Persistent Computations

Sven Kosub
Theoretische Informatik, Universität Würzburg
Am Exerzierplatz 3, D-97072 Würzburg
Email: *kosub@informatik.uni-wuerzburg.de*

December 22, 1998

Abstract

We study computational effects of persistent Turing machines, independently introduced by Goldin and Wegner [GW98], and Kosub [Kos98]. Persistence is a mode of interaction which makes it possible to consider the computational behavior of a Turing machine as an infinite sequence of autonomous computations. We investigate different computability concepts such as conditional and essential computability. Furthermore, we give a characterization of all non-immune sets in terms of persistent computations.

1 Introduction

Interaction extends the algorithmic view on computation. By the facility of computational devices to communicate with the world outside, in particular to receive informations from there, moments of non-controlability and of non-predictability come inherent in computations. The more powerful interaction can be, the less dominating is the algorithm for the behavior the device is actually showing. Not least for that reason, a paradigm shift from algorithmic to interactive view on computing is proposed (see e.g. [Weg98, Mil93]).

In theory of computation, modes of interaction were considered as long as the formal notion of an algorithm was made precise. Such modes include well-known concepts like oracles, advice functions, or interactive proof systems. All in common is that a computation is only considered until the output becomes fixed; from the possibility of succeeding computations it is refrained. Taking an observer's point of view on the computational behavior, this seems to be a fundamental lack of interactive quality.

In this paper we study some computational effects of the simplest mode of interaction that fulfills such interactive quality—persistence. Persistence is the property of data, facts, or objects to survive computations. It plays an important role in database theory, artificial intelligence, and object-oriented programming. In terms of Turing machines persistence means that tape inscriptions kept written after the end of the computation, and can influence a succeeding computation if they are added to the new input. Implementing this idea into formally sound model it will suffice to adapt the semantics of a Turing machine. This is why we can speak about persistence as the simplest mode of interaction; we do not need to change the model, we only look at computations interactively. The notion of the persistent Turing machine examined here was independently defined by Goldin and Wegner [GW98], and under a different name by Kosub [Kos98]. All the results mentioned below are contained in the latter work.

Persistence is inherently chronologic. The behavior of a persistent machine at the current input—the local behavior—depends heavily on all its preceding inputs typically in exactly that order they are given to the machine. Hence, the global behavior of machines depends on infinite input sequences, so-called streams [Weg98].

In [GW98] it is proved that (the glass-box notion of) persistence and (the black-box notion of) sequential interaction are the same in the sense of distinguishability of system behavior no matter if the computational process shows "functional" behavior. The approach here concerns infinite sequences of natural numbers at which, not to leave the classical area of theory of computation, we understand each number occuring in the sequence as an argument of a function to be computed. Rigorously we take the function to give a connection between inputs and outputs only of those computations being actually realized along a sequence of inputs, not of those computations that are potentially possible. As a concrete example this implies that the domain of such a function contains no more than the elements of the sequence. A computed function hence depends on both the algorithm and the input sequence. Clearly, to speak about functions in a mathematically correct sense, some uniqueness requirements for computations are necessary in relation to same inputs.

One could argue the model of the persistent Turing machine is of relatively few interest since, if we always know whole pre-histories, then everything can be already computed by a standard Turing machine. This objection is maintainable, but it does not fully touch our model. Actually, even this argument is used by us to define a local behavior in the model, i.e. how a concrete machine behaves to a single computation. All in all, we take a global view on computations that is expressed in considering the sequence and hence a special order in pre-histories. This order is given by prolongation of the pre-histories according to the sequence.

In a strong sense each persistent machine deals with an infinite object. In non-classical areas of computing theory there are also computational models handling infinite objects (see e.g. [Rog67, SW78, CG78, CG80, KW85, Wei85, Ko91]). Most of them are extensions of standard Turing machines to infinite sequences over finite alphabets. Insofar, they lay near to persistent machines. Apart from difficulties in assigning computational processes to certain inputs, differences are mainly caused in understanding the objects dealt with. As an example, on the one hand sequences are considered as words which belong to sets grouped in language classes, and Turing machines acting on these sequence shall decide in some sense a language of such a class, i.e. a set of sequences. On the other side, using persistent Turing machines we are interested in the function produced by the machine under the condition of the input sequence. We ask for an individual object, a single function.

It turns out that every partial function is computable by a very simple persistent machine considering an appropriate input sequence. Persistent interaction is in some sense much more powerful than algorithmically computing. But if we only admit persistent machines that compute the same functions under all input sequence (notice the similiarity to Schöning's notion of robust machines [Sch85]), then exactly the (partially) recursive functions can be computed. Surprisingly, it is emphasized that this does not change if we demand the computational invariance to hold only for almost all sequences. Considering further weak invariance conditions we get also a characterization of all non-immune sets.

The paper is organized as follows. Section 2 lays mathematical background for our investigations. In Section 3 the concept of the persistent Turing machine is precisely defined. Further, we take into consideration two natural computability concepts defined over persistent machines: conditional and essential computability. In Section 4 we study some notions connected with conditional computations. Section 7 is devoted to probability aspects of conditional computations converging in essential computability. Intermediate sections contain some initiating results about connections between oracle machines and persistent machines (Section 5), and about degrees of independence between succeeding computations (Section 6). We finish with the conclusion in Section 8. Generally, the setting is recursion-theoretic.

2 Preliminaries and Notions

Basic set theory is supposed to be known. For an arbitrary set A, $\overline{A}$ denotes the complement of A, and $\mathfrak{P}A$ denotes the set of all subsets of A. By $\#A$ we denote the number of elements of the set A. A co-finite set A is a set with $\#\overline{A} < \infty$. The characteristic function χ_A of a set A is 1 for each element of A, and is 0 for each element of $\overline{A}$.

The finite alphabet $\Sigma = \{0,1\}$ is used. Σ^* is the set of all finite words over Σ with the empty word λ. For $w \in \Sigma^*$ the length of w is denoted by $|w|$. In Σ^* we assume the usual lexicographical order $<$, i.e. the lexicographical order with primary respect to the word length. Let $\mathbb{N} = \{0,1,2,\dots\}$ be the set of natural numbers. We identify $\mathbb{N}$ and Σ^* by the easily computable and easily invertible bijection $B : \Sigma^* \longrightarrow \mathbb{N}$ defined as $B(w) = \#\{\, v \in \Sigma^* \mid v < w \,\}$.

