Persistent Computations of Turing Machines

Harald Hempel[1] and Madlen Kimmritz[2,⋆]

[1] Institut für Informatik
Friedrich-Schiller-Universität Jena
07743 Jena, Germany
hempel@informatik.uni-jena.de
[2] Mathematisches Seminar
Christian-Albrechts-Universität zu Kiel
24098 Kiel, Germany
mkimmritz@yahoo.de

Abstract. In this paper we formally define the notion of persistent Turing machines to model interactive computations. We compare the power of persistent computations with their classical counterparts.

1 Introduction

In many real world applications of algorithms it is often not the case, that computations start with an input and empty memory to produce some output. In human guided search [KL02] human knowledge and computer power interact to achieve better results than humans or computers alone could provide. In a 3-brain approach [A85] two computer programs provide alternative intermediate results with the human having the final say which one to use in the still ongoing computation. These two approaches serve merely as successful examples [KL02, LM03, LM03a, AS03] to illustrate that human-machine-interaction is used in practice. Classical (Turing machine) based models of computation can hardly describe these. A promising approach to do so is the notion of persistent Turing machines [GW98, Ko98]. A persistent Turing machine, or persistent computation (of a Turing machine), receives a sequence of inputs and produces a sequence of outputs while the computation on a single input may depend on all computations on previous inputs. In this paper we give a sound formal definition of the notion of persistent computations that until now was missing in the literature and so will lay the ground for methodically studying the power of persistent computations. We will also show how persistent computations relate to classical Turing machine computations. Since a persistent computation is based on an underlying Turing machine and a sequence of inputs (or an input function) we will look at various restrictions of input functions as well as Turing machines and compare the resulting types of persistent computations with their classical counterparts.

It has been shown that any function can be computed by a persistent computation [Ko98]. We will show that when restricting to computable input functions

⋆ Work done in part while working at Friedrich-Schiller-Universität Jena.

a persistent computation can only produce computable output functions. The power of persistent computations however is illustrated by the fact that a single input function is sufficient to compute all total recursive functions and primitive recursive input funtions suffice to compute all recursive functions (see Theorems 3.1 and 3.2). It also turns out that persistent computations give rise to new classes of functions that have no obvious classical counterparts (see Theorems 3.6 and 3.7).

The paper is organized as follows. In Section 2 we will formally introduce and define the notion of persistent computations and what it means to compute a function with a persistent computation. Section 3 contains some of our results and their proofs. We mention in passing that the authors have also obtained results concerning the notions of persistent-decidability as well as persistent-enumerability [HK].

2 Basic Concepts and Definitions

In this section we will develop our model of persistent computations.

Let $\Sigma = \{0, 1\}$ be our alphabet. We assume the reader to be familiar with the basic concepts and notations of recursion theory [Ro67]. Depending on the context we will view functions as to map from $\mathbb{N}$ to $\mathbb{N}$, from $\mathbb{N}$ to Σ^* or as to map from Σ^* to Σ^*. Due to the tight connection (there is a polynomial-time computable and invertible bijection) between $\mathbb{N}$ and Σ^* this will not weaken our results but simplify their presentation. Note that we will solely consider functions of that type. Let id be a bijective function and maps from $\mathbb{N}$ to Σ^* such that $id(0) = \varepsilon$, $id(1) = 0$, $id(2) = 1$, $id(3) = 00$ and so on. So, id^{-1} exists and, in particular, id and id^{-1} are primitive recursive. Let nd be the everywhere undefined function, i.e. $nd(n)$ is undefined for all $n \in \mathbb{N}$, which is partial recursive as well. For a function f let D_f and R_f denote the domain of definition and the range of f, respectively. The sets of partial and total recursive functions will be denoted by $\mathbb{P}$ and $\mathbb{R}$, respectively. We will denote $\mathbb{P}\mathrm{r}$ as the set of all primitive recursive functions. For any set of functions $\mathcal{F}$, let $\mathcal{F}_{\mathrm{total}}$ and $\mathcal{F}_{\mathrm{bij}}$ denote the sets of all total and bijective functions from $\mathcal{F}$, respectively.

2.1 The Model

In the formal Turing machine model each computation of a Turing machine starts with an input tape containing the input and worktapes that are empty. To model interactive and persistent behavior, as it is to be found in many todays real world scenarios, as already mentioned in the introduction, we introduce the notion of persistent computations of Turing machines. Even though this model has been studied in the literature before [GW98, Ko98] a clear definition has been missing until now.

