{aligned}
&h^*(0, f(x_1, x_2, n), x_1, x_2, n, 0, 0, 0, 0, z) = \\
&= \mathsf{fP}_{\ell_3} \circ \mathsf{push}_2 \circ \mathsf{S}_i(f(x_1, x_2, n+1), f(x_1, x_2, n), x_1, x_2, n, 0, 0, 0, 0, z) \\
&= \mathsf{fP}_{\ell_3} \circ \mathsf{push}_2(f(x_1, x_2, n+1), f(x_1, x_2, n), x_1, x_2, n+1, 0, 0, 0, 0, z) \\
&= \mathsf{fP}_{\ell_3}(f(x_1, x_2, n+1), 0, x_1, x_2, n+1, 0, 0, 0, 0, \langle\!\langle f(x_1, x_2, n), z\rangle\!\rangle) \\
&= (0, f(x_1, x_2, n+1), x_1, x_2, n+1, 0, 0, 0, 0, \langle\!\langle f(x_1, x_2, n), z\rangle\!\rangle)
\end{aligned}
$$

   which is ready for the next recursive step. Notice that h^* does not adhere to Definition 10 because of its fifth argument, so h^* does not define a PRF function. However, it is sufficient that the $in_{\mathsf{RPRF}}(f)$ (we want define) adhere to Definition 10.
5. The application of $\mathsf{Rec}^1[h^*, \mathsf{Id}^{10}, \mathsf{Id}^{10}]$ to $y, 0, g(x_1, x_2), x_1, x_2, 0, 0, 0, 0, 0, z$, produces $y, 0, f(x_1, x_2, y), x_1, x_2, y, 0, 0, 0, 0, z'$ for some z'.
6. We can conclude by eliding a copy of y by applying dec^6_1 and then by applying a suitable finite permutation which moves the first two arguments just before the last one. Hence, f is RPRF^2_{18}-definable. $\square$

4 Unbounded minimization

In the thirties Kleene extends the class of primitive recursive functions with a *minimization operator* so introducing the class μRPF of μ-Recursive Partial Functions [17,23,33] which corresponds to the class of functions that Turing machines can compute. The definition of μRPF that we adhere to follows:

Definition 12 (μ-scheme) Let $g : \mathbb{N}^{k+1} \to \mathbb{N}$ be any partial function. The μ-scheme on g is $\mu z[g(z, x_1, \ldots, x_k) = 0]$. It stands for the least natural number n such that:

- $g(n, x_1, \ldots, x_k) = 0$;
- for all $m \in \{0, \ldots, n-1\}$, $g(m, x_1, \ldots, x_k)$ is defined and not equal to 0.

If no such an n exists, then $\mu z[g(z, x_1, \ldots, x_k) = 0]$ is undefined.

Definition 13 (μ-Recursive Partial Functions) The class μRPF of μ-Recursive Partial Functions is the least class of *partial* functions which contains the basic Primitive Recursive Functions of Definition 6 and which is closed under the composition scheme (Definition 7) the recursion scheme (Definition 8) and the μ-scheme (Definition 12).

We remark that Definition 13 builds the elements in the class μRPF by using any of the schemes we have recalled in Definition 7, 8 and 12 following [6,23,33]. Such a freedom to combine composition, recursion and μ-schemes is not necessary because the Normal Form Theorem on μRPF by Kleene says that we can reformulate every function in μRPF as a composition of a primitive recursive function, of the μ-scheme and of another primitive recursive function.

It is worth to remark that in [20,30] we find that the closure of any class of single-valued recursive bijective, i.e. total, functions by means of the μ-scheme yields a class of recursive bijective (total) functions closed under inversion. From that result, in principle, it might not be immediate to adapt the μ-scheme in order to use it on the class RPRF to generate all the partial injective functions. Luckily, RPRF is not a standard class of recursive functions. The functions of RPRF have $\mathbb{Z}$ and not $\mathbb{N}$ as their domain and co-domain; moreover, every function of RPRF is not single-valued. These two “anomalies” allow to get partial functions.