$\mathbb{N}^m$ is the set of m-dimensional vectors of natural numbers ($m \in \mathbb{N}, m \geq 1$), and $\mathbb{N}^*$ is the set of finite-dimensional vectors of natural numbers, i.e. $\mathbb{N}^*$ is the union of $\mathbb{N}^m$ for all $m \geq 1$. For any vector $\xi = (\xi_1, \dots, \xi_m) \in \mathbb{N}^*$ the dimension of ξ is defined as $\dim \xi = m$. On $\mathbb{N}^*$ a prefix relation is given: Say $\xi \sqsubseteq \eta$ if and only if $\dim \xi \leq \dim \eta$ and $\xi_j = \eta_j$ for every $1 \leq j \leq \dim \xi$. $\xi \sqsubset \eta$ if and only if $\xi \sqsubseteq \eta$ and $\xi \neq \eta$.

$\mathbb{N}^\infty$ denotes the set of all infinite sequences of natural numbers. For $\xi \in \mathbb{N}^\infty$ the set $\{\xi_1, \xi_2, \xi_3, \dots\}$ is denoted by R_ξ and is called *range of* ξ. A sequence with $R_\xi = \mathbb{N}$ is said to be a *full* sequence. The set of all full sequences is denoted by $\mathbb{N}^\infty_T$.

Only functions $f : \mathbb{N} \longrightarrow \mathbb{N}$ are of interest. $\mathcal{F}_P$ is the set of all properly partial functions including the function ν nowhere defined, and $\mathcal{F}_T$ is the set of all total functions. $\mathcal{F}$ is the union of $\mathcal{F}_P$ and $\mathcal{F}_T$. For any function $f \in \mathcal{F}_T$ and any set $A \subseteq \mathbb{N}$ the function $f|_A$ is set to be $f(x)$ if $x \in A$ or is set to be undefined (denoted by $-$) otherwise. For partial functions $f, g \in \mathcal{F}$ we say $f \simeq g$ if and only if the domains D_f of f and D_g of g are equal and for every $x \in D_f$ holds $f(x) = g(x)$. The range of f is denoted by R_f. For a set $A \subseteq \mathbb{N}$ we define $f(A) := \{\, f(x) \mid x \in A \,\}$ and $f^{-1}(A) := \{\, x \in \mathbb{N} \mid f(x) \in A \,\}$.

We adopt familiarity with general recursion-theoretic notions of effective computability as in [Rog67, Soa87]). Along this line, our basic model of computation is the (multitape) Turing machine (see e.g. [Rog67, Soa87]). REC is the class of recursively decidable (or recursive, for short) sets in $\mathbb{N}$, and RE is the class of recursively enumerable sets. If writing REC_+ and RE_+, then the classes excluding finite sets are considered. A *recursive* sequence ξ is a full sequence for which exists a total recursive function f such that $\xi_{j+1} = f(j)$ for all j.

Let $\langle \cdot, \cdot \rangle$ be any effectively computable and effectively invertible standard pairing function of a pair of natural numbers into a natural number. This is extended to an encoding of every finite tuple of natural numbers into a natural number by the usual inductive construction: (i) $\langle x \rangle := \langle 1, x \rangle_2$, and (ii) $\langle x_1, \dots, x_k \rangle := \big\langle k, \langle x_1, \dots, x_{k-1} \rangle \big\rangle_2$.

In the following we make use of probability-theoretic and measure-theoretic terms, so the reader is assumed to be familiar with basic concepts (see e.g. [Fel68, Shi95]).

We need some special probability fields or measurable spaces, respectively. The first is according to $\mathbb{N}^\infty$. Let d be any distribution function on $\mathbb{N}$. If d is recursive and satisfies $d(n) > 0$ for every $n \in \mathbb{N}$ then call such d *plausible*. Each d gives a probability measure on $\mathbb{N}$. Extending this to the m-dimensional case, we define a probability measure $\mathbf{P}_d^m$ on $\mathfrak{P}\mathbb{N}^m$ using the product measure construction. Explicitely

$$\mathbf{P}_d^m(A) = \prod_{j=1}^{m} \sum_{n \in A_j} d(n)$$

for every $A = A_1 \times \cdots \times A_m \in \mathfrak{P}\mathbb{N}^m$. The specification of the proper probability model we use follows standard argumentations.

Definition 2.1. Let d be any distribution function, let $m, t_1, \ldots, t_m \in \mathbb{N}$ with $t_1 < \cdots < t_m$, and let $A = A_1 \times \cdots \times A_m \in \mathfrak{P}\mathbb{N}^m$.

1. The set $C_{t_1,\ldots,t_m}(A)$ is called *cylinder in* $\mathbb{N}^\infty$ if and only if

$$C_{t_1,\ldots,t_m}(A) := \begin{cases} \{\, s \in \mathbb{N}^\infty \mid s_{t_j} \in A_j \text{ for } 1 \le j \le m \,\}, & \text{if } m \ge 1, \\ \mathbb{N}^\infty, & \text{if } m = 0. \end{cases}$$

2. $C(\mathbb{N}^\infty)$ is the set of all cylinders in $\mathbb{N}^\infty$.

3. The σ-algebra $\mathfrak{C}(\mathbb{N}^\infty) := \sigma\big(C(\mathbb{N}^\infty)\big)$ is called the *σ-algebra of cylinders in* $\mathbb{N}^\infty$.

4. On $C(\mathbb{N}^\infty)$ the measure $\hat{\mathbf{P}}_d$ is defined as $\hat{\mathbf{P}}_d(C_{t_1,\ldots,t_m}(A)) := \mathbf{P}_d^m(A)$.

The following results are well-known or easy to see.

Proposition 2.2. *Let d be any distribution function.*

1. *$C(\mathbb{N}^\infty)$ is an algebra.*

2. *Let $C_{s_1,\ldots,s_m}(A), C_{t_1,\ldots,t_n}(B) \in C(\mathbb{N}^\infty)$. Then if $\{s_1, \ldots, s_m\} \cap \{t_1, \ldots, t_n\} = \varnothing$ then $C_{s_1,\ldots,s_m}(A)$ and $C_{t_1,\ldots,t_n}(B)$ are independent.*

3. *There exists an unique probability measure $\mathbf{P}_d$ on $\mathfrak{C}(\mathbb{N}^\infty)$ which coincides with $\hat{\mathbf{P}}_d$ restricted to $C(\mathbb{N}^\infty)$.*

So, the triple $\big(\mathbb{N}^\infty, \mathfrak{C}(\mathbb{N}^\infty), \mathbf{P}_d\big)$ is our probability field according to $\mathbb{N}^\infty$ for any distribution function d.

