Safe recursion over an arbitrary structure
PAR, PH and DPH

Bournez, Olivier; Cucker, Felipe; De Naurois, Paulin Jacobé; Marion, Jean-Yves

Published in:
Electronic Notes in Theoretical Computer Science

Published: 28/11/2003

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

Published version (DOI):
10.1016/S1571-0661(03)00005-7

Publication details:
Bournez, O., Cucker, F., De Naurois, P. J., & Marion, J-Y. (2003). Safe recursion over an arbitrary structure: PAR, PH and DPH. *Electronic Notes in Theoretical Computer Science*, *90*, 3-14. https://doi.org/10.1016/S1571-0661(03)00005-7

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 90 (2003) 3–14
www.elsevier.com/locate/entcs

Safe Recursion Over an Arbitrary Structure: PAR, PH and DPH

Olivier Bournez[a], Felipe Cucker[b,1], Paulin Jacobé de Naurois[a,1] and Jean-Yves Marion[a]

[a] *LORIA/INRIA, 615 rue du Jardin Botanique, BP 101, 54602 Villers-lès-Nancy Cedex, Nancy, France*
email: {bournez, denauroi, marionjy}@loria.fr

[b] *Department of Mathematics, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, HONG KONG*
e-mail: macucker@math.cityu.edu.hk

Abstract

Considering the Blum, Shub, and Smale computational model for real numbers, extended by Poizat to general structures, classical complexity can be considered as the restriction to finite structures of a more general notion of computability and complexity working over arbitrary structures.
In a previous paper, we showed that the machine-independent characterization of Bellantoni and Cook of sequential polynomial time for classical complexity is actually the restriction to finite structures of a characterization of sequential polynomial time over arbitrary structures.
In this paper, we prove that the same phenomenon happens for several other complexity classes: over arbitrary structures, parallel polynomial time corresponds to safe recursion with substitutions, and the polynomial hierarchy corresponds to safe recursion with predicative minimization.
Our results yield machine-independent characterizations of several complexity classes subsuming previous ones when restricted to finite structures.

Keywords: Complexity, computability, sequential polynomial time, parallel polynomial time

1 Introduction

Classical computability and complexity can be considered as a restricted case of a more general notion of computability and complexity over arbitrary structures. Indeed, considering the notion of computation introduced by Blum,

[1] Partially supported by City University of Hong Kong SRG grant 7001290.

doi:10.1016/S1571-0661(03)00005-7

Shub and Smale in [4] for computations over the real numbers, and extended by Poizat in [11,22] for general structures, computations over finite structures correspond to classical computability and complexity, whereas computations over non-finite structures, such as real or complex numbers, give birth to new complexity classes and results, and provide new insights for understanding complexity and computability theory [3].

To understand computability in a whole perspective, in the spirit of the monographs [3,22], it might be important to understand which results are specific to classical complexity and which one are special cases of results working over arbitrary logical structures. In particular, it might be important to understand whether machine independent characterizations of complexity classes exist for computability over arbitrary structures.

In last year ICC workshop [6], we proved that safe primitive recursion principle of Bellantoni and Cook [2] for characterizing sequential polynomial time is actually valid over arbitrary structures.

This paper is a contribution to the natural following steps: understand whether other complexity classes can be characterized implicitly in a same spirit.

First, based on Leivant and Marion in [19], we characterize parallel polynomial time (the results presented in [6] and this result are also presented in [5]).

Theorem 1.1 *Over any structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$*, the set of functions computable with safe recursion with substitution over* $\mathcal{K}$ *is the set of functions computed in parallel polynomial time over* $\mathcal{K}$ *of polynomial output size.*

Observe that, unlike Leivant and Marion, Theorem 1.1 characterizes parallel polynomial time and not polynomial space: for classical complexity both classes correspond. However over arbitrary structures, this is not true, since the notion of working space may be meaningless: as pointed out by Michaux [21], on some structures like $(\mathbb{R}, +, -, *, \leq, \mathbf{0}, \mathbf{1})$, any computable function can be computed in constant working space.

Second, based on [1], we characterize problems of the polynomial hierarchy.

