A Class of Reversible Primitive Recursive Functions

Luca Paolini[a,1,2], Mauro Piccolo[a,1,2] and Luca Roversi[a,1,2]

[a] *Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino*

Abstract

Reversible computing is bi-deterministic which means that its execution is both forward and backward deterministic, i.e. next/previous computational step is uniquely determined. Various approaches exist to catch its extensional or intensional aspects and properties. We present a class RPRF of reversible functions which holds at bay intensional aspects and emphasizes the extensional side of the reversible computation by following the style of Dedekind-Robinson Primitive Recursive Functions. The class RPRF is closed by inversion, can only express bijections on integers — not only natural numbers —, and it is expressive enough to simulate Primitive Recursive Functions, of course, in an effective way.

Keywords: Reversible computing, Recursive permutations, Primitive Recursive Functions.

1 Introduction

Reversible computing (sometimes called isentropic or adiabatic computing) is, on its own, an unconventional form of computing. Origins of reversible computing trace back to the study of entropy in physical systems [16]. The goal was relating thermodynamic properties of the system with the amount of information that it could carry around. In the sixties, Landauer was the first to define a technique for transforming irreversible computations into equivalent reversible ones [9]. Landauer thought his machines could not reversibly get rid of their undo trails. Lecerf first described a technique to uncompute histories [10], but he was unaware of the thermodynamic applications. Bennett [2] rediscovered Lecerf reversal. "Bennett's trick" corresponds to copying the output before uncomputing the undo trail, thereby showing for the first time reversible computations that could avoid entropy generation. The moral of these studies tells us that, if a physical system performs a logically irreversible operation then it must increase the entropy of the environment [19]. When a computational system erases a bit of information, it must dissipate $\ln 2 \times kT$ energy, where k is Boltzmann's constant and T is the temperature. For $T = 300$ Kelvins

[1] Partially supported by the LINTEL project.

[2] Email: luca.paolini@unito.it, mauro.piccolo@unito.it,luca.roversi@unito.it

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

(room temperature), this is about 2.9×10^{-21} Joules (roughly, the kinetic energy of a single air molecule at room temperature). Today's computers erase a bit of information (in the above sense) every time they perform a logic operation, so their hunger for energy is ever-increasing. Reversible computing can avoid to use irreversible operations and entropy increasing.

Here above we have recalled the Physics related aspects that make reversible computation relevant. From a Computer Science foundational point of view reversible computing is interesting because it subsumes classical computing: every computation in a classical model can be simulated by a reversible one [14]. Moreover, aspects of reversible computation are ubiquitous in everyday classical computations. We can find them in activities spanning from software verification to programming languages, passing through computer architectures, as well as part of innovative computing models, like quantum, bio, chemical and molecular ones.

Reversible Turing-machines. Foundational studies on the notion of "reversible computation" exist. They have been chiefly devoted to frame the thermodynamic relations between entropy and computation via Turing-machines [1,2,6]. A reversible Turing-machine is both deterministic (like a classical Turing-machine) and backward-deterministic, i.e. it is bi-directionally deterministic. The backward determinism allows to easily reverse the computation, viz. we can undo a reversible program step by step eventually re-establishing former situations [1]. Only recently, recursion-theoretic arguments have been surveyed with some degree of systematization in [1].

This work develops a starting proposal to a recursion theory of reversible functions, in the line of Dedekind-Robinson-Kleene.

Dedekind-Robinson-Kleene Functions. We start recalling the distinguishing aspects of Kleene's Partial Recursive Functions [7], that we simply call Partial Recursive Functions, abbreviated as (RF). These functions form an extension of the Dedekind-Robinson Primitive Recursive Functions (PRF) this paper starts from.

Our starting point are RF and PRF for various reasons. First, we want to manage entities that compose because they stand for and are written as functions. Second, RF, as well as PRF, balance intensional and extensional aspects. Intensionally, they can be taken as programming languages whose semantics is given informally. Extensionally, RF deals with *partial functions* [3] while PRF with *total ones*, both shifting the focus on functions closer to what other computational models can express and providing support to functional, or compositional, programming.

We aim at giving a prominent computational status to the operation of functional inverse. The inverse f^{-1} of a function f is defined by reversing its underlying relation [4], viz. $(y, x) \in f^{-1}$ if and only if $(x, y) \in f$.

[3] 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 *co-functional* 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 *co-total* 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. A function is injective whenever its graph is a co-functional relation. A function is surjective whenever its graph is a co-total relation.

[4] The inverse of a partial function may not be functional. The inverse of a total function may be not total. However, restricted (and effective) operation of inversion can be defined also in such cases, e.g. see [12].

We focus on bijections, because the inversion of a computable bijection is always a computable function, see [13, p.31].

A little bit more specifically, our goal is the synthesis of a formalism which, at least, describes a reasonable large class of first-order functions whose graphs can be effectively inverted inside the formalism itself. The simple and effective inversion operation that we propose in this work takes great advantage from the compositional nature of the computational model we introduce. As a side effect we shall eventually be able to compare the programming style our computational model supplies with those ones available inside the Reversible Turing Machines as in [1] and Janus [18].

Primitive Recursive functions. We develop our goal gradually. In this paper, we do not aim at the synthesis of a full analogous of RF which, we recall, is Turing-complete. We present what we think is a good candidate we can identify as being the analogous of PRF i.e. a class of terms which could capture all the algorithms developed in mathematics until the first part of XX Century, see [13]. PRF is not Turing-complete and corresponds to a core of RF which only contains everywhere-defined, i.e. total, functions. In analogy to PRF, we supply Reversible Primitive Recursive Functions (RPRF) which always terminate and keeps semantic redundancies at bay (with a relatively light abstract syntax).

We show that PRF can be simulated by RPRF, so RPRF functions inherit the following statement, typically associated to PRF:

"programs which terminate but do not belong to PRF *are rarely of practical interest".*

The main problem we must cope with is to identify the *right class of total functions* acting as the extensional model of reference. Identifying such a class is not obvious for the following reasons.