In similar way we construct a measurable space on the set of number-theoretic functions $\mathcal{F}$. A little difference is in the definitions of the cylinders. Here cylinders in $\mathcal{F}$ are defined as

$$\hat{C}_{t_1,\ldots,t_m}(A) = \begin{cases} \{\, f \mid \{t_1, \ldots, t_m\} \subseteq D_f \text{ and } f(t_j) \in A_j \text{ for } 1 \le j \le m \,\}, & \text{if } m \ge 1, \\ \mathcal{F}, & \text{if } m = 0 \end{cases}$$

for $A = A_1 \times \cdots \times A_m \in \mathfrak{P}\mathbb{N}^m$. Note that $\hat{C}(\mathcal{F})$, the set of all cylinders in $\mathcal{F}$, is not an algebra, but is closed under intersection and (finite) union. With $\hat{\mathfrak{C}}(\mathcal{F})$ we denote the σ-algebra generated by $\hat{C}(\mathcal{F})$.

3 Persistent Machines

We specify our fundamental model of a persistent Turing machine. As mentioned in the introductory section, the definition affects only the semantic of standard Turing machines, not the syntax.

Definition 3.1. [GW98, Kos98] A *persistent* Turing machine K is a Turing machine with an one-way read-only input, an one-way write-only output tape, and with at least one additional work tape.

Initially, all tapes of a machine are cleared. The interaction now works as follows: The user writes an input on the input tape beginning with the field where the reading head is placed to the right. Subsequently the user starts a run. The machine scans the input from the input tape, and executes its algorithm with all inscriptions being contained at work tapes. Finally, the machine writes a word to the output tape beginning with a label. The output of such a computation is the word between the label and the current head position. Now, if the computation is finished, the user may write a new input on the input tape, and the process starts afresh. So, the results of computations formally depend on all input ever been written on the input tape. This is expressed in terms of a local output function.

Definition 3.2. Let K be a persistent Turing machine, and let $x_1, \ldots, x_{n+1} \in \mathbb{N}$. The *local output function out_K of* K is a mapping from $\mathbb{N} \times \mathbb{N}^*$ to $\mathbb{N}$ defined as

$$out_K\big(x_{n+1}|(x_1,\ldots,x_n)\big) = \begin{cases} z_1, & \text{if } n = 0 \text{ and } K \text{ outputs } z_1 \text{ on input } x_1 \text{ where all} \\ & \text{work tapes are cleared before the start,} \\ z_2, & \text{if } n > 0 \text{ and } K \text{ outputs } z_2 \text{ on input } x_{n+1} \text{ under the} \\ & \text{condition that } K \text{ has finished before the computa-} \\ & \text{tions according to the inputs } x_1,\ldots,x_n \text{ in exactly} \\ & \text{this order,} \\ -, & \text{if on input } x_j \text{ with } j \in \{1,\ldots,n+1\} \text{ the compu-} \\ & \text{tation does not terminate getting the inputs in this} \\ & \text{order.} \end{cases}$$

For $n = 0$ the definition of the local output function captures precisely the semantic of the standard Turing machine. The term *local* is justified since this output function determines only the behavior of a persistent machine at a $(n+1)$-dimensional point (sequence) in $\mathbb{N}^*$. One cannot speak about a number-theoretic function computed by K without knowing the order of such points. To obtain an output function which makes this possible it is necessary to assume that whenever along an infinite sequence of inputs the same input is given, the machine K writes the same output on the output tape. A machine satisfying this assumption is a consistent machine.

Definition 3.3. Let K be a persistent Turing machine.

1. Let $\xi \in \mathbb{N}^\infty$. K is said to be *consistent in* ξ if and only if for all $i, j \in \mathbb{N}$ holds that $\xi_{i+1} = \xi_{j+1}$ implies $out_K\big(\xi_{i+1}|(\xi_1,\ldots,\xi_i)\big) = out_K\big(\xi_{j+1}|(\xi_1,\ldots,\xi_j)\big)$.
2. K is said to be *consistent* if and only if K is consistent in ξ for every $\xi \in \mathbb{N}^\infty$.

Clearly, not every persistent machine is consistent, but this restriction to the machines is not very hard. Easily, each persistent machine can be modified in a way that the output computed by the machine for the first time the input occurs is written everytime for the same input.

Any consistent persistent Turing machine defines global behavior along a sequence.

Definition 3.4. Let K be a consistent persistent Turing machine, and let $\xi \in \mathbb{N}^\infty$. The *global output function $out_K(\cdot|\xi)$* is the function for every $x \in \mathbb{N}$ defined as

$$out_K(x|\xi) = \begin{cases} out_K\big(\xi_i|(\xi_1,\ldots,\xi_{i-1})\big), & \text{if an } i \text{ with } x = \xi_i \text{ exists,} \\ -, & \text{otherwise.} \end{cases}$$

Throughout the following we suppose a persistent machine to be consistent such that the global output function is always well-defined.

4 Conditional Computabilities

In this section we focus our attention to number-theoretic functions from $\mathcal{F}$ that are in some sense computable by persistent machines. By our approach a computed function does not only depend on the machine, but also on the sequence given to the machine. This is an additional condition for computing.

Definition 4.1. Let $f \in \mathcal{F}$, let K be a (consistent) persistent Turing machine, and let $\xi \in \mathbb{N}^\infty$.

1. *K computes f under the condition ξ* if and only if $R_\xi = D_f$ and $f(x) = out_K(x|\xi)$ for every $x \in R_\xi$. (K, ξ) is called a *conditional computation of f*.
2. *K computes f under the weak condition ξ* if and only if $R_\xi \subseteq D_f$ and $f(x) = out_K(x|\xi)$ for every $x \in R_\xi$. Then, (K, ξ) is called a *weakly conditional computation of f*.
3. *K computes f (weakly) conditionally* if and only if there is a sequence $\xi \in \mathbb{N}^\infty$ such that (K, ξ) is a (weakly) conditional computation of f.
4. The function f is *(weakly) conditionally computable* if and only if there is a (consistent) persistent Turing machine K computing f (weakly) conditionally.

It is intuitively clear that making rigorous use of persistent machines we can conditionally compute any number-theoretic function if we choose an appropriate machine and an appropriate sequence.

Theorem 4.2. *Every function $f \in \mathcal{F}$ is conditionally computable.*

Proof. The case of a finite domain D_f of f is obvious since each of the finitely many function values of f (if there are any) and the domain D_f can be explicitely encoded into a machine. So, let D_f be infinite, i.e. $D_f = \{a_1, a_2, \ldots, a_k, \ldots\}$ with $a_1 < a_2 < \cdots < a_k < \ldots$. Encoding f in a sequence we define such $\xi \in \mathbb{N}^\infty$ as follows

$$\begin{array}{lclclclcl}
\xi_1 & = & \xi_2 & = & \cdots & = & \xi_{f(a_2)+1} & = & a_1, \\
\xi_{f(a_2)+1+1} & = & \xi_{f(a_2)+1+2} & = & \cdots & = & \xi_{f(a_2)+1+f(a_3)+1} & = & a_2, \\
\vdots & & \vdots & & & & \vdots & & \vdots \\
\xi_{\sum_{j=2}^{k} f(a_j)+k} & = & \xi_{\sum_{j=2}^{k} f(a_j)+k+1} & = & \cdots & = & \xi_{\sum_{j=2}^{k+1} f(a_j)+k} & = & a_k, \\
\vdots & & \vdots & & & & \vdots & & \vdots
\end{array}$$

Clearly, using $f(a_1)$ as a compiled-in constant there is a persistent Turing machine K that reconstructs each function value $f(a_j)$ for $j \geq 2$ from the sequence ξ. This shows the conditional computability of any f with infinite domain. ❑

The following theorem gives connections between classical Turing computability and conditional computability by using robustness properties of some persistent machines.