One difficulty is that over an arbitrary structure, two kinds of nondeterminism may be considered according to whether the witness is allowed to be an arbitrary element of the structure or is restricted to be in $\{0, 1\}$ [4,3]. The latter is usually called *digital* and a letter D is used to denote complexity classes arising from the use of digital nondeterminism. Note that in classical complexity theory, over a finite structure, these two notions of nondeterminism coincide and they yield the same polynomial hierarchy.

We prove:

Theorem 1.2 *Over any structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, *the set of decision problems computable in the polynomial hierarchy over* $\mathcal{K}$ *corresponds to safe recursion with predicative minimization.*

Theorem 1.3 *Over any structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, *the set of decision problems computable in the digital polynomial hierarchy over* $\mathcal{K}$ *corresponds to safe recursion with digital predicative minimization.*

This work stems from the notion of safe recursion introduced by Bellantoni and Cook in [2]. Other machine independent characterizations of complexity classes have been obtained in classical complexity. Such characterizations include descriptive characterization based on finite model theory like Fagin [10], characterization by function algebra like [8], or by combining both kinds of characterization like in [14,23]: see [7,16,9] for more complete references. One benefit of the approach of Bellantoni and Cook, and the alternative approach of Leivant [18] by mean of data tiering, is that they do not require explicit upper bounds on computational resources or restrictions on the growth rates.

When considering computations over arbitrary structures, machine independent characterizations of several complexity classes inspired by finite model theory have already been obtained [3,12,13]. They basically require over arbitrary structure to distinguish two (not so natural) types of functions (called "number terms" and "index terms" in [13]) in order to be able to use finiteness considerations over the models even in presence of infinite underlying domains like the field of real numbers. We believe our approach to give nicer machine-independent characterizations of complexity classes.

From a programming perspective, a way of understanding all our results is to see computability over arbitrary structures like a programming language with extra operators which come from some external libraries. This observation, and its potential to build methods to automatically derive computational properties of programs, along the lines of [15,17,20], is a main motivation of our work.

This paper is organized as follows. In Section 2, we recall some of our results in [6]. In Section 3, we characterize parallel polynomial time. In Section 4, we characterize the polynomial hierarchy. In Section 5, we characterize the digital polynomial hierarchy. Section 6 is a conclusion.

2 Preliminaries

We assume that the reader has some familiarities with the BSS model of computation. Detailed accounts can be found in [3] —for structures like real and

complex numbers— or [22] —for considerations about more general structures.

Definition 2.1 A *structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$ is given by some underlying set $\mathbb{K}$, a family of operators op_i with arities, and a finite number of relations $rel_1, \ldots, rel_l$ with arities. Constants correspond to operators of arity 0. While the index set I may be infinite, the number of operators with arity greater than 1 needs to be finite. We will not distinguish between operator and relation symbols and their corresponding interpretations as functions and relations respectively over the underlying set $\mathbb{K}$. We assume that the equality relation $=$ is a relation of the structure, and that there are at least two constant symbols, with different interpretations (denoted by $\mathbf{0}$ and $\mathbf{1}$ in our work) in the structure.

An example of structure is $\mathcal{K} = (\mathbb{R}, +, -, *, =, \leq, \{\mathbf{c}_r\}_{r\in\mathbb{R}})$. Another example, corresponding to classical complexity and computability theory is $\mathcal{K} = (\{0,1\}, \vee, \wedge, =, \mathbf{0}, \mathbf{1})$.

We denote by $\mathbb{K}^* = \bigcup_{i\in\mathbb{N}} \mathbb{K}^i$ the set of words over the alphabet $\mathbb{K}$. The space $\mathbb{K}^*$ is the analogue to Σ^* the set of all finite sequences of zeros and ones.

In what follows, words of elements in $\mathbb{K}$ will be represented with overlined letters, while elements in $\mathbb{K}$ will be represented by letters. For instance, $a.\overline{x}$ stands for the word in $\mathbb{K}^*$ whose first letter is a and which ends with the word $\overline{x}$. We denote by ϵ the empty word. The length of a word $\overline{w} \in \mathbb{K}^*$ is denoted by $|\overline{w}|$.

Let us first recall our notion of safe recursion, as found in [6], extending the notion of safe recursive functions over the natural numbers found in [2].

In the spirit of [2], safe recursive functions have two different types of arguments, each of them having different properties and purposes. The first type of argument, called *normal*, can be used to make basic computation steps or to control recursion. The second type of argument, called *safe*, can not be used to control recursion. This distinction between safe and normal arguments ensures that safe recursive functions can be computed in polynomial time.

