HAL open science

The decidable properties of subrecursive functions

Mathieu Hoyrup

To cite this version:

Mathieu Hoyrup. The decidable properties of subrecursive functions. International Colloquium on Automata, Languages, and Programming (ICALP) 2016, Jul 2016, Rome, Italy. 10.4230/LIPIcs.ICALP.2016.108 . hal-01308224

HAL Id: hal-01308224

https://inria.hal.science/hal-01308224v1

Submitted on 27 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

The decidable properties of subrecursive functions

Mathieu Hoyrup[1]

1 **LORIA, Inria**
615, rue du jardin botanique
54600 Villers-lès-Nancy, France
mathieu.hoyrup@inria.fr

Abstract

What can be decided or semidecided about a primitive recursive function, given a definition of that function by primitive recursion? What about subrecursive classes other than primitive recursive functions? We provide a complete and explicit characterization of the decidable and semidecidable properties. This characterization uses a variant of Kolmogorov complexity where only programs in a subrecursive programming language are allowed. More precisely, we prove that all the decidable and semidecidable properties can be obtained as combinations of two classes of basic decidable properties: (i) the function takes some particular values on a finite set of inputs, and (ii) every finite part of the function can be compressed to some extent.

1998 ACM Subject Classification F.1.1 Models of Computation; F.4.3 Formal Languages/Decision Problems

Keywords and phrases Rice theorem, subrecursive class, decidable property, Kolmogorov complexity, compressibility

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

What can be decided about a function $f : \mathbb{N} \to \mathbb{N}$ if f is represented by a program computing it? What can be semidecided?

In the 50's, many computability theory results have been proved in order to answer these questions. The answers depend on the class of functions considered.

Partial computable functions.

For the class of partial computable functions, Rice [11] proved that no non-trivial property is decidable, and Shapiro [12] refined it by characterizing the semidecidable properties. These results show that having a program computing f does not give more information than having an oracle giving the values of f, in the sense that the two presentations induce the same classes of decidable and semidecidable properties. In other words, the only way of exploiting a program computing f is to execute it on any input to obtains the values of f. Hence the code of the program contains no more information than a black-box containing the program.

Total computable functions.

For the class of total computable functions, Kreisel, Lacombe, Shœnfield [8] and independently Ceitin [2] characterized the decidable properties. Again they are the same whether the function is presented by a program or by an oracle.

However, Friedberg [4] showed that the semidecidable properties of total computable functions do not admit such a characterization. In that case, having a program computing

© Mathieu Hoyrup;
licensed under Creative Commons License CC-BY
Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

f gives more information than having the values of f. In [6] we proved that the extra information is simply the size of the program. A remaining open problem is to obtain a nice characterization of the semidecidable properties of total computable functions, when they are presented by programs.

Subrecursive classes.

In this paper we investigate the case of a subrecursive class, i.e. a class of total computable functions that can be effectively enumerated. Examples of such classes are the primitive recursive functions, complexity classes such as FPTIME, or the class of provably total functions in Peano Arithmetic. Here the function in the class is presented by a program in a restricted programming language. As for the whole class of total computable functions, the semidecidable properties are not the same if the function is presented by an oracle. However we obtain a characterization of the semidecidable properties, when the function is presented by a subrecursive program. This characterization uses a version of Kolmogorov complexity restricted to a subrecursive programming language. This is the main result of the paper.

We also prove that the semidecidable properties of total computable functions do not admit an analogous characterization.

We also discuss the difference between having an oracle giving the values of a function, and a black-box containing a program computing that function. The difference is that no assumption is made on the time it takes for the oracle to answer, while a program in a black-box has particular halting times. We show that in general it does not make any difference, but we also show a situation where the halting times of the black-box can be exploited.

1.1 Notations.

Let $\mathbb{N}^*$ be the set of finite sequences of natural numbers and $\mathbb{N}^{\mathbb{N}}$ the Baire space of functions from $\mathbb{N}$ to $\mathbb{N}$. Given $f \in \mathbb{N}^{\mathbb{N}}$ and $n \in \mathbb{N}$, $f\restriction_n$ denotes the finite sequence $(f(0), \ldots, f(n-1)) \in \mathbb{N}^*$. We say that $f \in \mathbb{N}^{\mathbb{N}}$ extends $v = (v_0, \ldots, v_{n-1}) \in \mathbb{N}^*$ if $f(0) = v_0, \ldots, f(n-1) = v_{n-1}$, i.e. if $f\restriction_n = v$. We denote by $[v] \subseteq \mathbb{N}^{\mathbb{N}}$ the set of all extensions of v and call it a cylinder. The Baire space is endowed with the topology whose open sets are unions of cylinders. An effective open set is the union of a computable sequence of cylinders.

2 Decidable and semidecidable properties

In this paper, a ***subrecursive class of functions*** is simply a class $\mathscr{C}$ of total computable functions that can be computably enumerated: there is a numbering $\mathscr{C} = \{f_i : i \in \mathbb{N}\}$ such that f_i is computable uniformly in i, i.e. the mapping $(i, n) \mapsto f_i(n)$ is computable. If $f \in \mathscr{C}$ then a $\mathscr{C}$-index of f is any i such that $f = f_i$ (a function may have several indices). There are usually many ways of indexing such a class, and they may induce different decidable properties. A thorough investigation about indexings of subrecursive classes can be found in [7].

Examples of subrecursive classes are: the primitive recursive functions, the class FPTIME of polynomial-time computable functions, the class of provably total computable functions in Peano arithmetic. A numbering of a class can be obtained from a numbering of the programs in a subrecursive programming language, or a restricted model of computation. Hence having an index of a function is usually equivalent to having a program for that function, in the restricted language. Famous examples of restricted programming language

are: definitions by primitive recursion, LOOP programming language [10], polynomially clocked Turing machines, proofs of totality in Peano Arithmetic.