On one side, we might rely on [1]. It contains a statement saying that Reversible Turing Machines compute *injective* RF. The statement in [1] seems to suggest to consider the class of Injective Primitive Recursive Functions (JPRF) as our extensional model of reference to identify a functional language which is able to talk about reversible functions. This choice is doomed to failure. The reason is that JPRF is not closed under inversion. There is a function f 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.

The situation does not improve if we think of restricting our extensional model to BPRF, i.e. the class of all Bijective Primitive Recursive Functions. BPRF is strictly smaller than JPRF, but the problem becomes somewhat "bigger".

Theorem 1.1 (Kuznekov [8]) *There is an* $f \in$ BPRF *whose inverse* f^{-1} *does not even belong to* PRF.

Proof. See [15, Exercise 5.7, p.25]. We sketch the proof. Consider a total computable function whose rate of growth is too fast to be primitive recursive, e.g. consider the Ackermann-like function $A_0(x) = 2x$ and $A_{n+1}(x) = \underbrace{A_n \ldots A_n}_{x}$ (1).

It is possible to prove that $\{< x, y > | A_x(x) = y\}$ is *primitive recursive*, although the injective increasing function $x \mapsto A_x(x)$ is not. Let R denote the range of $x \mapsto A_x(x)$, i.e. $R = \{y | \exists x,\ A_x(x) = y\}$. R is infinite and co-infinite. The predicate

checking $y \in R$ is still *primitive recursive*, because $\bigvee_{x=0}^{y-1} A_x(x) = y$ is *primitive recursive*. For an infinite set $S \subseteq \mathbb{N}$, let $\pi_s : \mathbb{N} \to \mathbb{N}$ be defined as $\pi_S(0) \stackrel{def}{=} \min(S)$ and $\pi_S(n+1) \stackrel{def}{=} \min(S - \{\pi_S(0), \ldots, \pi_S(n)\})$. Patently, if S is primitive recursive then π_s is a primitive recursive bijection (by bounded minimization). Let f be the permutation of $\mathbb{N}$ defined by

$$f(y) = \begin{cases} 2\pi_R^{-1}(y) & \text{if } y \in R, \\ 2\pi_{\mathbb{N}\setminus R}^{-1}(y) + 1 & \text{if } y \in \mathbb{N}\setminus R \end{cases} .$$

Despite f is primitive recursive, its inverse f^{-1} is not, because $f^{-1}(2x) = \pi_R(x) = A_x(x)$. □

Corollary 1.2 BPRF *and general recursive bijections*[5] *are two different classes of functions. In particular, primitive recursive permutations and general recursive permutations are different classes of functions.*

Proof. Let f be the permutation defined in the proof of Theorem 1.1. □

Moreover, it is not possible to enumerate general recursive permutations because it would provide an effective enumeration of total recursive functions as well.

Theorem 1.3 *General recursive permutations cannot be recursively enumerated.*

Proof. Assume $\phi_0, \ldots, \phi_n, \ldots$ be a such effective enumeration. We aim to build a permutation ψ (viz. a bijection on $\mathbb{N}$) different from all enumerated ones.

- Assume that there is a $k \in \mathbb{N}$ such that $\phi_0(0) \neq \phi_k(k)$. We can define ψ by induction, $\psi(0) \stackrel{def}{=} \phi_k(k)$ and $\psi(n+1) \stackrel{def}{=} \min(\mathbb{N} - \{\psi(0), \ldots, \psi(n), \phi_{n+1}(n+1)\})$.
- Let $h = \phi_0(0)$. If $\phi_i(i) = h$ for each $i \in \mathbb{N}$ then, let $\psi(0) \stackrel{def}{=} h$. Clearly, ψ is different from all ϕ_{i+1}, because each involved function is a bijection. Let $m \in \mathbb{N}$ be such that $m \neq \phi_0(0) = h$ and $m \neq \phi_0(1)$; we can define $\psi(1) \stackrel{def}{=} m$ (so ψ is different of ϕ_0 on 1). It is easy to complete the definition of ψ in a suitable way, e.g. $\psi(n+2) \stackrel{def}{=} \min(\mathbb{N} - \{\psi(0), \ldots, \psi(n+1)\})$.

□

The above properties forcefully addressed us to look for a class of total computable functions F whose main features are that (i) every element of PRF has a faithful counterpart in F, (ii) F is rich enough to represent $\mathbb{N}$ by means of a suitable encoding and (iii) F can be represented in PRF.

Reversible Primitive Recursive Functions (RPRF). It is the class of functions we propose to fulfill our goals. RPRF includes only bijections and is closed under the meta-operation of inversion which is effective. Moreover, RPRF is expressive enough to represent PRF and, in particular, the gödelization.

Related Papers. In [11] a reversible for-language has been proposed and studied. The programs of such a language only expresses total reversible functions.

[5] Recall that the class of general recursive functions is formed by all and only total recursive functions. Recall that endo-bijections are called permutations.

The language is presented in a very appealing programming style which does not make the comparison with the terms of RPRF immediate. The work [11] mainly aims at an algebraic and programming study of reversibility. In particular, it does not characterize the set of functions that can be represented. Our conjecture is that the language in [11] would be equivalent to the set of functions in RPRF only once extended with stack-like data-types.

A work that introduces a language related to RF is [5]. First it provides a representation of the full set of invertible partial recursive functions between natural numbers, unlike ours. Moreover, a second difference with RPRF is that the language in [5] heavily depends on the explicit binary representation of numbers. This difference is essential for us, since we aim at characterizations independent from the data representation as for Dedekind-Robinson-Kleene ones.