To emphasize the distinction between normal and safe variables we will write $f : N \times S \to R$ where N indicates the domain of the normal arguments and S that of the safe arguments. If all the arguments of f are of one kind, say safe, we will write $\emptyset$ in the place of N. Also, if $\bar{x}$ and $\bar{y}$ are these arguments, we will write $f(\bar{x}; \bar{y})$ separating them by a semicolon ";". Normal arguments are placed at the left of the semicolon and safe arguments at its right.

Definition 2.2 The set of *safe recursive functions over* $\mathbb{K}$ is the smallest set of functions $f : (\mathbb{K}^*)^k \times (\mathbb{K}^*)^l \to \mathbb{K}^*$, for some k and l, containing the basic safe functions, and closed under the operations of safe composition and safe

recursion.

Our basic safe functions are of four kinds:

(i) *Functions making elementary manipulations of words of elements in* $\mathbb{K}$:

$$\begin{array}{ccc} \mathrm{hd}(;a.\overline{x}) = a & \mathrm{tl}(;a.\overline{x}) = \overline{x} & \mathrm{cons}(;a.\overline{x_1},\overline{x_2}) = a.\overline{x_2} \\ \mathrm{hd}(;\epsilon) = \epsilon & \mathrm{tl}(;\epsilon) = \epsilon & \mathrm{cons}(;\epsilon,\overline{x_2}) = \overline{x_2} \end{array}$$

(ii) *Projections*: For any $n \in \mathbb{N}$, $i \leq n$

$$\overset{n}{\underset{i}{\mathrm{Pr}}}(;\overline{x_1},\ldots,\overline{x_i},\ldots,\overline{x_n}) = \overline{x_i}.$$

(iii) *Functions of structure* $\mathcal{K}$: for any operator (including the constants treated as operators of arity 0) op_ι or relation rel_ι of arity n_ι we have

$$\mathrm{Op}_\iota(;a_1.\overline{x_1},\ldots,a_{n_\iota}.\overline{x_{n_\iota}}) = (op_\iota(a_1,\ldots,a_{n_\iota})).\overline{x_{n_\iota}}$$

$$\mathrm{Rel}_\iota(;a_1.\overline{x_1},\ldots,a_{n_\iota}.\overline{x_{n_\iota}}) = \begin{cases} \mathbf{1} \text{ if } rel_\iota(a_1,\ldots,a_{n_\iota}) \\ \epsilon \text{ otherwise} \end{cases}$$

(iv) *Test function* :

$$\mathsf{Test}(;\overline{x},\overline{y},\overline{z}) = \begin{cases} \overline{y} & \text{if } \mathrm{hd}(\overline{x}) = \mathbf{1} \\ \overline{z} & \text{otherwise} \end{cases}$$

The operations mentioned above are defined as follows.

(1) *Safe composition.* Assume $g : (\mathbb{K}^*)^m \times (\mathbb{K}^*)^n \to \mathbb{K}^*$, $h_1,\ldots,h_m : \mathbb{K}^* \times \emptyset \to \mathbb{K}^*$ and $h_{m+1},\ldots,h_{m+n} : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ are given safe recursive functions. Then their safe composition is the function $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ defined by

$$f(\overline{x};\overline{y}) = g\left(h_1(\overline{x};),\ldots,h_m(\overline{x};);h_{m+1}(\overline{x};\overline{y}),\ldots,h_{m+n}(\overline{x};\overline{y})\right).$$

(2) *Safe recursion.* Assume $h_1,\ldots,h_k : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ and $g_1,\ldots,g_k : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^{k+1} \to \mathbb{K}^*$ are given functions. Functions $f_1,\ldots,f_k : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ can then be defined by safe recursion:

$$f_1(\epsilon,\overline{x};\overline{y}),\ldots,f_k(\epsilon,\overline{x};\overline{y}) = h_1(\overline{x};\overline{y}),\ldots,h_k(\overline{x};\overline{y})$$

$$f_1(a.\overline{z},\overline{x};\overline{y}) = \begin{cases} g_1(\overline{z},\overline{x};f_1(\overline{z},\overline{x};\overline{y}),\ldots,f_k(\overline{z},\overline{x};\overline{y}),\overline{y}) & \text{if } \forall i\ f_i(\overline{z},\overline{x};\overline{y}) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

$$\vdots$$

$$f_k(a.\overline{z},\overline{x};\overline{y}) = \begin{cases} g_k(\overline{z},\overline{x};f_1(\overline{z},\overline{x};\overline{y}),\ldots,f_k(\overline{z},\overline{x};\overline{y}),\overline{y}) & \text{if } \forall i\ f_i(\overline{z},\overline{x};\overline{y}) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