Observe that two programming languages admitting computable translation procedures in both directions induce the same decidable and semidecidable properties. This is for instance the case of definitions by primitive recursion and LOOP programs.

Let $A \subseteq \mathscr{C}$. First observe that the property $f \in A$ is semidecidable given f by an oracle exactly when A is the intersection of an effective open subset of the Baire space with $\mathscr{C}$. Indeed, when the machine semideciding $f \in A$ accepts f in finite time so it has only read a finite segment of f hence it will accept all functions in some cylinder $[f\restriction_n]$. The property $f \in A$ is decidable given f by an oracle exactly when both A and $\mathscr{C} \setminus A$ are the intersections of effective open sets with $\mathscr{C}$.

The goal of this paper is to obtain a similar understanding of the decidable and semidecidable sets $A \subseteq \mathscr{C}$, when $f \in \mathscr{C}$ is presented by a $\mathscr{C}$-index rather than an oracle.

In order to investigate this problem we introduce a notion of Kolmogorov complexity adapted to the class $\mathscr{C}$.

▶ **Definition 2.1.** The $\mathscr{C}$-complexity of $f : \mathbb{N} \to \mathbb{N}$ is

$$K_{\mathscr{C}}(f) = \begin{cases} \min\{i : f_i = f\} & \text{if } f \in \mathscr{C}, \\ +\infty & \text{otherwise.} \end{cases}$$

If $v = (v_0, \ldots, v_n)$ is a finite sequence of natural numbers then its $\mathscr{C}$-complexity is

$$K_{\mathscr{C}}(v) = \min\{i : f_i \text{ extends } v\} = \min\{K_{\mathscr{C}}(f) : f \in [v]\}.$$

If no f_i extends v then $K_{\mathscr{C}}(v) = +\infty$. Observe that for $f : \mathbb{N} \to \mathbb{N}$, $K_{\mathscr{C}}(f\restriction_n)$ is nondecreasing and converges to $K_{\mathscr{C}}(f)$ (which may be infinite). By the assumptions on $\mathscr{C}$, the quantity $K_{\mathscr{C}}(v)$ is computable from v (when it is finite – in, general, the predicate $K_{\mathscr{C}}(v) = i$ is decidable), contrary to usual Kolmogorov complexity which is upper semicomputable only. However $K_{\mathscr{C}}$ usually does not belong to the class $\mathscr{C}$ (modulo encoding of $\mathbb{N}^*$ in $\mathbb{N}$).

It may seem more consistent with usual notions of Kolmogorov complexity (see e.g. [9]) to take for instance $\log(i)$ instead of i in the definition, or to use a machine that is universal for the class $\mathscr{C}$ and define $K_{\mathscr{C}}$ in terms of the size of its inputs. All these choices are equally acceptable and lead exactly to the same result. The important point is that for each such notion of complexity K', an upper bound on $K_{\mathscr{C}}(f)$ can be uniformly computed from any upper bound on $K'(f)$ and vice-versa. Here we take the simplest definition of complexity following directly from the enumeration of $\mathscr{C}$, to avoid technicality.

2.1 An index gives more information than an oracle

Kreisel-Lacombe-Shœnfield/Ceitin's theorem implies that the properties of total computable functions that are decidable from indices are also decidable from oracles. In essence, Rice's theorem states the same for the class of partial computable functions. In general the situation is different when restricting to some subrecursive class: there exist properties that are decidable from indices but not from oracles.

Let us first give a concrete class of such properties. A ***computable order*** is a nondecreasing unbounded computable function $h : \mathbb{N} \to \mathbb{N}$, such as $\lfloor \log(n) \rfloor$, n^2 or 2^n for instance.

▶ **Definition 2.2.** Let $\mathscr{C}$ be a subrecursive class of functions and h a computable order. We define the set $A_{\mathscr{C},h}$ of $(\mathscr{C}, h)$***-anticomplex*** functions as

$$A_{\mathscr{C},h} = \{f : \mathbb{N} \to \mathbb{N} : \forall n, K_{\mathscr{C}}(f\restriction_n) \leq h(n)\}.$$

We borrow the terminology from [3], where the notion of anti-complex set is defined in terms of usual Kolmogorov complexity, and is studied from a computability-theoretic perspective.

▶ **Proposition 2.1.** For $f \in \mathscr{C}$, the property $f \in A_{\mathscr{C},h}$ is decidable given any $\mathscr{C}$-index of f.

Proof. Given an index i for f, one has $K_{\mathscr{C}}(f\restriction_n) \leq K_{\mathscr{C}}(f) \leq i$ for all n, so f belongs to $A_{\mathscr{C},h}$ if and only if $K_{\mathscr{C}}(f\restriction_n) \leq h(n)$ for all n such that $h(n) < i$. This property is decidable as $K_{\mathscr{C}}(f\restriction_n)$ is computable from i and n and only a finite number of values of n has to be checked. ◀

Note that it is important that h be unbounded. One can easily show that if for each $g \in \mathscr{C}$ the set $\{j \in \mathbb{N} : f_j = g\}$ is not decidable (usual subrecursive classes satisfy this condition), then when h is bounded the property $f \in A_{\mathscr{C},h}$ is not decidable given any $\mathscr{C}$-index of f.

In general $A_{\mathscr{C},h}$ is no more decidable if instead of an index of f one is only given f as oracle.

▶ **Proposition 2.2.** If $\mathscr{C}$ is dense in $\mathbb{N}^{\mathbb{N}}$ then $A_{\mathscr{C},h}$ has empty interior in $\mathscr{C}$ (i.e. does not contain the intersection of a cylinder with $\mathscr{C}$), therefore $A_{\mathscr{C},h}$ is not semidecidable when the input function is given as oracle.