2 Reversible Primitive Recursive Functions

We introduce the class of *Reversible Primitive Recursive Functions* that we abbreviate as RPRF. RPRF includes only total functions. Among them we show that all primitive recursive functions can be represented by using an embedding technique that recalls the ones we can find in [2,17,18]. One novelty of RPRF, as compared to the class of primitive recursive functions (and classic recursion theory), is that its functions operate on integers instead than on natural numbers (as first proposed in [11]). The reason is that natural numbers do not form a group (endowed by inverses) with standard operations. 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 [6]. 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 sake of simplicity, we restrict ourselves to consider only arity-respecting functions so to include only permutations. Finally, let $\langle\_, \_\rangle : \mathbb{Z}^2 \to \mathbb{Z}$ be a given bijection. For our purposes its full definition is irrelevant [7].

Definition 2.1 [***Reversible Primitive Recursive functions***] The set RPRF of Reversible Primitive Recursive functions contains *total* functions from $\mathbb{Z}^k$ to $\mathbb{Z}^k$, for every $k \geq 0$. We inductively define RPRF as follows.

- RPRF includes the successors $\mathsf{S}_i(x_1, \ldots, x_i, \ldots, x_k) = (x_1, \ldots, x_i + 1, \ldots, x_k)$ and the predecessors $\mathsf{P}_i(x_1, \ldots, x_i, \ldots, x_k) = (x_1, \ldots, x_i - 1, \ldots, x_k)$ where $1 \leq i \leq k$.
- RPRF includes every *finite permutation* of a k-tuple, defined as follows. Let $\ell = i_1, \ldots, i_k$ be an ordered list of pairwise distinct natural numbers from 1 through k. The associated finite permutation is $\mathsf{fP}_\ell(x_1, \ldots, x_k) = (x_{i_1}, \ldots, x_{i_k})$.
- Let $j, k \in \mathbb{N}$ be greater than 1 and let $k_1, \ldots, k_j \in \mathbb{N}$ be such that $k = \sum_{i=1}^{j} k_i$. For every $1 \leq i \leq j$, let $g_i : \mathbb{Z}^{k_i} \longrightarrow \mathbb{Z}^{k_i}$ and $f : \mathbb{Z}^k \longrightarrow \mathbb{Z}^k$ belonging to RPRF. The *series composition* of f with $g_1, \ldots, g_j$ from $\mathbb{Z}^k$ to $\mathbb{Z}^k$ is $\circ[f; g_1, \ldots, g_j](\vec{x}_1, \ldots, \vec{x}_j) =$

[6] We recall that $\mathbb{N}$ and $\mathbb{Z}$ are in bijection, see [3, Example 5.1].

[7] For those who are curious, $\langle\_, \_\rangle$ can be defined by suitably composing the bijection between $\mathbb{N} \leftrightarrow \mathbb{Z}$ as in the Example 5.1 of [3] and Cantor's pairing variant defined in [4] which is a bijection as well.

$f(g_1(\vec{x}_1),\ldots,g_j(\vec{x}_j))$ and belongs to RPRF. Of course, for every $1 \leq i \leq n$, we assume that $\vec{x}_i$ contains k_i elements.

- RPRF includes 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 arguments but the one in position h. This latter is incremented by $\langle\!\langle x_i, x_j \rangle\!\rangle$. The function $\mathsf{subPair}_h^{(i,j)}$ is the identity on all arguments but for the one in position h. This latter is decremented by $\langle\!\langle x_i, x_j \rangle\!\rangle$. For example, $\mathsf{addPair}_1^{(2,3)}(n, x, y, \ldots) = (n + \langle\!\langle x, y \rangle\!\rangle, x, y, \ldots)$ and $\mathsf{subPair}_1^{(2,3)}(n, x, y, \ldots) = (n - \langle\!\langle x, y \rangle\!\rangle, x, y, \ldots)$ where $1 \leq i < j \leq k$.
- RPRF includes the unpairing 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-th. They are *incremented* by x and y, respectively, if $\langle\!\langle x, y \rangle\!\rangle$ 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-th. They are *decremented* by x and y, respectively, if $\langle\!\langle x, y \rangle\!\rangle$ is the argument of position h. For instance, $\mathsf{addUnPair}_1^{(2,3)}(\langle\!\langle x', y' \rangle\!\rangle, x, y, \ldots) = (\langle\!\langle x', y' \rangle\!\rangle, x + x', y + y', \ldots)$ and $\mathsf{subUnPair}_1^{(2,3)}(\langle\!\langle x', y' \rangle\!\rangle, x, y, \ldots) = (\langle\!\langle x', y' \rangle\!\rangle, x - x', y - y', \ldots)$ where $1 \leq i < j \leq k$.
- Let $f, g, h : \mathbb{Z}^k \longrightarrow \mathbb{Z}^k$ be elements of RPRF. For every $n \geq 0$, let $f^{\ddot{n}}$ denote $\circ[f; \ldots \circ [f; f]\ldots]$ with n occurrences of f. RPRF includes the *recursive scheme* $\mathsf{Rec}^i[f, g, h] : \mathbb{Z}^{k+1} \to \mathbb{Z}^{k+1}$, for each i such that $1 \leq i \leq k+1$, defined as follows:

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

and belongs to RPRF. Of course, we assume that $\vec{x_0}, \vec{x_1}$ contain $i-1$ elements, while $\vec{z_0}$ and $\vec{z_1}$ contain $(k+1)-i$ elements. □

Definition 2.1 follows the pattern that drives the definition of Primitive Recursive Functions (PRF) but shows distinctive features. Among the basic functions RPRF explicitly contains the predecessor, unlike PRF. Projections are missing from RPRF, because intrinsically irreversible. We "replace" them with all finite permutations.