Before we give formal definitions we recall the intuitive description. In contrast to classical Turing machine computations their persistent counterparts do not start out from an empty worktape. More precisely, the contents of the worktape at the start of the computation on an input x is identical to the content of the worktape at the end of the computation on the input preceeding x. So in fact, we view our machines as to receive sequences of inputs where the starting configuration for each input depends on the end configuration of the machine on previous inputs. That is what gave the model the name "persistent" computations. We now turn to formally define our model.

Definition 2.1. *Let M be a Turing machine with one input tape, one worktape and one output tape, input alphabet Σ and worktape alphabet Γ. Without loss of generality assume $\Sigma \subseteq \Gamma$. Let $f : \mathbb{N} \to \Sigma^*$ be a total function.*

1. *The functions $work_M : \Sigma^* \times \Gamma^* \to \Gamma^*$ and $out_M : \Sigma^* \times \Gamma^* \to \Gamma^*$ are defined as follows:*
   For all $x \in \Sigma^*$ and all $y \in \Gamma^*$, $work_M(x, y)$ and $out_M(x, y)$ are the contents of the worktape and output tape, respectively, of M in the end configuration if the computation of M on input x and initial worktape content y halts. Otherwise $work_M(x, y)$ and $out_M(x, y)$ are undefined.
   The computation $comp_M(x, y)$ of M on input x with initial worktape content y is the sequence of configurations that M on input x with initial worktape content y passes through, if that computation reaches a halting state. Otherwise $comp_M(x, y)$ is undefined.
2. *The mapping $hist_{M,f} : \mathbb{N} \to \Gamma^*$ is recursively defined as:*

$$hist_{M,f}(0) = \varepsilon \text{ and for all } i \in \mathbb{N}^+$$

$$hist_{M,f}(i+1) = \begin{cases} work_M(f(i), hist_{M,f}(i)), & \text{if } hist_{M,f}(i) \neq n.d., \\ n.d., & \text{if } hist_{M,f}(i) = n.d.. \end{cases}$$

3. *The output function $g_{M,f} : \mathbb{N} \to \Gamma^*$ is, for all $i \in \mathbb{N}$, defined via*

$$g_{M,f}(i) = out_{M,f}(f(i), hist_{M,f}(i)).$$

Thus a persistent computation receives a sequence of inputs, modelled as the function f in the above definition, and produces a sequence of outputs, named $g_{M,f}$ above. Persistence, the survival of information from previous computations, is modelled by the function $hist_{M,f}$. In other words, a persistent computation is a sequence of computations of a classical Turing machine M on inputs $f(0)$, $f(1)$, ... where the contents of the worktape from the previous computation is the initial content of the worktape for the current computation.

The function f in the above definition will be called input function and we denote the set of all total functions mapping from $\mathbb{N}$ to Σ^* by $\mathbb{I}\text{n}$.

Such persistent computations can produce an infinite or finite sequence of output values depending on whether all "local" computations of M halt or not (see also Figure 1).

In analogy to classical (Turing machine) computations we define the concept of a *persistent computation* via the notion of a configuration in the obvious way.

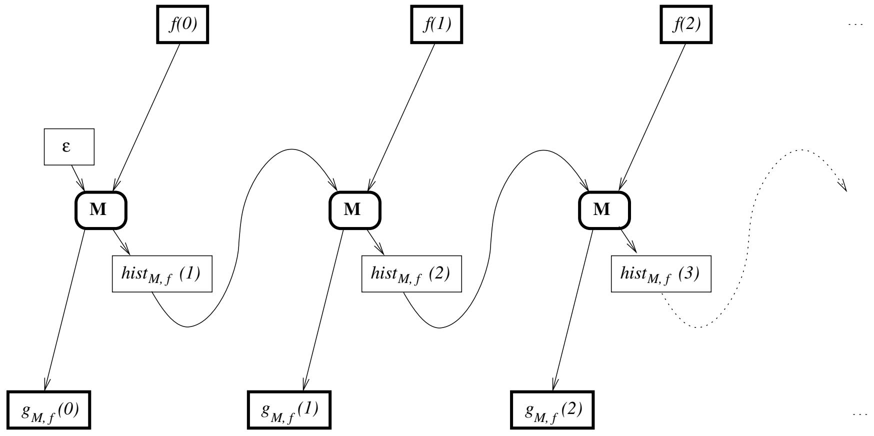

Fig. 1. Schematic illustration of a persistent computation of a Turing machine M on input sequence $f(0)$, $f(1)$, $f(2)$, $\ldots$ if all values of $hist_{M,f}$ are defined