Theorem 4.3. *Let $g \in \mathcal{F}$ and $g \not\simeq \nu$. There is a partial recursive function $f \in \mathcal{F}$ with $D_g \subseteq D_f$ and $f|_{D_g} \simeq g$ if and only if there exists a persistent Turing machine K such that (K, ξ) is a conditional computation of g for every $\xi \in \mathbb{N}^\infty$ with $R_\xi = D_g$.*

Proof. From left to right, note that each partial recursive function can be computed by a standard Turing machine M that finishs computations with cleared work tapes. For the other direction, it is sufficient to observe that the persistent machine K gives already right outputs for any $x \in D_g$ if x is the first element of a sequence ξ with $R_\xi = D_g$. ❑

From that theorem we immediately obtain some corollaries.

Corollary 4.4. *A function $f \in \mathcal{F}$ with $f \not\simeq \nu$ and recursively enumerable D_f is partial recursive if and only if there exists a persistent Turing machine K such that (K, ξ) is a conditional computation of f for every $\xi \in \mathbb{N}^\infty$ with $R_\xi = D_f$.*

Corollary 4.5. *A total function $f \in \mathcal{F}$ is recursive if and only if there exists a persistent Turing machine K such that (K, ξ) is a weakly conditional computation of f for every $\xi \in \mathbb{N}^\infty$.*

Usually, in recursion theory not general functions are investigated, but characteristic functions. So, the study is focussed on sets or languages. Clearly, the corresponding notions are also introducable into our model. Especially, the notion of a (consistent) persistent Turing acceptor is immediately to obtain from the standard Turing acceptor (see [Soa87]) and our specifications in Sect. 3. A (local, global) acceptance function *acc* is a 0-1-valued (local, global) output function of a persistent Turing machine. We use the terms *acceptor* and *machine* synonymously. The precise meaning yields from the context. Note that a characteristic function is a total one.

Definition 4.6. Let $A \subseteq \mathbb{N}$, let K be a (consistent) persistent Turing machine, and let $\xi \in \mathbb{N}_T^\infty$.

1. K *accepts* or *decides* A *under the condition* ξ if and only if $\chi_A = acc_K(x|\xi)$ for every $x \in \mathbb{N}$. (K, ξ) is called a *conditional acceptance of* A. As usual we employ the notation $L(K, \xi) = A$ to describe the conditional acceptance of A.

2. K *accepts* or *decides* A *conditionally* if and only if there is a sequence $\xi \in \mathbb{N}_T^\infty$ such that $L(K, \xi) = A$.

3. The set A is said to be *conditionally decidable* if and only there is a (consistent) persistent Turing machine K accepting A conditionally.

Results from Theorem 4.2 and Theorem 4.3 looks like as follows in terms of sets.

Corollary 4.7. *Let $A \subseteq \mathbb{N}$ be arbitrary.*

1. A is conditionally decidable.

2. $A \in$ REC if and only if there is a persistent Turing acceptor K such that $A = L(K, \xi)$ for every $\xi \in \mathbb{N}_T^\infty$.

5 Stability and Natural Oracles

In this section we will point out connections between persistent and oracle machines. For that, take a look at Theorem 4.2 where we have encoded whole functions in sequences (not uniquely). Computing a function we merely have to reproduce the function from the sequence, strictly speaking we reproduce the function from the initial segment of the sequence up to the current input. In this sense a sequence can be viewed as a natural or realistic oracle.

A critical point is that the construction makes too extreme use of the input sequence. Nothing is said about the function inside the corresponding machine except the value $f(a_1)$, and only inserting an a_j at the position in the sequence where the a_j's stand changes the function to be computed. The construction is rather sensitive.

To eliminate these problems we introduce a stability concept of persistent computations under certain sequences. For this purpose we select a possibly finite subsequence that is called the *relief.*

Definition 5.1. Let $\xi \in \mathbb{N}^\infty$. A sequence $\xi^R \in (\mathbb{N}^* \cup \mathbb{N}^\infty)$ is said to be the *relief of* ξ if and only if ξ^R is obtained from ξ by the following steps:

1. $\xi^R_1 := \xi_1$, and $X_1 := \{\xi_1\}$,
2. If there exists j with $j \notin X_i$ then $\xi^R_{i+1} := \xi_{\min\{j|\xi_j \notin X_i\}}$ and $X_{i+1} := X_i \cup \{\xi^R_{i+1}\}$, otherwise we stop with $\dim \xi^R = i$.

The relief of a sequence ξ contains the elements of R_ξ exactly one time and in that order they occur for the first time in ξ. For instance, the relief of $\xi = (0,2,1,1,2,5,0,4,3,\dots)$ begins with $(0,2,1,5,4,3,\dots)$. A persistent machine we can call to be stable if the global output function is equal under all sequences with the same relief.

Definition 5.2. Let K be a consistent persistent Turing machine.

1. Let $\xi \in \mathbb{N}^\infty$. K is said to be *relief-stable in* ξ if and only if for every $\eta \in \mathbb{N}^\infty$ holds that $\xi^R = \eta^R$ implies $out_K(\cdot|\xi) \simeq out_K(\cdot|\eta)$.
2. K is said to be *relief-stable* if and only if K is relief-stable in ξ for every $\xi \in \mathbb{N}^\infty$.

It is an open problem whether Theorem 4.2 holds for relief-stable machines. Obviously, functions not partial recursive can be computed conditionally by relief-stable machines since the cardinality of the set of all reliefs is greater than the cardinality of $\mathbb{N}$. Concretely, we have the following sufficient condition for conditional computability.

Theorem 5.3. *Let $f \in \mathcal{F}$ be an arbitrary function satisfying the following properties*

1. $R_f \in \mathrm{RE}$,

2. $\{\, a \in \mathbb{N} \mid \#f^{-1}(\{a\}) < \infty \,\} \in \mathrm{REC}$,

3. $\{\, x \in \mathbb{N} \mid f(x) \in \{\, a \in \mathbb{N} \mid \#f^{-1}(\{a\}) < \infty \,\} \,\} \in \mathrm{REC}.$

Then f is conditionally computable by relief-stable Turing machines.