For sake of completeness, we remark that $\mathsf{subUnPair}_h^{(i,j)}, \mathsf{subPair}_h^{(i,j)}$ can be defined in terms of $\mathsf{addUnPair}_h^{(i,j)}, \mathsf{addPair}_h^{(i,j)}$. This means that $\mathsf{addUnPair}_h^{(i,j)}, \mathsf{addPair}_h^{(i,j)}$ and $\mathsf{addUnPair}_h^{(i,j)}, \mathsf{addPair}_h^{(i,j)}$ are pairwise interdefinable. The only existing definition of a class of functions we are aware of and which makes use of pairing and unpairing is by Bernays and Robinson, see [13, p.72]. They used pairing and unpairing for a non standard definition of PRF, like we do. The main motivation to let pairing and unpairing available is to pack the information. A single argument can be used as a kind of store, like the coming example shows. The composition $\mathsf{fP}_{2,1,3,4} \circ \mathsf{subUnPair}_2^{(1,4)} \circ \mathsf{addPair}_2^{(1,4)} \circ \mathsf{subUnPair}_1^{(2,3)} \circ \mathsf{addPair}_1^{(2,3)}$ transforms $(0, x_2, x_3, x_4)$ in $(\langle\!\langle\langle\!\langle x_2, x_3 \rangle\!\rangle, x_4 \rangle\!\rangle, 0, 0, 0)$ as follows:

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

We remark that adding pairing-unpairing functions (non arity-respecting, similar to that of Bernays and Robinson) to RPRF will still provide a correct model of reversible computation.

The composition among elements of RPRF has no major differences with the composition scheme of PRF. The recursion scheme is, in fact, an iterator. It iteratively applies one of the three parameters in RPRF as many times as the value of the argument in position i if $x_i \neq 0$, one time otherwise. The value of that argument is not passed to the iterated function. Instead, it is preserved by the whole evaluation and 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.

We define some functions that will be useful later. 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 from the context, we omit the arity apex. Given two functions $f, g : \mathbb{Z}^k \to \mathbb{Z}^k$, we abbreviate $\circ[f; g]$ by means of the more standard $f \circ g$. Let $f_i : \mathbb{Z}^{k_i} \longrightarrow \mathbb{Z}^{k_i}$ and let $\vec{x}_i$ contains 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 $(f_1 \parallel \ldots \parallel f_n)(\vec{x}_1, \ldots, \vec{x}_n) \stackrel{def}{=} \mathsf{Id}^{k_1+\cdots+k_n}(f_1(\vec{x}_1), \ldots, f_n(\vec{x}_n))$.

We define an inversion operation that maps RPRF to RPRF in an effective way.

Definition 2.2 The function $\circledR : \mathsf{RPRF} \to \mathsf{RPRF}$ is defined inductively as follows.

- $\circledR(\mathsf{S}) \stackrel{def}{=} \mathsf{P}$ and $\circledR(\mathsf{P}) \stackrel{def}{=} \mathsf{S}$.
- For each permutation fP_ℓ, $\circledR(\mathsf{fP}_\ell)$ is equal to the (unique) finite permutation $\mathsf{fP}_{\ell'}$ that inverts fP_ℓ. Patently, this permutation always exists and belong to RPRF.
- $\circledR(\circ[f; g_1, \ldots, g_j]) \stackrel{def}{=} \circ[(\circledR(g_1) \parallel \ldots \parallel \circledR(g_n)); \circledR(f)]$.
- $\circledR(\mathsf{addPair}_h^{(i,j)}) \stackrel{def}{=} \mathsf{subPair}_h^{(i,j)}$ and $\circledR(\mathsf{subPair}_h^{(i,j)}) \stackrel{def}{=} \mathsf{addPair}_h^{(i,j)}$.
- $\circledR(\mathsf{addUnPair}_h^{(i,j)}) \stackrel{def}{=} \mathsf{subUnPair}_h^{(i,j)}$ and $\circledR(\mathsf{subUnPair}_h^{(i,j)}) \stackrel{def}{=} \mathsf{addUnPair}_h^{(i,j)}$.
- $\circledR(\mathsf{Rec}^i[f, g, h]) \stackrel{def}{=} \mathsf{Rec}^i[\circledR(f), \circledR(g), \circledR(h)]$.

Indeed RPRF is closed under inversion in the following strong sense.

Theorem 2.3 (RPRF is closed under inversion) *If $f : \mathbb{Z}^k \to \mathbb{Z}^k$ 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 2.2. □

Corollary 2.4 *Each* RPRF *is a bijective function on $\mathbb{Z}^k$, for some $k \in \mathbb{N}$.*

Theorem 2.3 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 [11], 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, indeed.

2.1 Expressiveness of RPRF

We define some functions in RPRF. First we want to suggest how Bennett's technique fits into RPRF. Second, we want to give a flavor about the expressiveness of the class of reversible functions we just introduced. 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:

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

  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, see Definition 2.1. 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:

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

  If $j < i$ then we can 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 $®(\mathsf{dec}^i_j) = \mathsf{inc}^i_j$.

- 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 $1 \leq i, j, h \leq 2+k$. We remark that $\mathsf{sum}(9, 5, -3) = (11, 5, -3)$. Generally speaking, the sum of two numbers needs an argument initialized to zero. This is a the typical side-effect of representing an inherently non-reversible function by a reversible one. To avoid such a side effect, [2] uses a third tape and [17] uses some input-constant.

- We define $\mathsf{mult} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ by means of $\mathsf{Rec}^3[\mathsf{inc}^2_1, \mathsf{Id}, \mathsf{dec}^2_1]$. It is easy to verify that $\mathsf{mult}(n, x_1, x_2, \ldots) = (n + \underbrace{x_1 + \ldots + x_1}_{x_2}, x_1, x_2, \ldots)$. and, on the other hand,

$\circledR(\mathsf{mult})(n, x_1, x_2, \ldots) = (n - (\underbrace{x_1 + \ldots + x_1}_{x_2}), x_1, x_2, \ldots)$. The function $\mathsf{mult}_h^{(i,j)} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$, which adds the products of the i,j-th arguments to the h-th one, is:

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

can be defined by $\mathsf{mult}_h^{(i,j)} = \mathsf{Rec}^j[\mathsf{inc}_h^i, \mathsf{Id}, \mathsf{dec}_h^i]$, provided that $1 \leq h < i < j \leq 3 + k$. It is easy to adapt the previous definition to all ordering of i, j, h provided that they are pairwise distinct.