Proof. For each $u = (u_0, \ldots, u_{n-1}) \in \mathbb{N}^*$, there exist only finitely many $i \in \mathbb{N}$ such that $K_{\mathscr{C}}(u_0, \ldots, u_{n-1}, i) \leq h(n+1)$. Take any i outside this finite set: the cylinder $[u_0, \ldots, u_{n-1}, i]$ is disjoint from $A_{\mathscr{C},h}$ but intersects $\mathscr{C}$, so $[u] \cap \mathscr{C}$ is not contained in $A_{\mathscr{C},h}$. ◀

All the usual subrecursive classes are dense in $\mathbb{N}^{\mathbb{N}}$. Observe that in computational complexity theory, one is more often interested in classes of *problems* rather than *functions*. Hence $\mathscr{C}$ could be the class of characteristic functions of subsets of $\mathbb{N}$ in some complexity class, such as P. In that case, $\mathscr{C}$ is not dense in $\mathbb{N}^{\mathbb{N}}$, however it is dense in $\{0,1\}^{\mathbb{N}}$ and a similar result holds.

▶ **Proposition 2.3.** The same result holds if $\mathscr{C}$ is dense in $\{0,1\}^{\mathbb{N}}$ and h is sufficiently small.

Proof. Let g be a computable order such that the number of finite sequences v such that $K_{\mathscr{C}}(v) \leq k$ is bounded by $g(k)$. If $K_{\mathscr{C}}$ is the notion of complexity from Definition 2.1 then one can take $g(k) = k+1$. For other notions of complexity based on length of binary programs, one could take $g(k) = 2^{k+1}$ instead.

Let h be a computable order such that $g \circ h(n) = o(2^n)$ (take for instance $h(n) = n$ in our case). Given $u \in \{0,1\}^*$, there exists $n \geq |u|$ such that $g \circ h(n) < 2^{n-|u|}$. By definition of g there are at most $g(h(n)) < 2^{n-|u|}$ finite sequences v such that $K_{\mathscr{C}}(v) \leq h(n)$. As there exist $2^{n-|u|}$ binary extensions of u of length n, at least one of them satisfies $K_{\mathscr{C}}(v) > h(n)$. Hence $[v] \cap \mathscr{C}$ is disjoint from $A_{\mathscr{C},h}$ and non-empty, so $A_{\mathscr{C},h}$ does not contain $[u] \cap \mathscr{C}$. ◀

The result does not hold for any class $\mathscr{C}$: if $\mathscr{C}$ is the class of constant functions, numbered in the obvious way, then having an oracle for $f \in \mathscr{C}$ is equivalent to having a $\mathscr{C}$-index of f. This class has the particular property that all the functions in $\mathscr{C}$ are isolated from each other and as we now show this is the only obstruction to generalizing Propositions 2.2 and 2.3. We recall that a function f is not isolated in $\mathscr{C}$ if for each $p \in \mathbb{N}$ there exists some $g \neq f$ in $\mathscr{C}$ such that $g\restriction_p = f\restriction_p$.

▶ **Proposition 2.4.** If $\mathscr{C}$ contains a function f that is not isolated in $\mathscr{C}$ then there is a computable order h such that f belongs to $A_{\mathscr{C},h}$ but not to its interior, therefore $A_{\mathscr{C},h}$ is not semidecidable when the input function is given as oracle.

Hence having an index for $f \in \mathscr{C}$ usually gives more information than having an access to f via an oracle, as it enables to decide more properties of f. What is the additional information? Having an index for f obviously bounds $K_{\mathscr{C}}(f)$, and we now show that in a sense this is the only additional information.

First observe that the proof of Proposition 2.1 actually shows that $A_{\mathscr{C},h}$ remains decidable if one is given f via an oracle *together with an upper bound on* $K_{\mathscr{C}}(f)$. This is the case of every decidable, and even semidecidable property.

▶ **Proposition 2.5.** Let $A \subseteq \mathscr{C}$ be such that the problem $f \in A$ is semidecidable given a $\mathscr{C}$-index of f. Then the problem $f \in A$ is semidecidable given an access to f as oracle together with any upper bound on $K_{\mathscr{C}}(f)$.

Proof. Given f and $k \geq K_{\mathscr{C}}(f)$, one can progressively reject all numbers $i \leq k$ such that $f_i \neq f$. In parallel one can progressively accept all numbers $i \leq k$ that are accepted by the semidecision procedure for A. Wait for a stage when every number $i \leq k$ is accepted or rejected. If this happens then accept f. ◀

If one defines $\mathscr{C}_k = \{f : K_{\mathscr{C}}(f) \leq k\} = \{f_i : i \leq k\}$ then Proposition 2.5 implies the existence of uniformly effective open sets $U_k \subseteq \mathbb{N}^{\mathbb{N}}$ such that $A \cap \mathscr{C}_k = U_k \cap \mathscr{C}_k$ for all $k \in \mathbb{N}$. Indeed, U_k is defined as the union of finite prefixes of oracles f read and accepted by the machine semideciding A, given k as upper bound on $K_{\mathscr{C}}(f)$.

It was proved in [6] that such a result also holds for the class of total computable functions and much more general classes of computable objects. The proof given here in the case of a subrecursive class $\mathscr{C}$ is much easier because we only deal with total programs (every f_i is total, so one can always recognize whether $f_i \neq f$).

2.2 The main result

We can now state our main result: the cylinders and the sets of anticomplex functions are the basic decidable properties, from which all decidable and semidecidable properties can be obtained.