Proof. If D_f is finite, the statement is obvious. Let D_f be infinite. Define $A := \{\, a \mid \#f^{-1}(\{a\}) < \infty \,\}$, $X := \{\, x \mid f(x) \in A \,\}$. At first, let $R_f \backslash A = \{f_1, f_2, \dots\}$ be infinite. The construction of a suitable relief-stable machine consists of two parts regarding to a current input x.

Case 1. If $x \in X$, then compute $f(x)$ with the help of the enumeration of A.

Case 2. If otherwise $x \notin X$, compute the index j of x in the relief of the input sequence. Calculate from j the number n in the relief restricted to inputs $z \in X$, and divide n into the left part n_l and the right part n_r according to the pairing function $\langle \cdot, \cdot \rangle_2$. Finally, output f_{n_l}.

Define $\eta \in \mathbb{N}^\infty$ to be any sequence with $R_\eta = X$, and $\varrho \in \mathbb{N}^\infty$ to be a sequence with $\varrho^R_n = x^{(f_{n_l})}_{n_r}$ where $f^{-1}(\{m\}) = \{x^{(m)}_1, x^{(m)}_2, \dots\}$. Notice that $R_\varrho = D_f \backslash X$. Evidently, the sequence ξ defined as $\xi := (\eta_1, \varrho_1, \eta_2, \varrho_2, \dots, \eta_k, \varrho_k, \dots)$ shows the conditional computability of f.

It remains to argue for $R_f \backslash A = \{f_0, f_1, \ldots, f_{k-1}\}$ is finite. The validity of the theorem in this case is easy to verify if we consider any sequence ϱ with $\varrho_j^R \in f^{-1}(\{f_{(j-1) \bmod k}\})$. Obviously, there is a relief-stable machine reproducing from such relief the function f according to the given inputs. ❑

As an immediate consequence, the relief-stable version of the theorem is valid for sets.

Corollary 5.4. *Every set $A \subseteq \mathbb{N}$ is conditionally decidable by relief-stable machines.*

Related to natural oracles we can state the following. Unfortunately, we have the condition that the equality of the output functions supposes infinite sets.

Theorem 5.5. *Let $M^{(\cdot)}$ be a standard deterministic oracle Turing machine that for every $A \subseteq \mathbb{N}$ and for every $x \in \mathbb{N}$ during the computation $M^{(A)}(x)$ asks only queries $y \in \mathbb{N}$ with $y \leq x$ and ever stops. Then there exists a relief-stable persistent Turing machine K, such that for every $A \subseteq \mathbb{N}$ not finite there is a sequence $\xi \in \mathbb{N}^\infty$ with $out_{M^{(A)}} \simeq out_K(\cdot|\xi)$.*

Proof. The idea is to fix an encoding T of a set A into a sequence ξ which makes possible both to interpret the order of occuring inputs in the sequence up to the current input as the corresponding part of A that can be asked during any computation, and to compute the right outputs. Consider a mapping T^* from $\mathbb{N}^\infty$ to $\mathfrak{P}\mathbb{N}$ defined as

$$T^*(\xi) := \{\, n \in \mathbb{N} \mid (\exists j)\big(\xi_j = n \wedge (\forall i < j)(\xi_i < \xi_j)\big) \,\}.$$

T^* has some properties that are easy to verify: (1) T^* is surjective, (2) $T^*(\xi) = T^*(\xi^R)$ for every $\xi \in \mathbb{N}^\infty$, and (3) for every $\xi \in \mathbb{N}^\infty$ holds that R_ξ is infinite if and only if $T^*(\xi)$ is infinite. T^* defines the origines of T. Having T^* in mind we can easily construct for any oracle machine $M^{(\cdot)}$ with above restrictions an appropriate relief-stable persistent machine K. Take into account that, since for every A the function $out_{M^{(A)}}$ is total, $R_\xi = \mathbb{N}$ holds for every $\xi \in \mathbb{N}^\infty$ with $T^*(\xi) = A$. ❑

6 Independence Degrees

Corollary 4.5 identifies total functions with highest independence in computations since for an arbitrary sequence always same outputs are given by a machine no matter if only relief-stable machines are admitted. Same considerations can be made about sets as stated in Corollary 4.7.

On the other side, a set only conditionally decidable by a machine which has different global acceptance functions for different input sequences can be assigned the lowest degree of independence. Surely, there are degrees in between.

A similar term to that of independence are studied in the setting of standard Turing machines. Together with the concept of helping, the notion of robustness of oracle machines was introduced by Schöning [Sch85] and subsequently investigated in various ways, e.g. [Ko87, Sch88, HH90, AKS95, NRS95]. A robust oracle machine is a standard oracle Turing machine accepting the same set relative to every oracle. Because of Theorem 5.5, our study of independence is loosely connected with the notion of robust oracle machine, and in such sense independences can be viewed as some kind of partial robustness.

In order to refine the concept of independence at first we consider uncritical inputs of a sequence, i.e. inputs swapping them do not change the global acceptance function. Our context are relief-stable machines.

Definition 6.1. Let $\xi \in \mathbb{N}_T^\infty$, and let $A \subseteq \mathbb{N}$.

1. A sequence $\eta \in \mathbb{N}_T^\infty$ is said to be a *A-permutation of ξ* if and only if there exists a bijective function $\beta : A \longrightarrow A$ such that for every $j \geq 1$

$$\eta_j^R = \begin{cases} \beta(\xi_j^R), & (\exists a \in A)(\xi_j^R = a), \\ \xi_j^R, & \text{otherwise.} \end{cases}$$

2. A set $L \subseteq \mathbb{N}$ is said to be *(A, ξ)-independent* if and only if there is a persistent Turing acceptor K such that $L = L(K, \eta)$ for every $\eta \in \mathbb{N}_T^\infty$ which is a A-permutation of ξ.

There would be some arbitrariness in our concept of independence degrees if we stop at this definition because of the consideration only of concrete set A. A statement about languages that are acceptable under conditions in which only two inputs are changeable would be impossible. To describe such or similar independences, systems of sets are necessary.

Definition 6.2. 1. A system $\mathcal{D}$ of subsets of $\mathbb{N}$ is called *independence.*

2. Let $\mathcal{D}$ be an independence. A set $L \subseteq \mathbb{N}$ is said to be *of independence $\mathcal{D}$* if and only if there exist a sequence $\xi \in \mathbb{N}_T^\infty$ and an $A \in \mathcal{D}$ such that L is (A, ξ)-independent.