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

The *coming 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 $\mathsf{totalNatPred} : \mathbb{Z}^{2+k} \to \mathbb{Z}^{2+k}$ can be defined as $\mathsf{S}_2 \circ \mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{P}_2$. The defined function grants 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 $\mathsf{totalNatMinus} : \mathbb{Z}^{3+k} \to \mathbb{Z}^{3+k}$ can be defined as $\mathsf{inc}_2^3 \circ \mathsf{Rec}^2[\mathsf{S}_1, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{dec}_2^3$ (where dec is defined above), that is $\mathsf{totalNatMinus}(0, x_1, x_2, \ldots) = ((x_1 \dot{-} x_2), x_1, x_2, \ldots)$.
- We define the factorial $\mathsf{fact} : \mathbb{Z}^{6+k} \to \mathbb{Z}^{6+k}$ such that, whenever $x \geq 0$ we have $\mathsf{fact}(0, x, 0, 0, 0, z, \ldots) = (x!, x, x, x!, z', \ldots)$.

  · Let $\ell_{1,3}$ be the finite list swapping first and third arguments and let $\ell_{4,5}$ be the finite list swapping 4th and 5th arguments, being the identity elsewhere.
  · Let clean_3 be $\mathsf{fP}_{\ell_{4,5}} \circ \mathsf{subUnPair}_4^{(3,5)} \circ \mathsf{addPair}_4^{(3,5)}$; it is easy to see that it satisfies $\mathsf{clean}_3(x_1, x_2, x_3, 0, x_5, \ldots) = (x_1, x_2, 0, 0, \langle\!\langle x_3, x_5 \rangle\!\rangle, \ldots)$.

  Finally, fact is $\mathsf{Rec}^2[\mathsf{clean}_3 \circ \mathsf{mult}_1^{(2,3)} \circ \mathsf{S}_2 \circ \mathsf{fP}_{\ell_{1,3}}, \mathsf{Id}, \mathsf{Id}] \circ \mathsf{S}_1$. (Remind that the recursion hide an argument).

3 Primitive Recursive functions and RPRF

We recall the class of Primitive Recursive Functions (PRF), see for instance [3,13]. It is the smallest class of functions on natural numbers:

- which contains the functions $\mathsf{0}(\vec{x}) = 0$, the successor $\mathsf{S}(x) := x + 1$ and the projections $\pi_i^k(x_1, \ldots, x_k) := x_i$ for all $k \geq i \geq 1$,

- which is closed under composition, i.e. the schema that given $g_1, \ldots, g_m, h$ of suitable arities, produces $f(\vec{x}) := h(g_1(\vec{x}), \ldots, g_m(\vec{x}))$, and
- which is closed under primitive recursion, viz. the function f which is defined from g and h by means of the schema $f(\vec{x}, 0) := g(\vec{x})$ and $f(\vec{x}, y+1) := h(f(\vec{x}, y), \vec{x}, y)$.

In coming subsections we show the computational equivalence of PRF and RPRF.

3.1 From PRF *to* RPRF

We present a formal correspondence between PRF and RPRF. Specifically, 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 $\mathbb{N} \subseteq \mathbb{Z}$.

Definition 3.1 [RPRF-***definable functions***] A function $f : \mathbb{N}^k \to \mathbb{N}$ is RPRF_h^k-definable (for $h \geq 3$) whenever there exists a function $\overline{\overline{f}} : \mathbb{Z}^{k+h} \to \mathbb{Z}^{k+h}$ in RPRF such that, for all $x_1, \ldots, x_k, z \in \mathbb{N}$, if $f(x_1, \ldots, x_k) = y$ then

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

for some $z' \in \mathbb{N}$. □

We write $\overline{\overline{f}} \in \mathsf{RPRF}_h^k$ to denote that $\overline{\overline{f}}$ represents f which is RPRF_h^k-definable. Some remarks on Definition 3.1 are in order. Extensionally, every $\overline{\overline{f}}$ behaves as an identity on all its arguments, but on the first and the last ones. This means that every argument with position $2 \leq i \leq k+h-1$ are moved to the output. (We remark that in the intensional "intermediate computation steps" these arguments can be altered). 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 first output which contains the value $f(x_1, \ldots, x_k)$ of the function we encode. Only the presence of the first argument makes the input and the output arities equal.

Lemma 3.2 (***"Weakening" on the map*** $\overline{\overline{(\ )}}$) *For every $f : \mathbb{N}^k \to \mathbb{N}$, if f is RPRF_h^k-definable, then f is also RPRF_{h+1}^k-definable.*

Lemma 3.2 holds because, if $f : \mathbb{N}^k \to \mathbb{N}$ is defined by $\overline{\overline{f}} \in \mathsf{RPRF}_h^k$ for some $h \geq 3$ then $\circledR\mathsf{fP}_\ell \circ (\overline{\overline{f}} \parallel \mathsf{Id}) \circ \mathsf{fP}_\ell$, where $\ell = 1, \ldots, k+h+1, k+h$, represents f in RPRF_{h+1}^k. We remark that $\circledR\mathsf{fP}_\ell = \mathsf{fP}_\ell$.