In [6], we proved:

Proposition 2.3 *Over any structure $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, the set of safe recursive functions over $\mathcal{K}$ is exactly the set of functions computed in polynomial time by a BSS machine over $\mathcal{K}$.*

3 A Characterization of the Parallel Class $\mathrm{FPAR}_{\mathcal{K}}$

3.1 A Parallel Model of Computation

Recall the notion of circuit over an arbitrary structure $\mathcal{K}$ [3,22].

Definition 3.1 A circuit over the structure $\mathcal{K}$ is an acyclic directed graph whose nodes, called *gates*, are labeled either as *input* gates of in-degree 0, *output* gates of out-degree 0, *test* gates of in-degree 3, or by a relation or an operation of the structure, of in-degree equal to its arity.

The evaluation of a circuit on a given assignment of values of $\mathbb{K}$ to its input gates is defined in a straightforward way, all gates behaving as one would expect. We just note that any test gate tests whether its first parent is labeled with $\mathbf{1}$, and returns the label of its second parent if this is true or the label of its third parent if not. This evaluation defines a function from $\mathbb{K}^n$ to $\mathbb{K}^m$ where n is the number of input gates and m that of output gates. See [22,3] for formal details.

We say that a family $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ of circuits computes a function $f : \mathbb{K}^* \to \mathbb{K}^*$ when the function computed by the nth circuit of the family is the restriction of f to $\mathbb{K}^n$. We say that this family is *P-uniform* when there exists a deterministic machine computing a description of the ith gate of the nth circuit in time polynomial in n.

Definition 3.2 $\mathrm{FPAR}_{\mathcal{K}}$ (resp. $\mathrm{PAR}_{\mathcal{K}}$) is the class of functions computable (resp. problems decidable) by P-uniform families of circuits of polynomial depth and such that $|f(\bar{x})| = |\bar{x}|^{\mathcal{O}(1)}$ for all $\bar{x} \in \mathbb{K}^*$.

It can be shown that these notions of parallel polynomial time correspond indeed to functions of polynomial output size and to problems which can be computed in polynomial time by natural notions of parallel machine over a structure $\mathcal{K}$: see [3].

The goal of this section is to prove that, over any structure $\mathcal{K}$, $\mathrm{FPAR}_{\mathcal{K}}$ also corresponds to the class of functions computable with safe recursion with

substitutions.

3.2 Safe Recursion with Substitutions

Definition 3.3 The set of functions defined with *safe recursion with substitutions* over $\mathcal{K}$ is the smallest set of functions $f : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to \mathbb{K}^*$, containing the basic safe functions, and closed under safe composition and the following operation:

Safe recursion with substitution. Let $h_1, \ldots, h_k : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, $g_1, \ldots, g_k : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^{kl+1} \to \mathbb{K}^*$, and $\sigma_{ij} : \emptyset \times \mathbb{K}^* \to \mathbb{K}^*$ for $0 < i \leq k$ and $0 < j \leq l$ be safe recursive functions. The functions $f_1, \ldots, f_k : (\mathbb{K})^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ are defined by safe recursion with substitutions as follows:

$$f_1(\epsilon, \overline{x}; \overline{u}, \overline{y}), \ldots, f_k(\epsilon, \overline{x}; \overline{u}, \overline{y}) = h_1(\overline{x}; \overline{u}, \overline{y}), \ldots, h_k(\overline{x}; \overline{u}, \overline{y})$$