3. Let $\mathcal{D}$ be an independence. The class of all sets $L \subseteq \mathbb{N}$ of independence $\mathcal{D}$ is called the *degree of independence $\mathcal{D}$* and is denoted by $\mathrm{Ind}(\mathcal{D})$, i.e.

$$\mathrm{Ind}(\mathcal{D}) = \{\, L \subseteq \mathbb{N} \mid L \text{ is of independence } \mathcal{D} \,\}.$$

Viewing as an operator, Ind is obviously monotonic. Degrees of independence with more complicated descriptions leads to larger classes of sets. Next results on singleton independences follow directly from definitions and results above. The proposition establishs the first simple structural relations between independence degrees.

Proposition 6.3. *1.* $\mathrm{Ind}(\{\varnothing\}) = \mathfrak{P}\mathbb{N}$, *and* $\mathrm{Ind}(\{\mathbb{N}\}) = \mathrm{REC}$.

2. Let $A, B \subseteq \mathbb{N}$. If $A \subseteq B$ then $\mathrm{Ind}(\{B\}) \subseteq \mathrm{Ind}(\{A\})$.

If we consider degrees of singleton independences of recursive sets, then the according independence degrees are precisely notable.

Theorem 6.4. *Let $A \in \mathrm{REC}$. Then* $\mathrm{Ind}(\{A\}) = \{\, L \subseteq \mathbb{N} \mid L \cap A \in \mathrm{REC} \,\}$.

Proof. Let $L \in \mathrm{Ind}(\{A\})$, i.e. there are a relief-stable persistent Turing machine K and a $\xi \in \mathbb{N}_T^\infty$ such that $L = L(K, \eta)$ for every A-permutation η of ξ. Without loss of generality, assume $\xi = \xi^R$. Let r be the least index of ξ with $\xi_r \in A$. Consider a standard Turing machine M that, on input x, checks $x \in A$, if checking was succesful then M simulates $acc_K\big(x \big| (\xi_1, \dots, \xi_{r-1})\big)$ and behaves accordingly, if otherwise checking $x \in A$ was wrong then M rejects. Since K works rightly for each A-permutation, M accepts x if and only if $x \in L \cap A$. Hence, $L \cap A \in \mathrm{REC}$.

Let $L \cap A \in \mathrm{REC}$. If $L \backslash A$ is finite then clearly $L \in \mathrm{REC}$ and, thus, $L \in \mathrm{Ind}(\{A\})$ by Proposition 6.3. It remains to prove the claim for the case $L \backslash A$ is infinite. So, let $L \backslash A = \{l_1, l_2, \dots\}$ with $l_1 < l_2 < \dots$. We will apply Theorem 5.5. Therefore, define a sequence $\xi \in \mathbb{N}_T^\infty$ to be

$$\xi = (l_1, 0, 1, \dots, l_1 - 1, l_2, l_1 + 1, l_1 + 2, \dots, l_2 - 1, l_3, l_2 + 1, \dots).$$

Employing operator T^* from the proof of Theorem 5.5, $T^*(\xi) = L\backslash A$. On the other hand, a standard oracle Turing machine $M^{(\cdot)}$ satisfying the assumption needed in Theorem 5.5 and giving $L\backslash A = L\big(M^{(L\backslash A)}\big)$, is easy to obtain. Hence, there is a relief-stable persistent machine K such that $L\backslash A = L(K,\xi)$ with ξ from above. One can transform K into K' without destroying relief-stability such that K', on the current input x, checks $x \in A$, if so (notice that then $x \notin L\backslash A$), then (1) takes over the answer whether $x \in L \cap A$, (2) computes y which is originally placed at the current position in ξ, and (3) generates the tape inscriptions like in the case y instead of x would have been asked; if $x \notin A$ then K' works like K. Since K' is controlling A, $L(K',\xi) = L(K',\xi')$ for every A-permutation ξ' of ξ. Moreover, $L(K',\xi) = (L\backslash A) \cup (L \cap A) = L$. Thus, $L \in \mathrm{Ind}(\{A\})$. ❑

Some corollaries are immediate. The first one shows that finitely many uncritical points in the sense above give as much structure in independence degrees as no point does.

Corollary 6.5. *If A is finite then* $\mathrm{Ind}(\{A\}) = \mathfrak{P}\mathbb{N}$.

For the second, observe that the independence degree according to a family of sets $\mathcal{D}$ is the same as the (finite, countable or uncountable) union of all independence degrees according to the singletons induced by the family $\mathcal{D}$. Note that if we admit REC then the independence degree contains whole $\mathfrak{P}\mathbb{N}$ because finite sets are included.

Corollary 6.6. $\mathrm{Ind}(\mathrm{REC}_+) = \big\{\, L \subseteq \mathbb{N} \;\big|\; L$ *contains an infinite recursive set* $\big\}$.

Corollary 6.6 can be expressed in terms of immunity. A set L is said to be *immune* if and only if L contains no infinite recursively enumerable (or equivalently, recursive) subset (see e.g. [Soa87]). So the right side describes even the class of non-immune sets.

Corollary 6.7. *An infinite set $L \in \mathbb{N}$ is of independence* REC_+ *if and only if L is not immune.*

7 Essential Computability

Another approach in quantifying independences of computations is to determine the probability with which a function can be conditionally computed. The highest degree of independence is associated with a function conditionally computable with probability 1, i.e. for almost every sequence.

There is a practical perspective behind that. This perspective can be expressed by the following question: Is it possible to increase the computability given by models of real machines, if we take the persistent view on them, but claiming correct computations only almost everytime when randomly choosing an input. It will turn out that this is not the case.

We formalize such understanding only for total functions $f \in \mathcal{F}_T$. Each consistent persistent Turing machine K can be interpreted as a mapping given by the global output function

$$K : \xi \longmapsto out_K(\cdot|\xi).$$

For our considerations it is necessary to show that K is a measurable mapping from $\big(\mathbb{N}^\infty, \mathfrak{C}(\mathbb{N}^\infty)\big)$ to $\big(\mathcal{F}, \hat{\mathfrak{C}}(\mathcal{F})\big)$. This is stated in the following.