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

Definition 3.3 *We call n^S-tuple each tuple of $\mathbb{N}^n$ such that $S \subseteq \{1, \ldots, n\}$ and for all $i \in S$ the i-th position of the tuple is 0 (no assumptions are done on the other positions). For any $n \in \mathbb{N}$ such that $n \geq 3$, let $\ell = 1, \ldots, n, n-1$.*
We denote push_i the term $\mathsf{fP}_\ell \circ \mathsf{subUnPair}_{n-1}^{(i,n)} \circ \mathsf{addPair}_{n-1}^{(i,n)}$ that maps a $n^{\{n-1\}}$-tuple

to a $n^{\{i,n-1\}}$-tuple as follows:

$$\mathsf{push}_i(\ldots,x_{i-1},x_i,x_{i+1},\ldots,0,x_n) = (\ldots,x_{i-1},0,x_{i+1},\ldots,0,\langle\!\langle x_i,x_n\rangle\!\rangle) \ .$$

Symmetrically, we denote pop_i the term $\mathsf{subPair}_{n-1}^{(i,n)} \circ \mathsf{addUnPair}_{n-1}^{(i,n)} \circ \mathsf{fP}_\ell$ that maps a $n^{\{i,n-1\}}$-tuple to a $n^{\{n-1\}}$-tuple as follows:

$$\mathsf{pop}_i(\ldots,x_{i-1},0,x_{i+1},\ldots,0,\langle\!\langle x_i,x_n\rangle\!\rangle) = (\ldots,x_{i-1},x_i,x_{i+1},\ldots,0,x_n) \ .$$

□

The condition "$n \geq 3$" of Definition 3.3 is an instance of the inequality "$h \geq 3$" in Definition 3.1. The operations of Definition 3.3 act on the last argument, but they use an auxiliary argument (the second last one) to be performed. This fact justifies the introduction of the constraint "$h \geq 3$" of Definition 3.1 whose ultimate meaning is to assure that: (i) the first argument returns the output of the defined function, (ii) the last argument is a sort of waste bin, and (iii) the penultimate input and output positions serve to correctly apply push_i and pop_i.

We can finally state the main theorem of this work.

Theorem 3.4 *Every $f \in \mathsf{PRF}$ is RPRF-definable.*

The proof of Theorem 3.4 is by induction on the definition of $f \in \mathsf{PRF}$. For 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 can define $\overline{\overline{\mathsf{0}}} := \mathsf{Id}^{k+3}$ with input and output arity $k+3$. The definition should not surprise because $\overline{\overline{\mathsf{0}}}(0,x_1,\ldots,x_k,0,y) = (0,x_1,\ldots,x_k,0,y)$, so it is RPRF_3^k-definable.
- Let f be $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$ for some fixed k. We can define $\overline{\overline{\pi_i^k}} := \mathsf{inc}_1^{i+1}$. ($\mathsf{inc}_1^{i+1}$ is defined in Section 2.1.) So, $\overline{\overline{\pi_i^k}}(0,x_1,\ldots,x_k,0,y) = (0+x_i,x_1,\ldots,x_k,0,y)$ and projections are RPRF_3^k-definable.
- Let f be $\mathsf{S} : \mathbb{N} \to \mathbb{N}$. We can define $\overline{\overline{\mathsf{S}}} := \mathsf{S}_1 \circ \mathsf{inc}_1^2$ with input and output arity $1+3$. So, $\overline{\overline{\mathsf{S}}}(0,x_1,0,y) = \mathsf{S}_1(0+x_1,x_1,0,y) = (x_1+1,x_1,0,y)$ and the PRF-successor is RPRF_3^1-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. Let g_i be $\mathsf{RPRF}_{l_i}^k$-definable, for all i such that $1 \leq i \leq l_i$, and let h be $\mathsf{RPRF}_{l_h}^m$-definable. Assume $l = \max\{l_1,l_2,l_3,l_h\}$ and $l' = 3 + m \cdot (k+l)$ so that, there are $\overline{\overline{g_1}},\overline{\overline{g_2}},\overline{\overline{g_3}} \in \mathsf{RPRF}_l^k$ and $\overline{\overline{h}} \in \mathsf{RPRF}_{l'-m}^m$. We aim at building a $\overline{\overline{f}} \in \mathsf{RPRF}_{l'}^k$ defining f.

  For sake of simplicity, we do not present the more general case but we discuss in detail the case $m = 3$, $k = 2$ and $l = 3$, which shows all the technical problems. Still, to simplify the reading we proceed step-by-step.

  (i) We are looking for a $\overline{\overline{f}} \in \mathsf{RPRF}_{18}^2$ thus, by Definition 3.1, we expect an input of the shape $0,x_1,x_2,\underbrace{0,\ldots,0}_{15},0,z$ with 20 arguments.

  (ii) We want to predispose the arguments for $\mathsf{Id}^3 \parallel \overline{\overline{g_1}} \parallel \overline{\overline{g_2}} \parallel \overline{\overline{g_3}} \parallel \mathsf{Id}^2$ that belongs to $\mathbb{Z}^{20} \to \mathbb{Z}^{20}$ because $\overline{\overline{g_1}},\overline{\overline{g_2}},\overline{\overline{g_3}} \in \mathsf{RPRF}_3^2$. So, we apply the next RPRF-functions:

(a) $\mathsf{inc}^2_{15} \circ \mathsf{inc}^2_{10} \circ \mathsf{inc}^2_5$ produces $0, x_1, x_2, \underbrace{0, x_1, 0, 0, 0}_{5}, \underbrace{0, x_1, 0, 0, 0}_{5}, \underbrace{0, x_1, 0, 0, 0}_{5}, 0, z$;

(b) $\mathsf{inc}^3_{16} \circ \mathsf{inc}^3_{11} \circ \mathsf{inc}^3_6$ produces $0, x_1, x_2, \underbrace{0, x_1, x_2, 0, 0}_{5}, \underbrace{0, x_1, x_2, 0, 0}_{5}, \underbrace{0, x_1, x_2, 0, 0}_{5}, 0, z$.