▶ **Theorem 2.3.** *Let $\mathscr{C}$ be a subrecursive class and $A \subseteq \mathscr{C}$. The following conditions are equivalent:*

1. *The property $f \in A$ is semidecidable given a $\mathscr{C}$-index of f,*
2. *A is an effective union of sets of the form $[v] \cap A_{\mathscr{C},h}$, i.e.*

$$A = \mathscr{C} \cap \bigcup_n ([v_i] \cap A_{\mathscr{C},h_i})$$

for some computable sequences of finite words $v_i \in \mathbb{N}^*$ and orders $h_i : \mathbb{N} \to \mathbb{N}$.

Proof. We prove that 1. implies 2., the other direction being a direct consequence of Proposition 2.1. We slightly reformulate the property $A_{\mathscr{C},h}$, using the following sets. Recall the sets $\mathscr{C}_k = \{f_i : i \leq k\}$ defined after Proposition 2.5. For $k, n \in \mathbb{N}$ let

$$\mathscr{C}_k^n = \bigcup_{u \in \mathbb{N}^n : K_{\mathscr{C}}(u) \leq k} [u] = \bigcup_{f \in \mathscr{C}_k} [f\restriction_n] = \bigcup_{i \leq k} [f_i\restriction_n].$$

Observe that $f \in A_{\mathscr{C},h}$ if and only if $f \in \mathscr{C}_{h(n)}^n$ for all n.

Assume that condition 1. in the statement of the theorem holds. By Proposition 2.5 there exist uniformly effective open sets $U_k \subseteq \mathbb{N}^{\mathbb{N}}$ such that $A \cap \mathscr{C}_k = U_k \cap \mathscr{C}_k$. One can take $U_{k+1} \subseteq U_k$, replacing U_k with $U_k \cup U_{k+1} \cup \ldots$ if necessary. It follows that $A = \mathscr{C} \cap \bigcap_k U_k$.

Given $k \in \mathbb{N}$ and $v \in \mathbb{N}^*$ such that $[v] \subseteq U_{k+1}$, we now build a computable order h such that $[v] \cap \mathscr{C}_{k+1} \subseteq [v] \cap A_{\mathscr{C},h} \cap \mathscr{C} \subseteq A$. In order to obtain the announced families v_i and h_i to cover the whole set A, we will simply start from all possible $k \in \mathbb{N}$ and all v in the effective enumeration of U_{k+1}.

We now define h, by first constructing a kind of inverse of h. More precisely we define a computable increasing sequence n_i such that for all i,

$$[v] \cap \mathscr{C}_{k+1}^{n_1} \cap \ldots \cap \mathscr{C}_{k+i}^{n_i} \subseteq U_{k+i+1}.$$

The base case $i = 0$ is satisfied as $[v] \subseteq U_{k+1}$. Once $n_1, \ldots, n_i$ have been defined,

$$\begin{aligned}[v] \cap \mathscr{C}_{k+1}^{n_1} \cap \ldots \cap \mathscr{C}_{k+i}^{n_i} \cap \mathscr{C}_{k+i+1} &\subseteq U_{k+i+1} \cap \mathscr{C}_{k+i+1} \\ &\subseteq A \\ &\subseteq U_{k+i+2}.\end{aligned}$$

The left-hand side is a finite set. For each f in that set, there is $n \in \mathbb{N}$ such that $[f\restriction_n] \subseteq U_{k+i+2}$. As the set is finite there is a single n that works for each f in the finite set. As this finite set is computable, such a n can be computed. We then define $n_{i+1} > n_i$ such that

$$[v] \cap \mathscr{C}_{k+1}^{n_1} \cap \ldots \cap \mathscr{C}_{k+i}^{n_i} \cap \mathscr{C}_{k+i+1}^{n_{i+1}} \subseteq U_{k+i+2}.$$

We then have

$$[v] \cap \mathscr{C}_{k+1} \subseteq [v] \cap \bigcap_{i \geq 1} \mathscr{C}_{k+i}^{n_i} \cap \mathscr{C} \subseteq \bigcap_n U_n \cap \mathscr{C} = A. \tag{1}$$

Let h be the computable order defined by $h(n) = k + \min\{i \geq 1 : n \leq n_i\}$. We claim, as announced, that

$$[v] \cap \mathscr{C}_{k+1} \subseteq [v] \cap A_{\mathscr{C},h} \cap \mathscr{C} \subseteq A.$$

The first inequality is straightforward: $h(n) \geq k+1$ for all n, so $\mathscr{C}_{k+1} \subseteq A_{\mathscr{C},h}$.

To prove the second inequality, observe that if $g \in A_{\mathscr{C},h}$ then for all i, $K_{\mathscr{C}}(g\restriction_{n_i}) \leq h(n_i) = k+i$, i.e. $g \in \bigcap_{i \geq 1} \mathscr{C}_{k+i}^{n_i}$, and then use (1). ◀

A set $A \subseteq \mathscr{C}$ is then decidable from $\mathscr{C}$-indices if and only if both A and $\mathscr{C} \setminus A$ can be expressed as effective unions of sets $[v] \cap A_{\mathscr{C},h}$.

A further question.

Each effective numbering of a class $\mathscr{C}$ induces particular classes of decidable and semidecidable properties. What can be said about the properties that are decidable or semidecidable for *every* effective numbering of $\mathscr{C}$? The least we can say is that any property of total functions that is Markov decidable, i.e. decidable given an arbitrary index of the total function, is also decidable for every effective numbering of $\mathscr{C}$ (see next section for more information about Markov computability), and similarly for semidecidable properties. Does the converse also holds? In a sequel to this paper we will show that it does not: in reasonable subrecursive classes, there exists a property that cannot be semidecided from arbitrary indices, but is semidecidable in any effective total numbering of the class.

2.3 The whole class of total computable functions

By the Kreisel-Lacombe-Shœnfield/Ceitin theorem, the properties of total computable functions that are Markov decidable, i.e., decidable from indices coincide with the ones that are decidable from oracles, hence are generated by cylinders. However Friedberg showed that is it not the case for Markov semidecidable properties. Can we obtain a characterization of these properties as in the case of subrecursive classes?