Definition 14 (μ-Reversible scheme) Let $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ be any partial function where $k \in \mathbb{N}$. The following schemes ${}_{add}\mu^{+}(g)$, ${}_{sub}\mu^{+}(g)$, ${}_{add}\mu^{-}(g)$ and ${}_{sub}\mu^{-}(g)$ denote functions from $\mathbb{Z}^{k+1}$ to $\mathbb{Z}^{k+1}$ which are the identity on all arguments but the first one:

- ${}_{add}\mu^{+}(g)(z, x_1, \ldots, x_k) = (z + n, x_1, \ldots, x_k)$ and ${}_{sub}\mu^{+}(g)(z, x_1, \ldots, x_k) = (z - n, x_1, \ldots, x_k)$ whenever:
  - $g(n, x_1, \ldots, x_k) = (0, y^n_1, \ldots, y^n_k)$;
  - $g(m, x_1, \ldots, x_k) = (y^m_0, y^m_1, \ldots, y^m_k)$ is defined and $y^m_0 \neq 0$, for all $m \in \{0, \ldots, n-1\}$;

- $\mathit{add}\mu^{-}(g)(z, x_1, \ldots, x_k) = (z + n, x_1, \ldots, x_k)$ and $\mathit{sub}\mu^{-}(g)(z, x_1, \ldots, x_k) = (z - n, x_1, \ldots, x_k)$ whenever:
  - $g(-n, x_1, \ldots, x_k) = (0, y_1^n, \ldots, y_k^n)$;
  - $g(-m, x_1, \ldots, x_k) = (y_0^m, y_1^m, \ldots, y_k^m)$ is defined and $y_0^m \neq 0$, for all $m \in \{0, \ldots, n-1\}$.

For every $\mathit{add}\mu^{+}(g)$, $\mathit{sub}\mu^{+}(g)$, $\mathit{add}\mu^{-}(g)$ and $\mathit{sub}\mu^{-}(g)$ if the value n cannot exist for the given $x_1, \ldots, x_k$, then they are undefined.

Both g and the corresponding μ-reversible scheme of Definition 14 have the same arity. The application of g hides the first argument which gets instantiated by integer values, until, eventually, the one we are looking for occurs.

We define μRRF by extending RPRF as expected.

Definition 15 (μ-Reversible Recursive Partial Functions) The class μRRF of μ-Reversible Recursive Partial Functions is the least class which contains the base Reversible Primitive Recursive functions of Definition 1 and which is closed under the sequential composition scheme (Definition 2), the iteration scheme (Definition 3) and the μ-Reversible scheme (Definition 14).

We overload Ⓡ from PRF to PRF of Definition 5 to a function from μRRF to μRRF.

Definition 16 Let $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ be a μRRF-function. The function Ⓡ : μRRF $\to$ μRRF is the extension of the namesake function in Definition 5 by the following clauses:

- Ⓡ$(\mathit{add}\mu^{+}(g)) = \mathit{sub}\mu^{+}(g)$ and Ⓡ$(\mathit{sub}\mu^{+}(g)) = \mathit{add}\mu^{+}(g)$;
- Ⓡ$(\mathit{add}\mu^{-}(g)) = \mathit{sub}\mu^{-}(g)$ and Ⓡ$(\mathit{sub}\mu^{-}(g)) = \mathit{add}\mu^{-}(g)$.

Theorem 5 (μRRF is closed under inversion) *If $f : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ is a μRRF then, $f(\vec{x}) = \vec{y}$ if and only if Ⓡ$(f)(\vec{y}) = \vec{x}$.*

Proof The proof is by induction on Ⓡ : μRRF $\to$ μRRF. □

Corollary 3 *Each μRRF is an injective partial function on $\mathbb{Z}^{k+1}$ (a.k.a. injective endorelation), for some $k \in \mathbb{N}$.*

From [25] we know that the class of RPRF-functions is computable because an embedding from RPRF to PRF exists. Of course μRRF cannot embed into PRF because of the unbounded search that the μ-reversible scheme requires. Therefore, μRRF strictly extends RPRF.