(iii) The application of $\mathsf{Id}^3 \parallel \overline{\overline{g_1}} \parallel \overline{\overline{g_2}} \parallel \overline{\overline{g_3}} \parallel \mathsf{Id}^2$ produces

$$0, x_1, x_2, \underbrace{g_1(x_1, x_2), x_1, x_2, 0, z_1}_{5}, \underbrace{g_2(x_1, x_2), x_1, x_2, 0, z_2}_{5}, \underbrace{g_3(x_1, x_2), x_1, x_2, 0, z_3}_{5}, 0, z\ .$$

(iv) Now, we predispose the arguments for the application of $\mathsf{Id}^2 \parallel \overline{\overline{h}}$ where $\overline{\overline{h}} \in \mathsf{RPRF}^3_{15}$ by pushing useless values on our "stack", by erasing copies of x_1 and x_2 and by permuting arguments:

(a) let $z^* = \langle\!\langle z_1, \langle\!\langle z_2, \langle\!\langle z_3, z \rangle\!\rangle \rangle\!\rangle$, so $\mathsf{push}_{18} \circ \mathsf{push}_{13} \circ \mathsf{push}_8$ produces

$$0, x_1, x_2, \underbrace{g_1(x_1, x_2), x_1, x_2, 0, 0}_{5}, \underbrace{g_2(x_1, x_2), x_1, x_2, 0, 0}_{5}, \underbrace{g_3(x_1, x_2), x_1, x_2, 0, 0}_{5}, 0, z^*\ ;$$

(b) $\mathsf{dec}^3_{16} \circ \mathsf{dec}^2_{15} \circ \mathsf{dec}^3_{11} \circ \mathsf{dec}^2_{10} \circ \mathsf{dec}^3_6 \circ \mathsf{dec}^2_5$ produces

$$0, x_1, x_2\, \underbrace{g_1(x_1, x_2), 0, 0, 0, 0}_{5}, \underbrace{g_2(x_1, x_2), 0, 0, 0, 0}_{5}, \underbrace{g_3(x_1, x_2), 0, 0, 0, 0}_{5}, 0, z^*\ ;$$

(c) a suitable finite permutation fP_{ℓ_2} can produce

$$x_1, x_2, \overbrace{0, g_1(x_1, x_2), g_2(x_1, x_2), g_3(x_1, x_2), \underbrace{0, \ldots, 0}_{12}, 0, \langle\!\langle z_1, \langle\!\langle z_2, \langle\!\langle z_3, z \rangle\!\rangle \rangle\!\rangle}^{18}\ ;$$

(v) Since $f(x_1, x_2) = h(g_1(x_1, x_2), g_2(x_1, x_2), g_3(x_1, x_2))$, the application of $\mathsf{Id}^2 \parallel \overline{\overline{h}}$ produces $x_1, x_2, \overbrace{f(x_1, x_2), g_1(x_1, x_2), g_2(x_1, x_2), g_3(x_1, x_2), \underbrace{0, \ldots, 0}_{12}, 0, z_4}^{18}$.

(vi) By applying $\mathsf{push}_4 \circ \mathsf{push}_5 \circ \mathsf{push}_6$ we push $g_1(x_1, x_2), g_2(x_1, x_2), g_3(x_1, x_2)$ on the "stack".

(vii) We conclude by permuting the $f(x_1, x_2)$ with the first two arguments. Thus we obtain, $f(x_1, x_2), x_1, x_2, \underbrace{0, \ldots, 0}_{15}, 0, z_5$ that respecting the Definition 3.1 makes f a RPRF^2_{18}-definable function.

- Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ be a PRF defined by means of $h : \mathbb{N}^{k+2} \to \mathbb{N}$ and $g : \mathbb{N}^k \to \mathbb{N}$, i.e. by means of the schema $f(\vec{x}, 0) := g(\vec{x})$ and $f(\vec{x}, y{+}1) := h(f(\vec{x}, y), \vec{x}, y)$. Let h be $\mathsf{RPRF}^{k+2}_{l_h}$-definable and let g is $\mathsf{RPRF}^k_{l_g}$-definable. Assume $l = \max\{l_g, 3{+}l_h\}$ so that, there are $\overline{\overline{g}} \in \mathsf{RPRF}^k_l$ and $\overline{\overline{h}} \in \mathsf{RPRF}^{k+2}_{l-3}$. We aim at building a $\overline{\overline{f}} \in \mathsf{RPRF}^{k+1}_l$.

  For sake of simplicity, we do not present the more general case but we discuss in detail the case $k = 2$, $l_g = 3$ and $l_h = 5$, which shows all the technical problems. Still, to simplify the reading, we proceed step-by-step. We remind that the evaluation of the PRF function $f(\vec{x}, n)$ starts by evaluating $g(\vec{x})$ and proceeds by iteratively applying h as many times as n.

(i) We are looking for $\overline{\overline{f}} \in \mathsf{RPRF}^3_8$ thus, by Definition 3.1, we would expect an input of the shape $0, x_1, x_2, y, 0, 0, 0, 0, 0, 0, z$ containing 11 arguments.

(ii) We want to predispose the arguments for $\mathsf{Id}^1 \parallel \overline{\overline{g}}$ that, belongs to $\mathbb{Z}^{11} \to \mathbb{Z}^{11}$ because $\overline{\overline{g}} \in \mathsf{RPRF}_8^2$. Thus, we apply a suitable finite permutation prefixing y to the remaining argument-list.

(iii) The application of $\mathsf{Id}^1 \parallel \overline{\overline{g}}$ produces $y, g(x_1, x_2), x_1, x_2, 0, 0, 0, 0, 0, 0, z$.

(iv) 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 2.1. 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 $\overline{\overline{f}}$ 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, serves to provide 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 $\mathsf{fP}_{\ell_3} \circ \mathsf{push}_2 \circ \mathsf{S}_5 \circ \overline{\overline{h}}$, 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 respect Definition 3.1 (because the fifth argument), so h^* does not define a PRF function. However, it is sufficient that the $\overline{\overline{f}}$ (we want define) respects Definition 3.1.

(v) 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'.

(vi) We can conclude by eliding the a copy of y by applying dec_1^6 and then applying a suitable finite permutation moving the first two arguments just before the last one. Hence, f is RPRF_{18}^2-definable.

3.2 From RPRF *to* PRF

We think that the mere intuition should convince that whatever we can compute inside RPRF we can also compute inside PRF. This subsection contains basic hints to make the intuition concrete.