Definition 2.2. *Let M be a Turing machine with one input tape, one worktape and one output tape, input alphabet Σ and worktape alphabet Γ. Without loss of generality assume $\Sigma \subseteq \Gamma$. Let $f \in \mathbb{I}\mathrm{n}$. The persistent computation of M on input f, i.e., on the input sequence $f(0)$, $f(1)$, $f(2)$, $\ldots$ is denoted by $M(f)$ and defined as*

$$M(f) =_{df} (comp_M(f(i), hist_{M,f}(i)))_{i\in\mathbb{N}},$$

if $comp_M(f(i), hist_{M,f}(i))$ is defined for all $i \in \mathbb{N}$. If $comp_M(f(i), hist_{M,f}(i))$ is not defined for all $i \in \mathbb{N}$ we define

$$M(f) =_{df} (comp_M(f(0), hist_{M,f}(0)), \ldots, comp_M(f(i_0), hist_{M,f}(i_0)))$$

where i_0 is the smallest $i \in \mathbb{N}$ such that $comp_M(f(i), hist_{M,f}(i))$ is undefined.

Note that all definitions of this section can easily be modified to deal with Turing machines having more than one worktape. Also, the general concept of persistence can be generalized to other models of computation, such as RAM-programs, MARKOV-algorithms and others, where all inner program variables persist between the end of a computation and the start of a new computation.

2.2 Persistent Computations of Functions

Note that the mapping from $f(i)$ to $g_{M,f}(i)$ in Definition 2.2 is in general not a function, since we might have $f(i) = f(j)$ and yet $g_{M,f}(i) \neq g_{M,f}(j)$ for some $i \neq j$. This observation leads to the following definition.

Definition 2.3 ([Ko98]). *Let M be a Turing machine and $f \in \mathbb{I}\mathrm{n}$.*

1. *The persistent computation $M(f)$ is called consistent if and only if for all $i, j \in \mathbb{N}$ the following condition holds for all $i, j \in \mathbb{N}$:*

$$f(i) = f(j) \rightarrow g_{M,f}(i) = g_{M,f}(j).$$

2. *A function $h : \Sigma^* \to \Sigma^*$ is said to be persistent computable (p-computable) if and only if there exists a Turing machine M and a function $f \in \mathbb{In}$ such that*
   (a) $M(f)$ is consistent,
   (b) $D_h \subseteq R_f$ and
   (c) for all $i \in \mathbb{N}$, $g_{f,M}(i) = h(f(i))$.
   In that case we say that M on input f p-computes the function h.
3. *A function $h : \mathbb{N} \to \mathbb{N}$ is said to be p-computable if and only if the mapping $\widehat{h} : \Sigma^* \to \Sigma^*$ being defined via $h(n) = id^{-1}(\widehat{h}(id(n)))$ for all $n \in \mathbb{N}$ is p-computable.*
4. *The set of all p-computable functions $h : \mathbb{N} \to \mathbb{N}$ is denoted by* $\mathrm{p}\mathbb{F}$.

Note that the requirement $D_h \subseteq R_f$ (instead of $D_h = R_f$) in Part 2b of Definition 2.3 gives more flexibility when it comes to p-computing partial functions. In particular there are two ways to realize $h(x) = n.d.$ for some x during a p-computation $M(f)$ of h, either x never shows up as an input, $x \notin R_f$, or even though $x \in R_f$, some prior runs of M do not terminate during the p-computation.

An interesting result concerning the persistent computation of functions was first observed in [Ko98].

Theorem 2.4 (Kosub). *[Ko98] Every function $h : \mathbb{N} \to \mathbb{N}$ is p-computable.*

The idea of the proof is to construct a (piecewise constant) input function f, such that the length of a constant part of the input function encodes the function value of the function h to be computed, on the input that will be provided by f when it changes its value the next time. The underlying Turing machine simply counts, how often the input value does not change and outputs that number when the input value changes. Note that since there are no restrictions on the input functions for persistent computations the encoding of otherwise uncomputable functions into the input leads to the above statement.