Theorem 6 μRRF *contains only effective (reversible) functions, viz. all its functions can be simulated on reversible Turing machines.*

4.1 Turing-Completeness

We here focus on how to encode μRPF into μRRF functions. Aiming at the Turing completeness of μRRF we need to extend the notion of definability of Definition 10 to consider partial functions.

Definition 17 (μRRF*-definable functions*) A partial function $f : \mathbb{N}^k \to \mathbb{N}$ in μRPF is $\mu\mathsf{RRF}^k_h$-definable (for $h \geq 3$) if and only if $in_{\mu\mathsf{RRF}}(f) : \mathbb{Z}^{k+h} \to \mathbb{Z}^{k+h}$ exists in μRRF such that, for all $0, x_1, \ldots, x_k, z \in \mathbb{N}$:

- $z' \in \mathbb{N}$ exists such that:

$$in_{\mu\mathsf{RRF}}(f)(0, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z) = (y, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z')$$

  whenever $f(x_1, \ldots, x_k) = y$ and
- $in_{\mu\mathsf{RRF}}(f)(0, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z)$ is undefined whenever $f(x_1, \ldots, x_k)$ is undefined.

We write "μRRF-definable" to abbreviate "$\mu\mathsf{RRF}^k_h$-definable", for some given $k, h \in \mathbb{N}$, when possible.

Clearly Lemma 2 can be extended from RPRF-definable functions to μRRF-definable functions, providing a reversibilization of the whole language.

Theorem 7 *Every $f \in \mu$RPF is μRRF-definable.*

Proof We show that every f defined as in Definition 13 is μRRF-definable. The proof is by structural induction on the definition of f.

- The first part of the proof closely mimics the proof of Theorem 4. By following Definition 17, we check that if $f(x_1, \ldots, x_k)$ is undefined, then $in_{\mu\mathsf{RRF}}(f)(x_0, x_1, \ldots, x_k, 0, \ldots, 0, z)$ is undefined as well. In most of the cases f and $in_{\mu\mathsf{RRF}}(f)$ are both total. So the above implication is true. In the remaining cases, it is sufficient to remark that the composition is undefined whenever at least one of its arguments is. The same holds for the recursion scheme because it unfolds to a sequence of compositions.
- The proof is complete once shown that every function in μRPF, whose definition relies on some occurrences of the μ-scheme of Definition 12, has its counterpart in μRRF. Let $g : \mathbb{N}^{k+1} \to \mathbb{N}$ be in μRPF and let $\mu n[g(n, x_1, \ldots, x_k) = 0]$ be the function that we must show to be in μRRF. By induction, $in_{\mu\mathsf{RRF}}(g) \in \mu\mathsf{RRF}^{k+1}_h$ for some $h \geq 3$.
  By Definition 17, we can apply $in_{\mu\mathsf{RRF}}(g) : \mathbb{Z}^{k+1+h} \to \mathbb{Z}^{k+1+h}$ to a list of arguments having shape $(0, \underbrace{m, x_1, \ldots, x_k}_{k+1}, \underbrace{0, \ldots, 0}_{h-2}, z)$ because, we search for a $n \in \mathbb{N}$ such that:
  - $in_{\mu\mathsf{RRF}}(g)(0, n, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z) = (0, n, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z')$, and

- $in_{\mu\mathsf{RRF}}(g)(0, m, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z) = (z_m, m, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z')$ is defined with $z_m \neq 0$, for every $m \in \{0, \ldots, n-1\}$.

Let fP_ℓ be the following permutation:
$\mathsf{fP}_\ell(n, x_1, \ldots, x_k, x'_1, \ldots, x'_{h-2}, z) = (x'_{h-2}, n, x_1, \ldots, x_k, x'_1, \ldots, x'_{h-3}, z)$.
We can then search for $n \in \mathbb{N}$ such that:

- $(in_{\mu\mathsf{RRF}}(g) \circ \mathsf{fP}_\ell)(n, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-1}, z) = (0, n, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z')$, and
- $(in_{\mu\mathsf{RRF}}(g) \circ \mathsf{fP}_\ell)(m, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-1}, z) = (z_m, m, x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-2}, z')$ with $z_m \neq 0$, for every $m \in \{0, \ldots, n-1\}$.