We start recalling from [3, Example 5.1] the two following isomorphic maps

$\alpha : \mathbb{Z} \to \mathbb{N}$ and $\alpha^{-1} : \mathbb{N} \to \mathbb{Z}$:

$$\alpha(x) = \begin{cases} 2x & \text{if } x \geq 0 \\ -2x-1 & \text{if } x < 0 \end{cases} , \qquad \alpha^{-1}(x) = \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ -\frac{x+1}{2} & \text{if } x \text{ is odd} \end{cases} .$$

Then, we fix the meaning of representing every multi-output element of RPRF by means of single-output elements in PRF. For every $f \in \mathsf{RPRF}$ with arity k, the map $\underline{\underline{(\ )}}$ supplies a family $\underline{\underline{f}} = \{f_i : \mathbb{N}^k \to \mathbb{N}\}_{1 \leq i \leq k}$ of functions in PRF which satisfy the following constraints:

$$(\alpha^{-1} \circ \pi_i \circ f)(x_1, \ldots, x_k) = f_i(\alpha(x_1), \ldots, \alpha(x_k)) \qquad 1 \leq i \leq k \ .$$

We do not supply a full blown definition of $\underline{\underline{(\ )}}$. We just give the translation of some of the basic functions of RPRF. To that purpose, let us recall that we can define the sum $+$, the multiplication $\times$ and the following functions as elements of PRF:

$$\mathtt{Not}(x) = \begin{cases} 1 & \text{if } x \text{ is } 0 \\ 0 & \text{if } x \text{ is } 1 \end{cases} \qquad \mathtt{isZ?}(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathtt{isE?}(x) = \begin{cases} 1 & \text{if } x \text{ is even} \\ 0 & \text{otherwise} \end{cases} \qquad \mathtt{isO?}(x) = \begin{cases} 1 & \text{if } x \text{ is odd} \\ 0 & \text{otherwise} \end{cases} .$$

The inductive definition of $\underline{\underline{(\ )}}$ proceeds by cases on its argument. We start making $\underline{\underline{\mathsf{P}}}$ explicit because the *predecessor* is a sort of conceptual pivot around which many of our design choices relative to RPRF rotate. Inside PRF the *predecessor* implements the following function on $\mathbb{N}$:

$$\mathtt{P}(x) = \begin{cases} x-1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} .$$

It cannot break through the barrier that 0 represents. Instead, the *predecessor* of RPRF is primitive and can decrease any of its arguments. It follows that $\underline{\underline{\mathsf{P}}}$ must allow to move along even numbers to simulate $\mathsf{P} \in \mathsf{RPRF}$ on positive values, and along the odd ones to simulate it on negative values with particular attention to the zero-valued argument. The family $\underline{\underline{\mathsf{P}}}$ needs only to contain the following single element:

$$\mathtt{S}(\mathtt{S}(x)) \times \mathtt{isO?}(x) + \mathtt{isE?}(x) \times (\mathtt{P}(\mathtt{P}(x)) \times \mathtt{Not}(\mathtt{isZ?}(x)) + \mathtt{S}(x) \times \mathtt{isZ?}(x))) \ .$$

It increments twice every odd argument, which comes from a negative value. In case of an even x, it distinguishes whether x is either greater than or equal to 0. In the last case it increases x once, so that the final value, through $\alpha^{-1}(x)$, becomes -1.

The translation of S does non need to discriminate a zero-valued argument, among the even ones. Its unique function is:

$$\mathtt{P}(\mathtt{P}(x)) \times \mathtt{isO?}(x) + \mathtt{S}(\mathtt{S}(x)) \times \mathtt{isE?}(x) \ .$$

Finally, the definition of fP_ℓ, with arity k and $\ell \equiv i_0, \ldots, i_{k-1}$, necessarily requires a family $\underline{\underline{\mathsf{fP}_\ell}}$ with k elements:

$$\underline{\underline{\mathsf{fP}_\ell}} = \{f_{i_j}(x_0, \ldots, x_{k-1}) = \pi^k_{i_j}(x_0, \ldots, x_{k-1})\}_{i_j \in \ell} \ .$$

Both $\underline{\underline{\circ[f; g_1, \ldots, g_j]}}$ and $\underline{\underline{\mathsf{Rec}[f, g, h]}}$ of course are a bit more involved. We leave them to an extended version of this work.

4 Conclusions

We conjecture that the proof of Theorem 1.1 can be generalized to prove that each language of functions being sufficiently expressive to code and decode TM-configurations either is complete or it lacks to contain one of its inverse (i.e. it is not closed by inversion).

It is an open issue if pairing and unpairing functions included in Definition 2.1 are independent from the remaining functions.

We plan to explore the above questions in order to build an extension of our language providing a fully blown characterization of reversible functions.

References

[1] 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.

[2] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17:525–532, 1973.

[3] N. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.

[4] 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.

[5] G. Jacopini and P. Mentrasti. Generation of invertible functions. *Theor. Comput. Sci.*, 66(3):289–297, 1989.

[6] 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.

[7] S. Kleene. *Introduction to Metamathematics.* Bibliotheca Mathematica. Wolters-Noordhoff, 1952.

[8] A. V. Kuznecov. On primitive recursive functions of large oscillation. *Doklady Akademii Nauk SSSR*, 71:233–236, 1950. In russian.

[9] R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, 1961.

[10] Y. Lecerf. Machines de turing réversibles. *Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences*, 257:2597–2600, 1963.

[11] A. B. Matos. Linear programs in a simple reversible language. *Theor. Comput. Sci.*, 290(3):2063–2074, 2003.

[12] 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.

[13] 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.

[14] K. S. Perumalla. *Introduction to Reversible Computing.* Chapman & Hall/CRC Computational Science. Taylor & Francis, 2013.

[15] R. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer, 1987.

[16] L. Szilard. Über die entropieverminderung in einem thermodynamischen system bei eingriffen intelligenter wesen. *Zeitschrift für Physik*, 53(11-12):840–856, 1929.

[17] 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.

[18] 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.

[19] H. Zenil. Information theory and computational thermodynamics: Lessons for biology from physics. *Information*, 3(4):739–750, 2012.