We leave this problem open, but we show that the analog of Theorem 2.3 does not hold. We first introduce the analog of Definition 2.2. Here, φ_i is some Gödel numbering of the partial computable functions.

▶ **Definition 2.4.** If $v = (v_0, \ldots, v_n) \in \mathbb{N}^*$ then its ***complexity*** is

$$K(v) = \min\{i : \varphi_i \text{ extends } v\}.$$

Let h be a computable order. We define the set A_h of ***h-anticomplex*** functions as

$$A_h = \{f : \mathbb{N} \to \mathbb{N} : \forall n, K(f\restriction_n) \leq h(n)\}.$$

Again, the property $f \in A_h$ is semi-decidable (but this time not decidable) from any index of the total computable function f, but it is not semidecidable if f is given by an oracle. Observe that in the definition of $K(v)$, one considers all *partial* computable functions extending v. A direct analog of Definition 2.1 would be to consider *total* functions only. However the resulting anticomplexity property would not be semidecidable.

We now prove that the sets of anticomplex functions do not generate all the semidecidable properties.

▶ **Theorem 2.5.** *There is a semidecidable property of total computable functions that does not contain any non-empty set $[v] \cap A_h$.*

Proof. Let $t(j,i)$ be a partial computable function such that if φ_j is total then $t(j,i)$ is defined for all i. Define the set

$$B = \bigcap_i B_i \text{ where } B_i = \bigcup_{j \leq i} [\varphi_j\restriction_{t(j,i)}]$$

where $[\varphi_j\restriction_{t(j,i)}]$ is empty if $t(j,i)$ is not defined or φ_j is not defined on the first $t(j,i)$ inputs. The property $f \in B$ is semidecidable from indices of f. Indeed, if φ_i is total then $\varphi_i \in B \iff \varphi_i \in B_0 \cap \ldots \cap B_{i-1}$, which is semidecidable.

We now take $t(j,i)$ to be the halting time of $\varphi_j(i)$ plus $i+1$. We prove that the corresponding set B does not contain any non-empty set $[v] \cap A_h$. Let h be some computable order. We want to build a function f in $A_h \setminus B$, i.e. in $A_h \setminus B_b$ for some $b \in \mathbb{N}$.

▶ **Lemma 2.6.** *Given $a, b \in \mathbb{N}$ one can compute $m = m(a,b)$ such that if $[\varphi_a\restriction_m] \setminus B_b$ is non-empty then it contains some f such that $K(f) \leq h(m)$.*

Proof. The idea is simply that for $a, b, m \in \mathbb{N}$, if $[\varphi_a\restriction_m] \setminus B_b$ is non-empty then it contains a function whose complexity can be controlled. Indeed, while such a function cannot be effectively found as the set B_b is only enumerable, it becomes possible if some extra bits of information about B_b are provided.

Consider an algorithm that on inputs a, b, M and $p \leq b+1$ tries to find a set $L \subseteq \{0, \ldots, b\}$ of p elements such that for all $j \in L$, $\varphi_j\restriction_{t(j,b)}$ is defined, tests whether $[\varphi_a\restriction_M] \setminus \bigcup_{j \in L} [\varphi_j\restriction_{t(j,b)}]$ is non-empty and if it is so computes some total function f in that set (p is a guess about the number of cylinders in B_b).

The complexity of the output of the algorithm can be bounded in terms of the complexity of its inputs: there is a total computable function $m_0(a,b,e)$ such that if $M := \varphi_e(a,b)$ is defined and $p \leq b+1$ and the algorithm finds a function f, then $K(f) \leq m_0(a,b,e)$.

Now by the Recursion Theorem, there is e such that $\varphi_e(a,b) = \min\{m : h(m) \geq m_0(a,b,e)\}$. Let $m(a,b) = \varphi_e(a,b)$.

Applying the algorithm on inputs $a, b, m(a,b), p$ where p is the number of values of $j \leq b+1$ such that $\varphi_j\restriction_{t(j,b)}$ is defined (p is the "right guess") gives a function $f \in [\varphi_a\restriction_{m(a,b)}] \setminus B_b$ such that $K(f) \leq m_0(a,b,e) \leq h(m(a,b))$. ◀

We can make sure that $m(a,b) > b$ (in the proof above, take instead $\varphi_e(a,b) = \min\{m > b : h(m) \geq m_0(a,b,e)\}$).

▶ **Lemma 2.7.** *Let $a, b \in \mathbb{N}$ and $m = m(a,b)$. If $\varphi_a \in A_h$ and $[\varphi_a\restriction_m] \setminus B_b$ is non-empty then $[\varphi_a\restriction_m] \cap A_h \setminus B$ is non-empty.*

Proof. By Lemma 2.6 there exits $f \in [\varphi_a\restriction_m] \setminus B_b$ such that $K(f) \leq h(m)$. Of course, $f \notin B$ and we show that $f \in A_h$ i.e. that $K(f\restriction_i) \leq h(n)$ for all n.

For $n \leq m$, $K(f\restriction_n) = K(\varphi_a\restriction_n) \leq h(n)$ as $\varphi_a \in A_h$.

For $n \geq m$, $K(f\restriction_n) \leq K(f) \leq h(m) \leq h(n)$. ◀

▶ **Lemma 2.8.** *Let $v \in \mathbb{N}^*$ be such that $[v] \cap A_h \neq \emptyset$. There exist a, b such that $\varphi_a \in A_h$, $[\varphi_a\restriction_{m(a,b)}] \setminus B_b$ is non-empty and $[\varphi_a\restriction_{m(a,b)}] \subseteq [v]$.*

Proof. Define the computable function $b(a) = \min\{b \geq |v| : h(b) \geq a\}$. We now define a and will take $b := b(a)$.