So in order to be fair, the power of p-computability should be studied when all input functions are required to be computable. In the process we will not only study restrictions to the input functions but also restrictions to the underlying machine model.

Definition 2.5. *Let $\Phi \subseteq \mathbb{In}$, and $\mathfrak{M}$ be a collection of programs or machines.*

1. *The set of p-Φ-computable functions is defined as*

$$\mathrm{p}\mathbb{F}(\Phi) =_{df} \{h \in \mathbb{F} \mid (\exists \mathrm{TM}\ M)(\exists f \in \Phi)[M \textit{ on input } f \textit{ p-computes } h]\}.$$

2. *The set of p-$(\Phi, \mathfrak{M})$-computable functions is defined as*

$$\mathrm{p}\mathbb{F}(\Phi, \mathfrak{M}) =_{df} \{h \in \mathbb{F} \mid (\exists \mathrm{TM}\ M \in \mathfrak{M})(\exists f \in \Phi)[M \textit{ on input } f \textit{ p-computes } h]\}.$$

Note that the Turing machine used in the above proof sketch of Theorem 2.4 does not do much more then counting. So any set Ψ of machines which are flexible enough to count will be able to p-$(\mathbb{In}, \Psi)$-compute all functions. Hence, the focus in the upcoming section will be on restricting the set of input functions.

3 The Power of Persistent Computations

As already mentioned above the true power of persistent computations can only be judged if the set of input functions is restricted to computable functions. It turns out that with this restriction, persistent computations can only compute recursive functions and, even stronger, primitive recursive input functions suffice to p-compute all recursive functions.

Theorem 3.1. $\mathbb{P} = \mathrm{p}\mathbb{F}(\mathbb{R}) = \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r})$.

Proof. Since we clearly have $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}) \subseteq \mathrm{p}\mathbb{F}(\mathbb{R})$ it suffices to prove $\mathrm{p}\mathbb{F}(\mathbb{R}) \subseteq \mathbb{P}$ and $\mathbb{P} \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r})$.

We first show $\mathrm{p}\mathbb{F}(\mathbb{R}) \subseteq \mathbb{P}$. Let $h \in \mathrm{p}\mathbb{F}(\mathbb{R})$. Hence, there exist a Turing machine M and an input function $f \in \mathbb{R}$ such that $M(f)$ p-computes h. In particular, for every $n \in D_h$ there is an $i \in \mathbb{N}$ such that $id(n) = f(i)$ and $h(n) = id^{-1}(g_{M,f}(i))$. It is not hard to see that $g_{M,f}$ is a recursive function, since f itself is total and recursive and in order to compute $g_{M,f}(i)$ for some $i \in \mathbb{N}$ we simply have to run the machine M $i+1$ times on the inputs $f(0)$, $f(1)$,..., $f(i)$ (in that order) while preserving the content of the worktapes between consecutive runs of M. More formally, the following recursive scheme clearly holds:

$$\begin{aligned}
hist_{M,f}(0) &= \epsilon \\
g_{M,f}(0) &= out_M(f(0), hist_{M,f}(0)) \\
hist_{M,f}(i+1) &= work_M(f(i), hist_{M,f}(i)) \text{ for all } i \geq 0 \\
g_{M,f}(i+1) &= out_M(f(i), hist_{M,f}(i)), \text{ for all } i \geq 0.
\end{aligned}$$

Since out_M, $work_M$, and f are recursive functions and $hist_{M,f}$ and $g_{M,f}$ are defined via a simultaneous recursion, based on out_M, $work_M$, and f, we conclude that also $hist_{M,f}$ and $g_{M,f}$ are recursive. Furthermore, for all $n \in \mathbb{N}$ we have $h(n) = id^{-1}(g_{M,f}(\min\{i \in \mathbb{N} : f(i) = id(n)\}))$. Since id, id^{-1}, f and $g_{M,f}$ are recursive functions and the class of recursive functions is closed with respect to the μ-operator it follows that h is recursive.

The inclusion $\mathbb{P} \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r})$ can be shown as follows. Let $h \in \mathbb{P}$. It is well known that for all $g_1 \in \mathbb{P}$ there exists a function $g_2 \in \mathbb{P}\mathrm{r}$ with $D_{g_1} = R_{g_2}$. So let $f \in \mathbb{P}\mathrm{r}$ be a function such that $D_h = R_f$. Let M be the Turing machine that computes the function h such that in each halting configuration the worktape is empty. Then, M on input f clearly p-computes h and thus $h \in \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r})$. □

Further restricting the input functions to total and surjective recursive functions ($\mathbb{R}_{\mathrm{surj}}$) or to the single function id (recall that $id : \mathbb{N} \to \Sigma^*$ is bijective), we obtain the following results.