Proposition 7.1. *Every consistent persistent Turing machine K is a random element with values in $\mathcal{F}$, i.e. K is $\mathfrak{C}(\mathbb{N}^\infty)\big/\hat{\mathfrak{C}}(\mathcal{F})$-measurable.*

Proof. It is sufficient to prove $\{\, \xi \in \mathbb{N}^\infty \mid K(\xi) \in A \,\} \in \mathfrak{C}(\mathbb{N}^\infty)$ for every set A from the generator $C(\mathbb{N}^\infty)$. We can further restrict ourselves to the simplest cylinders $\hat{C}_t(\{k\})$ with $t, k \in \mathbb{N}$, since these cylinders also generate $\hat{\mathfrak{C}}(\mathcal{F})$. Considering for arbitrary $t, k \in \mathbb{N}$ and every $m \in \mathbb{N}, m \geq 1$ the set

$$A^m_{t,k} := \big\{(n_1, \ldots, n_m) \;\big|\; \begin{array}{l} n_m = t \wedge (\forall 1 \leq j \leq m-1)(n_j \neq t) \wedge \\ out_K\big(n_m|(n_1, \ldots, n_{m-1})\big) = k \big\} \end{array}$$

we have immediately

$$\{\, \xi \in \mathbb{N}^\infty \mid K(\xi) \in \hat{C}_t\big(\{k\}\big) \,\} = \bigcup_{m=1}^{\infty} C_{1,\ldots,m}(A^m_{t,k})$$

which is obviously in $\mathfrak{C}(\mathbb{N}^\infty)$. ❑

For convenience we allow in our probability field every sequence to be from $\mathbb{N}^\infty$. As a consequence, conditional computations would be only weakly conditional computations. A way out is to consider all possible cuttings of the domain of a total function. So, let us define for any function $f \in \mathcal{F}_T$ the set $T_f := \{\, f|_A \mid A \subseteq \mathbb{N} \,\}$. Instead $\{\, \xi \in \mathbb{N}^\infty \mid K(\xi) \in T_f \,\}$ use as a shorthand $\{K(\xi) \in T_f\}$. Because $T_f \in \hat{\mathfrak{C}}(\mathcal{F})$ for every $f \in \mathcal{F}_T$ as can be seen by

$$T_f = \bigcap_{n=0}^{\infty} \big(\overline{\hat{C}_n(\mathbb{N})} \cup \hat{C}_n(\{f(n)\})\big),$$

asking for the probability of the set $\{K(\xi) \in T_f\}$ is well-founded. This gives background to speak in some sense about computation probabilities.

Fix an arbitrary distribution function d not necessarily plausible.

Definition 7.2. 1. For any function $f \in \mathcal{F}_T$ and for every persistent Turing machine K the *computation probability* $\mathbf{B}_K(f)$ *of* f *by* K is defined as

$$\mathbf{B}_K(f) := \mathbf{P}_d\big(\{K(\xi) \in T_f\}\big).$$

2. For any function $f \in \mathcal{F}_T$, the *computation probability* $\mathbf{B}(f)$ *of* f is defined as

$$\mathbf{B}(f) := \sup\{\, \mathbf{B}_K(f) \mid K \text{ is a persistent Turing machine} \,\}.$$

Of most considerable interest with respect to computation probabilities are the distinguished values 0 and 1. This is formally stated in the following. First statement is an anologon to the Lebesgue Density Theorem (see [Rog67, Sac66, MW95]) and is similar to be proven.

Theorem 7.3. *Let* $f \in \mathcal{F}_T$.

1. *If there exists a persistent Turing machine* K *such that* $\mathbf{B}_K(f) > 0$ *then for every* $\delta > 0$ *there is a persistent Turing machine* K' *such that* $\mathbf{B}_{K'}(f) \geq 1 - \delta$.

2. $\mathbf{B}(f)$ *is either equal to* 1 *or equal to* 0.

Proof. The second statement is an easy consequence of (1). So, let $\mathbf{B}_K(f) > 0$, and K be a persistent Turing machine. Thus there is an $\varepsilon > 0$ such that $\mathbf{P}_d\big(\{K(\xi) \in T_f\}\big) = \varepsilon > 0$. Hence, for every $n \geq 1$

$$\begin{aligned}
\mathbf{P}_d\big(\{K(\xi) \in T_f\}\big) &= \mathbf{P}_d\big(\{K(\xi) \in T_f\} \cap \{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) \\
&= \mathbf{P}_d\big(\{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) \cdot \\
&\quad\ \mathbf{P}_d\big(\{K(\xi) \in T_f\} \bigm| \{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) \\
&= \varepsilon.
\end{aligned}$$

Using the fact

$$\{K(\xi) \in T_f\} = \bigcap_{n=1}^{\infty} \{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}$$

it follows

$$\lim_{n\to\infty} \mathbf{P}_d\big(\{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) = \varepsilon$$

and

$$\lim_{n\to\infty} \mathbf{P}_d\big(\{K(\xi) \in T_f\} \bigm| \{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) = 1.$$

Now, let $\delta > 0$. Then there is an n_0 such that for every $n \geq n_0$ we have

$$\mathbf{P}_d\big(\{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) \geq 1 - \delta.$$

Let X be the set of finite sequences $\xi \in \mathbb{N}^*$ which satisfiy (i) $f(\xi_j) = \mathit{out}_K\big(\xi_j|(\xi_1, \ldots, \xi_{j-1})\big)$ for every $1 \leq j \leq n_0$, and (ii) for every initial segment $\xi' \sqsubset \xi$ there is a $1 \leq j \leq n_0$ such that $f(\xi') \neq \mathit{out}_K\big(\xi' \bigm| (\xi'_1, \ldots, \xi'_{j-1})\big)$. For $\xi \in X$ define $P_1(\xi) := \mathbf{P}_d\big(\{\xi \sqsubseteq \eta\}\big)$ and $P_2(\xi) := \mathbf{P}_d\big(\{K(\xi) \in T_f\} \bigm| \{\xi \sqsubseteq \eta\}\big)$. Then clearly

$$\mathbf{P}_d\big(\{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) = \sum_{\xi \in X} P_1(\xi) \geq \varepsilon > 0,$$

and further

$$\mathbf{P}_d\big(\{K(\xi) \in T_f\}\big) = \sum_{\xi \in X} \mathbf{P}_d\big(\{K(\xi) \in T_f\} \cap \{\xi \sqsubseteq \eta\}\big) = \sum_{\xi \in X} P_1(\xi) \cdot P_2(\xi).$$

Hence, we immediately obtain

$$\mathbf{P}_d\big(\{K(\xi) \in T_f\} \bigm| \{f(\xi_j) = \mathit{out}_K(\xi_j|\xi) \text{ for all } 1 \leq j \leq n\}\big) = \frac{\sum_{\xi \in X} P_1(\xi) \cdot P_2(\xi)}{\sum_{\xi \in X} P_1(\xi)} \geq 1 - \delta.$$

Obviously, so there must be a $\xi^0 \in X$ with $P_2(\xi^0) = \mathbf{P}_d\big(\{K(\xi) \in T_f\} \bigm| \{\xi^0 \sqsubseteq \eta\}\big) \geq 1 - \delta$. Now consider the persistent Turing machine K' that, on input x currently given, (1) checks whether x is the initial input, (2) if it is true simulates K on the input sequence ξ^0, and in any case (3) continues on x like K does. Obviously, $\mathbf{B}_{K'}(f) = \mathbf{P}_d\left(\{K'(\xi) \in T_f\}\right) = \mathbf{P}_d\left(\{K(\xi) \in T_f\} \middle| \{\xi^0 \sqsubseteq \eta\}\right) \geq 1 - \delta$. ❑

This theorem shows that there are only two different classes of functions which are in obvious sense complementary: essentially computable and singularely computable functions. We define only essential computability.