Finally, we let $in_{\mu\mathsf{RRF}}(\mu n[g(n, x_1, \ldots, x_k) = 0])$ be $add\mu^+(in_{\mu\mathsf{RRF}}(g) \circ \mathsf{fP}_\ell)$ which is $\mu\mathsf{RRF}^{k+1}_{h+1}$-definable because, if $\mu n[g(n, x_1, \ldots, x_k) = 0]$ is defined, then:

$$\begin{aligned} &in_{\mu\mathsf{RRF}}(\mu n[g(n, x_1, \ldots, x_{k+1}) = 0])(0, x_1 \ldots, x_k, \underbrace{0, \ldots, 0}_{h-1}, z) \\ &\qquad = (\mu n[g(n, x_1, \ldots, x_{k+1}) = 0], x_1, \ldots, x_k, \underbrace{0, \ldots, 0}_{h-1}, z') \ . \qquad \square \end{aligned}$$

5 Variants of $\mu\mathsf{RRF}$

In this section, we first discuss how to relax the structural constraint which forces the equality between the input and the output arity of every $f \in \mu\mathsf{RRF}$. We shall exploit the well-known existence of bijections from $\mathbb{Z}$ to $\mathbb{Z}^2$.

In the second part, we show that we do not harm the reversibility of any function $f \in \mu\mathsf{RRF}$ if we hide both its i-th argument and the corresponding i-th output, provided that the value of the i-th argument remains constant going from the input to the output of f, i.e. if the computation of the results that f supplies is *clean* [38].

5.1 Pairing maps in $\mu\mathsf{RRF}$

We here extend $\mu\mathsf{RRF}$ to $\mu\mathsf{RRF}'$ whose reversible functions can have different input and output arities.

Definition 18 For every $1 \leq i \leq k$, let $\mathsf{join}_i : \mathbb{Z}^{k+1} \to \mathbb{Z}^k$ and $\mathsf{split}_i : \mathbb{Z}^k \to \mathbb{Z}^{k+1}$ be total recursive functions such that:

$$\begin{aligned}
&\mathsf{join}_i(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \ldots, x_{k+1}) = \\
&\qquad (x_1, \ldots, x_{i-1}, «x_i, x_{i+1}», x_{i+2}, \ldots, x_{k+1}) \\
&\mathsf{split}_i(x_1, \ldots, x_{i-1}, «x''_i, x'_i», x_{i+1}, \ldots, x_{k+1}) = \\
&\qquad (x_1, \ldots, x_{i-1}, x'_i, x''_i, x_{i+1}, \ldots, x_{k+1}) \ .
\end{aligned}$$

Once fixed i, join_i and split_i are mutually inverse. i.e. their composition is the identity.

Definition 19 The class $\mu\mathsf{RRF}'$ is the least class which contains the base Reversible Primitive Recursive functions of Definition 1, the functions $\mathsf{join}_i : \mathbb{Z}^{k+1} \to \mathbb{Z}^k$ and $\mathsf{split}_i : \mathbb{Z}^k \to \mathbb{Z}^{k+1}$, for every $1 \leq i \leq k$, and which is closed under the sequential composition scheme (Definition 2), the iteration scheme (Definition 3) and the μ-Reversible scheme (Definition 14).

Let $«\_, \_» : \mathbb{Z}^2 \to \mathbb{Z}$ be a bijection such that:

$$«0, 0» = 0 \ . \tag{1}$$

The constraint (1) holds true for common pairing maps [35]. If it is true, we can adapt Definition 17 to $\mu\mathsf{RRF}'$ by removing the $h-2$ occurrences of the value 0 used as ancillae because we can generate as many copies of 0 as we need, starting from a single 0 and applying split_i. Symmetrically, we can reduce an arbitrary number of occurrences of 0 to a single 0 by means of join_i.