Theorem 3.2. *1.* $\mathrm{p}\mathbb{F}(\mathbb{R}_{\mathrm{surj}}) = \mathbb{R} \cup \{f \in \mathbb{P} \mid |D_f| < \infty\}$.
2. $\mathrm{p}\mathbb{F}(\{id\}) = \mathbb{R} \cup \{f \in \mathbb{P} \mid (\exists n \in \mathbb{N})[D_f = \{0, 1, 2, \ldots, n\}]\}$.
3. $\mathrm{p}\mathbb{F}_{\mathrm{total}}(\{id\}) = \mathbb{R}$.

The proof is omitted due to space restrictions. Note that the classes $\mathbb{R}$, $\mathbb{R} \cup \{f \in \mathbb{P} \mid (\exists n \in \mathbb{N})[D_f = \{0, 1, 2, \ldots, n\}]\}$, $\mathbb{R} \cup \{f \in \mathbb{P} \mid |D_f| < \infty\}$, and $\mathbb{P}$ form a chain of strict inclusions and thus we have an inclusion structure as shown in Figure 2.

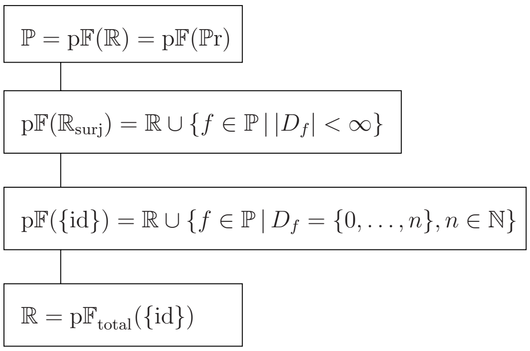
$\mathbb{P} = \mathrm{p}\mathbb{F}(\mathbb{R}) = \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r})$
$\mathrm{p}\mathbb{F}(\mathbb{R}_{\mathrm{surj}}) = \mathbb{R} \cup \{f \in \mathbb{P} \mid |D_f| < \infty\}$
$\mathrm{p}\mathbb{F}(\{\mathrm{id}\}) = \mathbb{R} \cup \{f \in \mathbb{P} \mid D_f = \{0, \ldots, n\}, n \in \mathbb{N}\}$
$\mathbb{R} = \mathrm{p}\mathbb{F}_{\mathrm{total}}(\{\mathrm{id}\})$

Fig. 2. Classes between $\mathbb{R}$ and $\mathbb{P}$

We will now turn to characterize the function class $\mathbb{P}\mathrm{r}$ in terms of persistent computations. Since we have $\mathrm{p}\mathbb{F}_{total}(\{id\}) = \mathbb{R}$, a restriction of the set of input functions alone will not be sufficient to reduce the power of persistent computations to characterize $\mathbb{P}\mathrm{r}$. We additionally will have to restrict the underlying Turing machine model. Let $\mathcal{TM}_{\mathbb{P}\mathrm{r}}$ denote the set of all Turing machines that compute functions that are primitive recursive and that have a primitive recursive *work* funtions.

Theorem 3.3. $\mathbb{P}\mathrm{r} = \mathrm{p}\mathbb{F}(\{id\}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$.

Proof. The inclusion $\mathbb{P}\mathrm{r} \subseteq \mathrm{p}\mathbb{F}(\{id\}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ is obvious. So it is sufficient to prove $\mathrm{p}\mathbb{F}(\{id\}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathbb{P}\mathrm{r}$.