Let $g \in [v] \cap A_h$. By the Recursion theorem, there is a such that

- For $i \neq b(a)$, $\varphi_a(i) = g(i)$,
- For $i = b(a)$, $\varphi_a(i)$ differs from each $\varphi_j(i)$ such that $j \leq b(a)$ and $\varphi_j(i)$ halts in at most $m(a, b(a))$ steps.

Let then $b = b(a)$.

▶ Claim 1. $\varphi_a \in A_h$, i.e. $K(\varphi_a\restriction_n) \leq h(n)$ for all n.

Indeed, for $n \leq b$ one has $K(\varphi_a\restriction_n) = K(g\restriction_n) \leq h(n)$ as $g \in A_h$. For $n > b$, $K(\varphi_a\restriction_n) \leq a \leq h(b) \leq h(n)$.

▶ Claim 2. Let $m = m(a,b)$. The set $[\varphi_a\restriction_m]$ is not contained in $B_b = \cup_{j \leq b}[\varphi_j\restriction_{t(j,b)}]$.

Indeed for each $j \leq b$:

- If $\varphi_j(b)$ halts in at most m steps then $\varphi_a(b) \neq \varphi_j(b)$ so $[\varphi_a\restriction_m]$ is disjoint from $[\varphi_j\restriction_{t(j,b)}]$ as both m and $t(j,b)$ are larger than b.
- If $\varphi_j(b)$ does not halt in at most m steps then $t(j,b)$ is either undefined or larger than m, so $\varphi_a\restriction_m$ does not contain $[\varphi_j\restriction_{t(j,b)}]$.

Finally, $[\varphi_a\restriction_{m(a,b)}] \subseteq [v]$ as $m(a,b) > b \geq |v|$ and $\varphi_a\restriction_{|v|} = g\restriction_{|v|} = v$. ◀

We can now conclude. If $[v] \cap A_h \neq \emptyset$ then applying Lemma 2.7 to a, b provided by Lemma 2.8 directly gives that $[v] \cap A_h \setminus B$ is non-empty, as it contains $[\varphi_a\restriction_{m(a,b)}] \cap A_h \setminus B$ which is non-empty. ◀

In other words, the complement of B is "so big" that its intersection with each A_h is dense in A_h.

We conjecture than there is no way of describing the semidecidable properties of total computable functions, using a parametrization by total computable functions. We say that a set W is extensional if when $\varphi_i = \varphi_j$ is total and $i \in W$, one has $j \in W$. An extensional

c.e. set W represents the semidecidable property $\{\varphi_i : i \in W \text{ and } \varphi_i \text{ is total}\}$. Let $\mathrm{Tot} = \{i : \varphi_i \text{ is total}\}$.

▶ **Conjecture.** There is no computable function $h : \mathrm{Tot} \to \mathbb{N}$ such that

- For all $i \in \mathrm{Tot}$, $W_{h(i)}$ is extensional,
- Every semidecidable property is represented by some $W_{h(i)}$ with $i \in \mathrm{Tot}$.

3 Black-box or oracle?

In computer science one often makes the distinction between accessing a program via its *code*, or as a *black-box*. For instance, this distinction appears naturally when validating or evaluating the correctness of a program, either by proving that its code is correct, or testing its outputs on a bunch of inputs, without looking at its code.

As for programs of every day life, looking at the code usually gives much more information than looking at its outputs. What about the general case of arbitrary programs, where information can be obfuscated? What is the difference between reading the code of a program and running it as a black-box? Does one obtain the same information about the function it computes? What additional information does the code of a program contains, compared to a black-box containing the program?

The results presented here (e.g., Proposition 2.5) and in [6] may be seen as answers to these questions. However, strictly speaking our results involve *oracles* more than *black-boxes*, the difference being that a black-box hides an actual program while no assumption is put on an arbitrary oracle. Does it make a difference? Does a black-box containing a program computing a function f give more information about f than an arbitrary oracle for f? For instance, could the particular halting times of the program (measurable from outside the black-box) be exploited in some way?

In this section we present a few results that are partial answers to these questions.

We first prove a result suggesting that a black-box does not give more information than an arbitrary oracle.

3.1 Observing a Turing machine

Here we prove that if the program is a Turing machine and that we can observe its execution, without knowing the complete transition table, we do not have more information than from an oracle giving the outputs of the machine.

Observing the execution of the machine means that at each step one can see the configuration of the machine, i.e. the contents of the tapes, the positions of the heads and the internal state. However, one may never know the complete transition table and the number of states. Equivalently, the observer progressively obtains the content of the transition table (at least its reachable part), but if the table is incomplete he may never know it entirely.

More formally, let us assume that the set Q of states of a Turing machine is a subset of $\mathbb{N}$, but is not known by the observer. Σ is some known finite alphabet. Instead of having access to the transition table δ as a finite function from $Q \times \Sigma$ to $Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$, the observer has access to δ as a partial function from $\mathbb{N} \times \Sigma \to \mathbb{N} \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$, defined on $Q \times \Sigma$ only. In particular no upper bound on the elements of Q is known.

▶ **Theorem 3.1.** *Let A be a set of total computable functions. The following are equivalent:*

1. *The problem $f \in A$ is semidecidable given an enumeration of a transition table of a Turing machine computing f,*
2. *The problem $f \in A$ is semidecidable given an oracle for f.*

Proof. The intuition is as follows. Assume that 1. holds. Given a total computable function f, there is a machine that outputs the same values as f on inputs $0, \ldots, n$ for some n, such that its transition table is accepted by the semidecision procedure and can be extended to the transition table of a machine computing g, for any g that coincides with f on $0, \ldots, n$. As a result, the cylinder $[f\restriction_n]$ is contained in A, which is open (and even effectively open).