Of course, the function $\circledR : \mu\mathsf{RRF} \to \mu\mathsf{RRF}$ extends to a namesake $\circledR : \mu\mathsf{RRF}' \to \mu\mathsf{RRF}'$.

Definition 20 The inductive clauses that define $\circledR : \mu\mathsf{RRF}' \to \mu\mathsf{RRF}'$ are the ones that define $\circledR : \mu\mathsf{RRF} \to \mu\mathsf{RRF}$ of Definition 16 plus the clauses:

$$\begin{aligned}
\circledR(\mathsf{join}_i) &= \mathsf{split}_i \ , \\
\circledR(\mathsf{split}_i) &= \mathsf{join}_i \ .
\end{aligned}$$

Theorem 8 *For every $f \in \mu\mathsf{RRF}'$, we have that $\circledR(f) \in \mu\mathsf{RRF}'$.*

5.2 Hiding and Uncovering

We show how extend our reversible functions by allowing the hiding of constant arguments of any given $f \in \mu\mathsf{RRF}$.

Definition 21 Let $f : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ be in $\mu\mathsf{RRF}$. For every $1 \leq i \leq k+1$, let $\mathsf{hide}_i(f) : \mathbb{Z}^k \to \mathbb{Z}^k$ be such that $\mathsf{hide}_i(f)(x_1, \ldots, x_k) = (y_1, \ldots, y_k)$ whenever:

$$\begin{aligned}
&f(x_1, \ldots, x_i, 0, x_{i+1}, \ldots, x_{k+1}) \text{ is defined, and} \\
&f(x_1, \ldots, x_i, 0, x_{i+1}, \ldots, x_{k+1}) = (y_1, \ldots, y_i, 0, y_{i+1}, \ldots, y_{k+1}) \ .
\end{aligned}$$

No loss of information follows from the application of $\mathsf{hide}_i(f)$ because the information it hides is the single value 0. In particular, the implementation of $\mathsf{hide}_i(f)$ in a reversible Turing machine is trivial and commonly used. It corresponds to forget that some portion of the tape is first used and then emptied [2,3]. The natural counterpart of hide_i is in the next definition.

Definition 22 Let $g : \mathbb{Z}^k \to \mathbb{Z}^k$ be in $\mu\mathsf{RRF}$. For every $1 \leq i \leq k$, let $\mathsf{uncover}_i(g) : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ be such that:

$$\mathsf{uncover}_i(g)(x_1, \ldots, x_i, 0, x_{i+1}, \ldots, x_{k+1}) = (y_1, \ldots, y_i, 0, y_{i+1}, \ldots, y_{k+1})$$

whenever $g(x_1, \ldots, x_k) = (y_1, \ldots, y_k)$.

We remark that $\mathsf{uncover}_i$ somewhat internalizes the Weakening Lemma 2.

Definition 23 The class $\mu\mathsf{RRF}''$ is the least class which contains the base Reversible Primitive Recursive functions of Definition 1, the function $\mathsf{hide}_i : \mathbb{Z}^k \to \mathbb{Z}^k$, for every $1 \leq i \leq k+1$, the function $\mathsf{uncover}_j : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$, for every $1 \leq j \leq k$, and which is closed under the sequential composition scheme (Definition 2), the iteration scheme (Definition 3) and the μ-Reversible scheme (Definition 14).

The function $\circledR : \mu\mathsf{RRF} \to \mu\mathsf{RRF}$ extends to a namesake $\circledR : \mu\mathsf{RRF}'' \to \mu\mathsf{RRF}''$ as follows.

Definition 24 The inductive clauses that define $\circledR : \mu\mathsf{RRF}'' \to \mu\mathsf{RRF}''$ are those ones that define $\circledR : \mu\mathsf{RRF} \to \mu\mathsf{RRF}$ of Definition 16 plus the clauses:

$$\circledR(\mathsf{hide}_i(g)) = \mathsf{hide}_i(\circledR(g))$$
$$\circledR(\mathsf{uncover}_i(g)) = \mathsf{uncover}_i(\circledR(g))$$

which hold for every $g : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$ in $\mu\mathsf{RRF}$.