Let $h \in \mathrm{p}\mathbb{F}(\{id\}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ be a function and let M be a Turing machine from $\mathcal{TM}_{\mathbb{P}\mathrm{r}}$ such that M on input id p-computes h. Applying an argument similar to the one in the proof of $\mathrm{p}\mathbb{F}(\mathbb{R}) \subseteq \mathbb{P}$ (see Theorem 3.1) while obeying that all involved functions are primitive recursive and $\mathbb{P}\mathrm{r}$ is closed under simultaneous recursion as well as the bounded μ-operator it is not hard to see that h is primitive recursive. $\square$

Next we will show that we can even allow more input functions than just id and still get a characterization of $\mathbb{P}\mathrm{r}$. Let $\mathbb{P}\mathrm{r}_{-1}$ denote the set of all functions $f \in \mathbb{P}\mathrm{r}$ such that the inverse f^{-1} is a function and also in $\mathbb{P}\mathrm{r}$, i.e. $\mathbb{P}\mathrm{r}_{-1} = \{f \in \mathbb{P}\mathrm{r} \mid f^{-1} \in \mathbb{P}\mathrm{r}\}$.

Theorem 3.4. $\mathbb{P}\mathrm{r} = \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$.

Proof. Since clearly $id, id^{-1} \in \mathbb{P}\mathrm{r}$ we have $\mathrm{p}\mathbb{F}(\{id\}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ and thus $\mathbb{P}\mathrm{r} \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ by Theorem 3.3. So it remains to show that $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathbb{P}\mathrm{r}$.

So let $h \in \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. Hence, there exist a function $f \in \mathbb{P}\mathrm{r}_{-1}$ and a Turing machine $M \in \mathcal{TM}_{\mathbb{P}\mathrm{r}}$ such that M on input f p-computes h. Without loss of generality let $work_M$ be primitive recursive. Similar to the argument in the proof sketch of Theorem 3.3 one can show that $g_{M,f}$ is primitive recursive since it can be described via simultaneous recursion and the functions out_M, $work_M$, and f. Since for all $n \in \mathbb{N}$, $h(n) = g_{M,f}(\min\{i \in \mathbb{N} : i = f^{-1}(n)\})$, using the fact that f^{-1} is primitive recursive and $\mathbb{P}\mathrm{r}$ is closed with respect to the application of the bounded μ-operator we obtain $h \in \mathbb{P}\mathrm{r}$. □

One might be tempted to conjecture that even $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) = \mathbb{P}\mathrm{r}$ holds. This is not the case, as we will show in the following. Recall that $\mathbb{P}\mathrm{r}_{\mathrm{bij}}$ denotes the set of all functions in $\mathbb{P}\mathrm{r}$ that are bijective. Clearly, $\mathbb{P}\mathrm{r}_{-1} \subseteq \mathbb{P}\mathrm{r}_{\mathrm{bij}}$. We will now argue that $\mathbb{P}\mathrm{r}_{-1} \subset \mathbb{P}\mathrm{r}_{\mathrm{bij}}$.

Lemma 1. $\mathbb{P}\mathrm{r}_{\mathrm{bij}}$ *is not closed with respect to inversion.*

Proof. By a result of Robinson ([Ro50]) we know that

$$\mathbb{R} = \Gamma_{\mathrm{ADD,SUB,INV}}(\{succ,\ x \dot{-} \lfloor\sqrt{x}\rfloor^2\})$$

where *succ* denotes the successor function and for all sets of functions A, the term $\Gamma_{\mathrm{ADD,SUB,INV}}(A)$ denotes the closure of A with respect to addition (ADD), subtraction (SUB) and a limited form of inversion (INV) where inversion can only be applied to bijective functions.

Recall that $\{succ,\ x \dot{-} \lfloor\sqrt{x}\rfloor^2\}) \subseteq \mathbb{P}\mathrm{r}$, $\mathbb{P}\mathrm{r}$ is closed with respect to ADD and SUB, and $\mathbb{R} \setminus \mathbb{P}\mathrm{r} \neq \emptyset$. Let $h \in \mathbb{R} \setminus \mathbb{P}\mathrm{r}$. The function h can be described by a finite sequence of successive applications of ADD, SUB, and INV on either *succ* or $x \dot{-} \lfloor\sqrt{x}\rfloor^2$. Since during the process of these successive applications of ADD, SUB, and INV, the functions we start from are in $\mathbb{P}\mathrm{r}$ and the function we end with is element of $\mathbb{R} \setminus \mathbb{P}\mathrm{r}$, there exist (intermediate) functions $f \in \mathbb{P}\mathrm{r}$ and $g \in \mathbb{R} \setminus \mathbb{P}\mathrm{r}$ such that:

1. $f \in \Gamma_{\mathrm{ADD,SUB,INV}}(\{succ,\ x \dot{-} \lfloor\sqrt{x}\rfloor^2\})$,
2. $f \in \mathbb{P}\mathrm{r}_{\mathrm{bij}}$,
3. $g \in \Gamma_{\mathrm{ADD,SUB,INV}}(\{succ,\ x \dot{-} \lfloor\sqrt{x}\rfloor^2\})$,
4. $g = f^{-1}$.