Let $E \subseteq \mathbb{N}$ be a noncomputable c.e. set. The following claim is obvious.

▶ Claim 3. Given i, one can effectively build a machine M_i such that on input n, $M_i(n)$ halts on the same configuration as the initial one (in particular its input tape contains n), except that its state is q_1 if i is enumerated in E by stage n, q_0 otherwise. q_0 and q_1 are never reached before and there is no transition from these states.

Let N be a Turing machine with initial state q_0, all the other states being fresh (no common state with the machines M_i). Think of N as computing a total function f, but we do not need to assume that N is total.

▶ Claim 4. For each i one can effectively build a Turing machine, denoted $N \circ M_i$, such that $N \circ M_i(n)$ reaches q_1 if and only if i is enumerated in E by stage n, and $N \circ M_i$ computes the same function as N if $i \notin E$.

Proof. Given i, taking the union of the transition tables of N and M_i, with the initial state of M_i as initial state, one gets a Turing machine $N \circ M_i$ which first behaves as M_i, and then if $i \notin E$ behaves as N. ◀

We now prove that A is the intersection of an effective open set U with the class of total computable functions, which is equivalent to 2. in the statement of Theorem 3.1. For each machine N, look for $i \in E$ such that an enumeration of $N \circ M_i$ is accepted by the semidecision procedure. Compute n_0 such that i is enumerated in E at stage n_0. If N is defined on inputs $0, \ldots, n_0 - 1$, with output values $v_0, \ldots, v_{n_0-1}$ respectively then enumerate the cylinder $[v_0, \ldots, v_{n_0-1}]$ in U.

▶ Claim 5. A is contained in U.

Proof. Let $f \in A$ and N be a machine computing f. When $i \notin E$, $N \circ M_i$ computes f so any enumeration of its transition table is accepted by the semidecision procedure. As E is not computable, there must exist $i \in E$ such that an enumeration of the table of $N \circ M_i$ is also accepted. Let n_0 be such that i is enumerated in E at stage n_0. $N(n)$ is defined for every $n < n_0$ and outputs $f(n)$, so the cylinder enumerated in U is $[f\restriction_{n_0}]$. ◀

▶ Claim 6. Conversely, every computable function in U belongs to A.

Proof. Let $[v_0, \ldots, v_{n_0-1}]$ be a cylinder enumerated in U. On inputs $n < n_0$, $N \circ M_i(n)$ outputs v_n, never reaching state q_1. On inputs $n \geq n_0$, $N \circ M_i$ ends in state q_1.

Let $g \in [v_0, \ldots, v_{n_0-1}]$ and M_g be a machine computing g with initial state q_1, all the other states being fresh. Taking the union of the transition tables of $N \circ M_i$ and M_g, with the initial state of M_i as initial state, one gets a machine M'_g computing g. The enumeration of the table of $N \circ M_i$, accepted by the semidecision procedure, can be extended to an enumeration of the table of M'_g, which is then also accepted (the semidecision procedure halts before being able to distinguish between $N \circ M_i$ and M'_g). As a result, $g \in A$. ◀

The proof is complete: given f by an oracle, evaluate it successively on all inputs and look for a cylinder of U containing f. ◀

3.2 A difference between a black-box and an oracle

We now exhibit a difference between having a program in a black-box and an oracle.

Instead of deciding or semi-deciding properties, a usual task it to compute a function. In [6] it is proved that

▶ **Theorem 3.2** ([6]). *Let $F : X_c \to Y$ where X, Y are effective topological spaces and X_c is the set of computable elements of X. The following statements are equivalent:*

- *There is a Turing machine that computes $F(x)$ given any index of x as input,*
- *There is a Turing machine that computes $F(x)$ given any name for x and any $k \geq K(x)$ as input.*

We do not insist on the notion of effective topological space, which is essentially a topological space with a countable basis. The classes of partial computable functions or total computable functions are examples of effective topological spaces. A name for an element x is an infinite binary string encoding x in some canonical way, which we do not describe here (the interested reader may consult [13]).

The assumption about effective topological spaces is essential as there is a non-effective topological space Y for which the result fails, which is the class $\mathcal{O}(\mathbb{N}^{\mathbb{N}})$ of open subsets of $\mathbb{N}^{\mathbb{N}}$ (which is not countably-based for the appropriate topology). Here we take for X the class $\mathcal{P}$ of partial computable functions.

▶ **Theorem 3.3** ([6]). *There is a functional $F : \mathcal{P} \to \mathcal{O}(\mathbb{N}^{\mathbb{N}})$ such that:*

- *There is a Turing machine that computes $F(\varphi)$ given any index of φ as input,*
- *There is <u>no</u> Turing machine that computes $F(\varphi)$ given any name for φ and any $k \geq K(\varphi)$ as input.*

Computing an element of $\mathcal{O}(\mathbb{N}^{\mathbb{N}})$, i.e. and open set $U \subseteq \mathbb{N}^{\mathbb{N}}$, consists in enumerating a list of finite words $v_i \in \mathbb{N}^*$ such that U is the corresponding union of cylinders $\bigcup_i [v_i]$. A name for a partial function φ is an infinite binary sequence such that $\varphi(m) = n$ if and only if the block $01^{\langle n,m \rangle}0$ appears in the sequence ($\langle ., . \rangle$ is a computable bijection between $\mathbb{N}^2$ and $\mathbb{N}$).