Theorem 9 *For every $f \in \mu\mathsf{RRF}''$, we have that $\circledR f \in \mu\mathsf{RRF}''$.*

6 Conclusions

This work naturally extends [25] where we introduce the class RPRF of (total) Reversible Primitive Recursive Functions. Coherently with the classical recursion theory, we here introduce the Turing-complete analogous of $\mu\mathsf{RPF}$ in the setting of reversible computations, i.e. $\mu\mathsf{RRF}$.

On the foundational side, we have some future goals.

- One is to consider a set of base reversible functions smaller than the one Definition 1 identifies. Then, it is possible to ask if RPRF built-in pairing and un-pairing are strictly necessary. The forthcoming work [24] shows that pairing and un-pairing are, in fact, definable in the language with a reduced set of base reversible functions.

– Another one is to further check the theoretical relevance of RPRF. We wonder about the existence of $\mathtt{T}_r$ in RPRF, the analogous of Kleene's predicate $\mathtt{T}$. The aim of looking for $\mathtt{T}_r$ is to prove a normalization theorem like the one that holds for μRPF [23,33]. This would amount to prove that every $g \in \mu$RRF can be described by means of the application of a first RPRF-function to a unique application of a recursion scheme to a further RPRF-function.
– More ambitiously we can think of extending the compositional nature of our recursion-theoretic characterizations of reversible functions in order to encompass higher order functions and functional programming languages. Starting points could be [1,9,7,26,19,16].

We also have at least a pair of pragmatic future goals.

– We aim at comparing the programming styles that our computational model supplies with those ones available inside Reversible Turing Machines as in [2,3] and other reversible programming languages [37,38].
– Another interesting issue is to find a more friendly notation for functions that have multiple outputs and therefore do not lend easily to a linear representation for composition. A natural approach should consider the many graphical formalisms for representing morphisms in monoidal categories, like the string diagrams described, for example, in [32].

Related Papers. The reversible programming language SRL is in [21]. Its programs represent total functions only. The characterization of the class of functions that SRL represents is till open. Our conjecture is that SRL would become equivalent to RPRF only once extended with stack-like data-types.

A language equivalent to μRPF is in [14]. It provides invertible partial recursive functions on natural numbers and not on integers. Working those functions on natural numbers, they necessarily depend on a specific representation of integers to encode the predecessor of μRPF. We see this as a useless restriction.

References

1. S. Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441–464, Dec. 2005.
2. H. B. Axelsen and R. Glück. What do reversible programs compute? In *14th International Conference on Foundations of Software Science and Computational Structures*, volume 6604 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 2011.
3. H. B. Axelsen and R. Glück. On reversible Turing machines and their function universality. *Acta Informatica*, 53(5):509–543, Aug 2016.
4. C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17:525–532, 1973.
5. C. H. Bennett. Notes on the history of reversible computation. *IBM J. Res. Dev.*, 32(1):16–23, Jan. 1988.
6. N. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.
7. V. Danos and L. Regnier. Reversible, irreversible and optimal λ-machines. *Theoretical Computer Science*, 227(1):79 – 97, 1999.