Since $\mathbb{P}\mathrm{r}_{\mathrm{bij}} \subseteq \mathbb{P}\mathrm{r}$ and $g \in \Gamma_{\mathrm{INV}}(\mathbb{P}\mathrm{r}_{\mathrm{bij}})$ we obtain $\mathit{\Gamma}_{\mathrm{INV}}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}) \neq \mathbb{P}\mathrm{r}_{\mathrm{bij}}$ and hence $\mathbb{P}\mathrm{r}_{\mathrm{bij}}$ is not closed under inversion. □

Since $\mathbb{P}\mathrm{r}_{-1}$ is clearly closed under inversion we have the following corollary.

Corollary 3.5. $\mathbb{P}\mathrm{r}_{-1} \subset \mathbb{P}\mathrm{r}_{\mathrm{bij}}$.

Moreover we can show, that the class $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ is located between $\mathbb{P}\mathrm{r}$ and $\mathbb{R}$.

Theorem 3.6. $\mathbb{P}\mathrm{r} \subset \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathbb{R}$.

Proof. We will first show $\mathbb{P}\mathrm{r} \subset \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$.

Since $\mathbb{P}\mathrm{r} = \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{-1}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ due to Theorem 3.4 and $\mathbb{P}\mathrm{r}_{-1} \subseteq \mathbb{P}\mathrm{r}_{\mathrm{bij}}$ we have $\mathbb{P}\mathrm{r} \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. It remains to show that there exists a function $h \in \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ that is not primitive recursive.

Let f be a function in $\mathbb{P}\mathrm{r}_{\mathrm{bij}} \setminus \mathbb{P}\mathrm{r}_{-1}$ and M be a Turing machine from $\mathcal{TM}_{\mathbb{P}\mathrm{r}}$ such that $g_{M,f'} = id$ for any input function f'. Clearly $M(f)$ is a consistent p-computation since f is bijective. Let h denote the function computed by the p-computation $M(f)$. Hence $h \in \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. However, we have $h(f(i)) = g_{M,f}(i) = i$ for all $i \in \mathbb{N}$ and thus $h = f^{-1}$. It follows that h is not primitive recursive.

We now proof $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathbb{R}$. The inclusion $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathbb{R}$ can be seen as follows. On the one hand it follows immediately from Theorem 3.1 that $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathbb{P}$ and on the other hand a p-computation $M(f)$ for a Turing machine M from $\mathcal{TM}_{\mathbb{P}\mathrm{r}}$ and a function $f \in \mathbb{P}\mathrm{r}_{\mathrm{bij}}$ always yields a total function.

To show the strictness of that inclusion we use the Ackermann function p (also known as Peter function) [Ro67]. It is known that $p \in \mathbb{R} \setminus \mathbb{P}\mathrm{r}$. Let the function q be defined as $q(n) = p(n,n)$ for all $n \in \mathbb{N}$. Assume that q is an element of $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. Hence there exist a function $f \in \mathbb{P}\mathrm{r}_{\mathrm{bij}}$ and a Turing machine $M \in \mathcal{TM}_{\mathbb{P}\mathrm{r}}$ such that $M(f)$ is a p-computation of q. Without loss of generality let $work_M$ be primitive recursive. It follows that $g_{M,f}$ is primitive recursive as well since it can be described via a simultaneous recursion based on out_M and $work_M$ (see the proof of Theorem 3.1). Since $(q \circ f)(n) = q(f(n)) = g_{M,f}(n)$ for all $n \in \mathbb{N}$ we have $q \circ f \in \mathbb{P}\mathrm{r}$. It is a well-known fact that the Peter function grows faster then any primitive recursive function, i.e.,

$$(\forall h \in \mathbb{P}\mathrm{r})(\exists m \in \mathbb{N})(\forall n \in \mathbb{N})[h(n) < p(m,n)].$$