Definition 7.4. Let f be a function from $\mathcal{F}_T$. Then f is said to be *essentially computable* if and only $\mathbf{B}(f) = 1$.

If we let the distribution function d be plausible then the following result shows that essential computability coincides with Turing computability when total function are considered. The proof generalizes a theorem of Sacks [Sac66, Rog67].

Theorem 7.5. *Let all notions be defined according to a plausible distribution function d. Let $f \in \mathcal{F}_T$ be an arbitrary function. Then f is essentially computable if and only if f is recursive.*

Proof. The inclusion from right to left is trivial. Define for a set $A \subseteq \mathbb{N}$,

$$Seq(A) := \bigcup_{\langle n_1,\ldots,n_k\rangle \in A} Z_{1,\ldots,k}\big(\{n_1\} \times \cdots \times \{n_k\}\big).$$

Notice that the value $\mathbf{P}_d\big(Seq(A)\big)$ is Turing computable for any finite set A. Now, let f be essentially computable. Then there is a persistent Turing machine K with $\mathbf{B}_K(f) \geq \frac{3}{4}$. Consider the function g defined by the following algorithm for an arbitrary input x

```
Let $n := 0, m := 0$, and $F_k := \varnothing$ for every $k \in \mathbb{N}$;
**while** $\max_{k<m} \mathbf{P}_d\big(Seq(F_k)\big) \leq \frac{1}{2}$ **do**
    Simulate $K$ one step on every inital segment $\langle \xi_1^j, \ldots, \xi_{r_j}^j \rangle = j$ with $0 \leq j \leq n$;
    **if** $K$ halts on a segment $(\xi_1^j, \ldots, \xi_{r_j-1}^j, x)$ with $0 \leq j \leq n$ **then**
        $s := out_K\big(x \big| (\xi_1^j, \ldots, \xi_{r_j-1}^j)\big)$;
        $F_s := F_s \cup \{j\}$;
        **if** $s > m$ **then** $m := s$;
    **else** $n := n + 1$;
**endwhile**;
Output $k$ with $\mathbf{P}_d\big(Seq(F_k)\big)$ maximal.
```

Clearly, g is Turing computable. Moreover, g is total and equal to f, both because $\mathbf{B}_K(f) \geq \frac{3}{4}$. Hence, f is total recursive. ❑

8 Conclusion

In the previous sections we have seen how the interaction mode called persistence can increase the computational power of algorithms. Every non-trivial function is conditionally computable if we take an observer's point of view. Additional restrictions to machines allowed to consider leads to weaker results. Here, the stress in on *machines*. What if we restrict the world from which the sequences can be taken? This touches the concept of environments.

Let us only consider sets. Then, as an example, a trivial observation in Theorem 4.2 is that already in the world $E_{mon} = \big\{\, \xi \in \mathbb{N}_T^\infty \;\big|\; (\forall j)(\xi_j \leq \xi_{j+1}) \,\big\}$ every set $A \subseteq \mathbb{N}$ can be conditionally decided. One could say that this environment E_{mon} generates $\mathfrak{P}\mathbb{N}$.

Clearly, every environment E with at least one full sequence ξ—an E containing exactly one sequence models a deterministic world—generates independently from the chosen machine type all recursively decidable sets. If we consider only environments of recursive sequences, what captures a strongly predictable world, then obviously we do not leave the world of recursive sets. So, the observer's view on persistent computations is only a worthwile study if we adopt the world in which machines are working to be inpredictable, i.e. non-Turing-computable.

References

[AKS95] V. Arvind, J. Köbler, and R. Schuler. On helping and interactive proof systems. *International Journal on Foundations of Computer Science*, 6(2):137–153, 1995.

[CG78] R. S. Cohen and A. Y. Gold. ω-computations on Turing machines. *Theoretical Computer Science*, 6:1–23, 1978.

[CG80] R. S. Cohen and A. Y. Gold. On the complexity of ω-type Turing acceptors. *Theoretical Computer Science*, 10:249–272, 1980.

[Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications* I. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 3rd edition, 1968.

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

[HH90] J. Hartmanis and L. A. Hemachandra. Robust machines accept easy sets. *Theoretical Computer Science*, 74(2):217–225, 1990.

[Ko87] K. Ko. On helping by robust oracle machines. *Theoretical Computer Science*, 52:15–36, 1987.

[Ko91] K. Ko. *Complexity Theory of Real Functions.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[Kos98] S. Kosub. Some remarks towards dynamic analysis of computation. Manuscript, March 1998.

[KW85] C. Kreitz and K. Weihrauch. Theory of representations. *Theoretical Computer Science*, 38:35–53, 1985.

[Mil93] R. Milner. Elements of interaction. *Communications of the ACM*, 36:78–89, 1993.

[MW95] W. Merkle and Y. Wang. Separations by random oracles and "Almost" classes for generalized reducibilities. In *Proceedings 20th Symposium on Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 179–190. Springer-Verlag, 1995.

[NRS95] A. V. Naik, K. W. Regan, and D. Sivakumar. On quasilinear-time complexity theory. *Theoretical Computer Science*, 148(2):325–349, 1995.

[Rog67] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.

[Sac66] G. E. Sacks. *Degrees of Unsolvability.* Number 55 in Annals of Mathematical Studies. Princeton University Press, 2., revised edition, 1966.

[Sch85] U. Schöning. Robust algorithms: A different approach to oracles. *Theoretical Computer Science*, 40:57–66, 1985.

[Sch88] U. Schöning. Robust oracle machines. In *Proceedings 13th Symposium on Mathematical Foundations of Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 93–106. Springer-Verlag, 1988.

[Shi95] A. N. Shiryaev. *Probability.* Number 95 in Graduate Texts in Mathematics. Springer-Verlag, 2nd edition, 1995.

[Soa87] R. I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer-Verlag, 1987.

[SW78] L. Staiger and K. W. Wagner. Rekursive Folgenmengen I. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 24:523–538, 1978. A preliminary version appeared as: K. W. Wagner and L. Staiger, Recursive ω-languages, *Proceedings 2nd International Conference on Fundamentals of Computation Theory*, volume 56 in Lecture Notes in Computer Science, pages 532-537. Springer-Verlag, 1977.

[Weg98] P. Wegner. Interactive foundations of computing. *Theoretical Computer Science*, 192:315–351, 1998.

[Wei85] K. Weihrauch. Type 2 recursion theory. *Theoretical Computer Science*, 38:17–33, 1985.