8. A. De Vos. *Reversible Computing: Fundamentals, Quantum Computing, and Applications*. Wiley-VCH, 2010.
9. A. Di Pierro, C. Hankin, and H. Wiklicky. Reversible combinatory logic. *Mathematical. Structures in Comp. Sci.*, 16(4):621–637, Aug. 2006.
10. G. Dowek. *Proofs and Algorithms: An Introduction to Logic and Computability*. Springer Publishing Company, Incorporated, 1st edition, 2011.
11. R. Glück and M. Kawabe. Revisiting an automatic program inverter for lisp. *SIGPLAN Not.*, 40(5):8–17, May 2005.
12. D. Gries. *The Science of Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1987.
13. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
14. G. Jacopini and P. Mentrasti. Generation of invertible functions. *Theoretical Computer Science*, 66(3):289–297, 1989.
15. G. Jacopini, P. Mentrasti, and G. Sontacchi. Reversible Turing machines and polynomial time reversibly computable functions. *SIAM J. Discrete Math.*, 3(2):241–254, 1990.
16. R. P. James and A. Sabry. Information effects. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 73–84, 2012.
17. S. Kleene. *Introduction to Metamathematics*. Bibliotheca Mathematica. Wolters-Noordhoff, 1952.
18. A. V. Kuznecov. On primitive recursive functions of large oscillation. *Doklady Akademii Nauk SSSR*, 71:233–236, 1950. In Russian.
19. I. Mackie. The geometry of interaction machine. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 198–208, New York, NY, USA, 1995. ACM.
20. A. I. Mal'cev. *Algorithms and recursive functions*. Wolters-Noordhoff, 1970. Translated from the first Russian ed. by Leo F. Boron, with the collaboration of Luis E. Sanchis, John Stillwell and Kiyoshi Iseki.
21. A. B. Matos. Linear programs in a simple reversible language. *Theoretical Computer Science*, 290(3):2063–2074, 2003.
22. J. McCarthy. The inversion of functions defined by Turing machines. In C. Shannon and J. McCarthy, editors, *Automata Studies, Annals of Mathematical Studies*, 34, pages 177–181. Princeton University Press, 1956.
23. P. Odifreddi. *Classical recursion theory: the theory of functions and sets of natural numbers*. Studies in logic and the foundations of mathematics. North-Holland, 1989.
24. L. Paolini, M. Piccolo, and R. Luca. A class of recursive permutations which is primitive recursive complete. Technical report, Rapporto dell'Università di Torino, 2017. Submitted to TCS.
25. L. Paolini, M. Piccolo, and L. Roversi. A class of reversible primitive recursive functions. *Electronic Notes in Theoretical Computer Science*, 322:227–242, 2016. Elsevier, Netherlands.
26. L. Paolini, M. Piccolo, and L. Roversi. A certified study of a reversible programming language. In T. Uustalu, editor, *TYPES 2015 postproceedings*, volume 69 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2017.
27. L. Paolini and M. Zorzi. qPCF: a language for quantum circuit computations. In T. Gopal, G. Jäger, and S. Steila, editors, *14th Annual Conference on Theory and Applications of Models of Computation*, volume 10185 of *Lecture Notes in Computer Science*, pages 455–469. Springer, Germany, 2017.
28. K. S. Perumalla. *Introduction to Reversible Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2013.
29. R. Péter. *Recursive functions*. Academic Press, 1967.
30. J. Robinson. General recursive functions. *Proceedings of the American Mathematical Society*, pages 703–718, 1950.
31. H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill series in higher mathematics. McGraw-Hill, 1967.
32. P. Selinger. *A Survey of Graphical Languages for Monoidal Categories*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

33. R. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer, 1987.
34. R. I. Soare. Chapter 1 - The History and Concept of Computability. In E. R. Griffor, editor, *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, pages 3 – 36. Elsevier, 1999.
35. M. P. Szudzik. The rosenberg-strong pairing function. *CoRR*, abs/1706.04129, 2017.
36. T. Toffoli. Reversible computing. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644. Springer, 1980.
37. T. Yokoyama, H. B. Axelsen, and R. Glück. Principles of a reversible programming language. In A. Ramírez, G. Bilardi, and M. Gschwind, editors, *Proceedings of the 5th Conference on Computing Frontiers, 2008, Ischia, Italy, May 5-7, 2008*, pages 43–54. ACM, 2008.
38. T. Yokoyama, H. B. Axelsen, and R. Glück. Fundamentals of reversible flowchart languages. *Theoretical Computer Science*, 611:87 – 115, 2016.
39. M. Zorzi. On quantum lambda calculi: a foundational perspective. *Mathematical Structures in Computer Science*, 26(7):1107–1195, 2016.