Hence, there exists an $m \in \mathbb{N}$ such that for all $n \in \mathbb{N}$, $q(f(n)) < p(m,n)$. Since f is surjective there exists $n_0 \in \mathbb{N}$ such that $f(n_0) = m$. It follows that

$$q(f(n_0)) = q(m) < p(m,m)$$

which contradicts the definition $q(n) = p(n,n)$ for all $n \in \mathbb{N}$. Hence our assumption $q \in \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ was false. □

Next we will classify the set $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ which turns out to be not equal to $\mathbb{P}\mathrm{r}$ as the above Theorem 3.6 implies. On the one hand $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ is a strict superset of $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ since input functions from $\mathbb{P}\mathrm{r}$ are more flexible than input functions from $\mathbb{P}\mathrm{r}_{\mathrm{bij}}$. On the other hand $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ remains a subset of $\mathbb{P}$ as the following proposition shows.

Theorem 3.7. $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathbb{P} \setminus \{nd\}$.

Proof. The first inclusion, $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ is easy to see. Note that $\mathbb{P}\mathrm{r}_{\mathrm{bij}} \subset \mathbb{P}\mathrm{r}$ and hence $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$, yet the function $h : \mathbb{N} \to \mathbb{N}$ that, for all $n \in \mathbb{N}$, is defined as

$$h(n) = \begin{cases} n, & \text{if } n \equiv 0 \mod 2, \\ \text{n.d.}, & \text{if } n \equiv 1 \mod 2, \end{cases}$$

is an element of $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \setminus \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$.

To show the other inclusion, $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subset \mathbb{P} \setminus \{nd\}$, first observe that the function nd is not in $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$ since $g_{M,f}(0)$ is defined for any $M \in \mathcal{TM}_{\mathbb{P}\mathrm{r}}$ and any input function f. Second, recall that we have $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}) = \mathbb{P}$ by Theorem 3.1 and thus $\mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}}) \subseteq \mathbb{P} \setminus \{nd\}$. And third, note that the strictness of that inclusion follows from the proof of Theorem 3.6. In particular, in that proof a function q was defined and it was shown that q is in $\mathbb{R} \setminus \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}_{\mathrm{bij}}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. Using the same argument one can also show that $q \in \mathbb{P} \setminus \mathrm{p}\mathbb{F}(\mathbb{P}\mathrm{r}, \mathcal{TM}_{\mathbb{P}\mathrm{r}})$. □

Acknowledgments. The authors would like to thank the anonymous referees for their very helpful comments.

References

[A85] Althöfer, I.: Das Dreihirn—Entscheidungsteilung im Schach. Computerschach & Spiele, 20–22 (December 1985)

[AS03] Althöfer, I., Snatzke, R.G.: Playing Games with Multiple Choice Systems. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 142–153. Springer, Heidelberg (2003)

[BF04] Brand, M., Frisken, S., Lesh, N., Marks, J., Nikovski, D., Perry, R., Yedidia, J.: Theory and Applied Computing: Observations and Anecdotes. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 106–118. Springer, Heidelberg (2004)

[GW98] Goldin, D., Wegner, P.: Persistence as a form of interaction, Technical Report CS-98-07, Brown University, Department of Computer Science (1998)

[HK] Hempel, H., Kimmritz, M.: Aspects of Persistent Computations (unpublished)

[Ko98] Kosub, S.: Persistent Computations, Technical Report No. 217, Julius-Maximilians-Universität Würzburg, Institut für Informatik (1998)

[KL02] Klau, G., Lesh, N., Marks, J., Mitzenmacher, M.: Human-guided tabu search. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence, Fourteenth Conference on Innovative Applications of Artificial Intelligence 2002, pp. 41–47. AAAI Press, Menlo Park (2002)

[LM03] Lesh, N., Mitzenmacher, M., Whitesides, S.: A complete and effective move set for simplified protein folding. In: Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology, pp. 188–195 (2003)

[LM03a] Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M.: New exhaustive, heuristic, and interactive approaches to 2D rectengular strip packing, TR2003-05. Mitsubishi Electric Research Laboratories, Cambridge (May 2003)

[Ro50] Robinson, R.: General recursive functions. Proceedings of the American Mathematical Society 1, JO 3-718 (1950)

[Ro67] Rogers, H.: The Theory of Recursive Functions and Effective Computability, 2nd edn. (1987). MIT Press, Cambridge (1967)

[Sch92] Schöning, U.: Theoretische Informatik kurz gefasst, Mannheim; Leipzig; Wien; Zürich: BI-Wissenschaftsverlag (1992)