Contrasting with Theorem 3.3 we now show that

▶ **Theorem 3.4.** *Let $F : \mathcal{P} \to \mathcal{O}(\mathbb{N}^{\mathbb{N}})$. The following statements are equivalent:*

- *There is a Turing machine that computes $F(\varphi)$ given any index of φ as input,*
- *There is a Turing machine that computes $F(\varphi)$ given an access to a black-box containing a program computing φ, and any upper bound onthe size of the program.*

The difference with the previous theorem is that:

- Contrary to an oracle producing a name, a black-box contains an actual program, with its particular halting times,
- For a particular program p computing a function φ, an upper bound on the size of p is always an upper bound on $K(\varphi)$ (the size of the shortest program computing φ), but not the converse. In particular the theorem fails if an upper bound on $K(\varphi)$ rather than the size of $|p|$ is provided.

Proof idea. The argument is essentially the same as in the proof of Proposition 2.5. The idea is that the observation of the black-box can be seen as a total computable function that, given an input and a time, tells whether the program on that input halts in that time.

Assume that F is computable from indices. Let φ be given by a black-box and k an upper bound on the size of the program in the black-box.

At each stage we will have a finite list L of programs, such that the program in the black-box belongs to this list. At the beginning, L is the set of programs whose size is bounded

by k. We enumerate the intersection of the open sets $F(\varphi_i)$ for all $i \in L$. From time to time we may remove an element of L that we know is not the program in the black-box. Each time we change L, we restart the enumeration of the intersection of the open sets $F(\varphi_i)$ for all i in the new list L (a larger open set). Eventually the list L will contain only programs computing the actual function φ, so we will enumerate the right open set.

We now explain how we progressively remove programs from L. For each program (in the list or in the black-box), each input n and each number t, we can decide whether the program halts in t steps on input n. If for some n and t a program is inconsistent with the black-box (one halts in t steps on input n but not the other), then the program can be rejected. If for some n the program and the black-box both halt on n giving different outputs, we can also reject the program.

Observe that we do not really need to have a precise measure of the halting time of the black-box: if we know that the actual halting time t of a program and the measured halting time $\tilde{t}$ are related by $|t - \tilde{t}| \leq 10$, or $t/2 \leq \tilde{t} \leq 2t$ for instance, then we only rejects programs that do not respect this gap w.r.t. the black-box. ◀

4 Two remarks on The Intensional Content of Rice's Theorem

In this paper we have investigated the properties of functions that are semidecidable, when the function is presented as a program computing it. Such a property can be alternatively seen as a c.e. set of programs that is *extensional*, in the sense that two equivalent programs – two programs computing the same function – are both in the set or both outside the set.

Asperti [1] investigates the case when extensionality is understood in a weaker sense, i.e. for a stronger notion of equivalence: two programs are equivalent if they compute the same function *and* have similar complexities (running time, or space, more generally any measure of complexity in Blum's sense). Such classes of programs are called Complexity Cliques.

It is proved in [1] that under certain assumptions on the measure of complexity (which should "behave well" w.r.t. the s-m-n function, composition and parallel computation),

▶ **Theorem 4.1** (Asperti [1]). *No Complexity Clique of total functions and containing programs with non constant complexity can be c.e.*

It is asked in [1] whether the assumption about non-constant complexity is needed.

We make the simple observation that it is indeed necessary, because the set of Turing machines with constant time complexity is a c.e. Complexity Clique. Indeed, given $c \in \mathbb{N}$, it is decidable whether a given Turing machine always halts in c steps, because one only has to evaluate it on inputs of size at most $c + 1$, during c steps. Hence it is semidecidable whether a Turing machine has constant time complexity, by trying every possible c.

We also observe that the assumptions about the measure of complexity cannot be dropped either, as the class of one-tape Turing machines that run in linear time is a c.e. Complexity Clique. Indeed, it was recently proved by Gajser [5] that for each $c \in \mathbb{N}$, whether a one-tape Turing machine halts in time cn on inputs of size n is decidable (his result is more general as it applies to a larger class of time bounds in $o(n \log n)$). It gives an indirect proof that one-tape Turing machines and their running time do not satisfy the assumptions of [1].

Acknowledgements We thank the anonymous referees for their useful comments that helped improving this article, and for suggesting the question at the end of Section 2.2.

References

1. Andrea Asperti. The intensional content of rice's theorem. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 113–119. ACM, 2008.
2. G. S. Ceitin. Algorithmic operators in constructive metric spaces. *Trudy Matematiki Instituta Steklov*, 67:295–361, 1962. English translation: *American Mathematical Society Translations*, series 2, 64:1-80, 1967.
3. Johanna N. Y. Franklin, Noam Greenberg, Frank Stephan, and Guohua Wu. Anti-complex sets and reducibilities with tiny use. *J. Symb. Log.*, 78(4):1307–1327, 2013.
4. Richard M. Friedberg. Un contre-exemple relatif aux fonctionnelles récursives. *Comptes Rendus de l'Académie des Sciences*, 247:852–854, 1958.
5. David Gajser. Verifying time complexity of turing machines. *Theoretical Computer Science*, 600:86 – 97, 2015.
6. Mathieu Hoyrup and Cristóbal Rojas. On the information carried by programs about the objects they compute. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 447–459. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
7. Dexter Kozen. Indexings of subrecursive classes. *Theoretical Computer Science*, 11(3):277 – 301, 1980.
8. Georg Kreisel, Daniel Lacombe, and Joseph R. Schœnfield. Fonctionnelles récursivement définissables et fonctionnelles récursives. *Comptes Rendus de l'Académie des Sciences*, 245:399–402, 1957.
9. Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.
10. Albert R. Meyer and Dennis M. Ritchie. The complexity of loop programs. In *Proceedings of the 1967 22Nd National Conference*, ACM '67, pages 465–469, New York, NY, USA, 1967. ACM.
11. Henry G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):pp. 358–366, 1953.
12. Norman Shapiro. Degrees of computability. *Transactions of the American Mathematical Society*, 82:281–299, 1956.
13. Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.