and

$$f_1(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \begin{cases} g_1\left(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \sigma_{11}(;\overline{u}), \overline{y}), \ldots, f_1(\overline{z}, \overline{x}; \sigma_{1l}(;\overline{u}), \overline{y}), \ldots\right. \\ \qquad \left. f_k(\overline{z}, \overline{x}; \sigma_{k1}(;\overline{u}), \overline{y}), \ldots, f_k(\overline{z}, \overline{x}; \sigma_{kl}(;\overline{u}), \overline{y}), \overline{y}\right) \\ \qquad \text{if } \forall i, j \ f_i(\overline{z}, \overline{x}; \sigma_{ij}(;\overline{u}), \overline{y}) \neq \perp \\ \perp \qquad \text{otherwise} \end{cases}$$

$$\vdots$$

$$f_k(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \begin{cases} g_k\left(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \sigma_{11}(;\overline{u}), \overline{y}), \ldots, f_1(\overline{z}, \overline{x}; \sigma_{1l}(;\overline{u}), \overline{y}), \ldots\right. \\ \qquad \left. f_k(\overline{z}, \overline{x}; \sigma_{k1}(;\overline{u}), \overline{y}), \ldots, f_k(\overline{z}, \overline{x}; \sigma_{kl}(;\overline{u}), \overline{y}), \overline{y}\right) \\ \qquad \text{if } \forall i, j \ f_i(\overline{x}, \overline{z}; \sigma_{ij}(;\overline{u}), \overline{y}) \neq \perp \\ \perp \qquad \text{otherwise.} \end{cases}$$

The functions σ_{ij} are called *substitution functions.*

Theorem 3.1 *Over any structure $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, the set of functions definable with safe recursion with substitution over $\mathcal{K}$ is exactly* $\mathrm{FPAR}_{\mathcal{K}}$.

An immediate corollary is the following:

Corollary 3.2 *Over any structure $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, the set of decision functions definable with safe recursion with substitution over $\mathcal{K}$ is exactly the set of decision functions computed in parallel polynomial time over $\mathcal{K}$.*

In the classical setting (see [19]), safe recursion with substitution characterizes the class FPSPACE. However, in the general setting, this notion of working space is meaningless, as pointed in [21]: on some structures like $(\mathbb{R}, 0, 1, =, +, -, *)$, any computation can be done in constant working space. However, since in the classical setting we have FPAR = FPSPACE, our result extends the classical one from [19].

4 A Characterization of $\mathrm{PH}_{\mathcal{K}}$

4.1 Polynomial hierarchy over a structure $\mathcal{K}$

As for the classical settings, the polynomial hierarchy over a given structure $\mathcal{K}$ can be defined in several equivalent ways, including logical descriptions, or definitions by successive relativizations of non-deterministic polynomial time: see [3].

Focusing on a logical point of view, we have:

Definition 4.1 The polynomial time hierarchy over $\mathcal{K}$ is $\mathrm{PH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \Sigma_{\mathcal{K}}^i$, where $\Sigma_{\mathcal{K}}^i$ is the class of decision problems defined in the following way: $S \in \Sigma_{\mathcal{K}}^i$ if and only if

- there exists a polynomial time BSS machine M_p over $\mathcal{K}$
- there exist polynomial functions $p_1, \ldots, p_i : \mathbb{N} \to \mathbb{N}$
- $\overline{x} \in S \Leftrightarrow \exists \overline{y_1} \in \mathbb{K}^{\leq p_1(|x|)} \forall \overline{y_2} \in \mathbb{K}^{\leq p_2(|x|)} \ldots \mathrm{Q} \overline{y_i} \in \mathbb{K}^{\leq p_i(|x|)}$, M_p accepts $(\overline{x}, \overline{y_1}, \ldots, \overline{y_i})$
  where $\mathbb{K}^{\leq n} = \{x \in \mathbb{K}^* \mid |x| \leq n\}$, $\mathrm{Q} = \exists$ if i is odd and $\mathrm{Q} = \forall$ otherwise.

The class $\Pi_{\mathcal{K}}^i$ is defined replacing the third line above by $\overline{x} \in S \Leftrightarrow \forall \overline{y_1} \in \mathbb{K}^{\leq p_1(|x|)} \exists \overline{y_2} \in \mathbb{K}^{\leq p_2(|x|)} \ldots \mathrm{Q} \overline{y_i} \in \mathbb{K}^{\leq p_i(|x|)}$, M_p accepts $(\overline{x}, \overline{y_1}, \ldots, \overline{y_i})$.

In addition, we define $\square_{\mathcal{K}}^i = \Sigma_{\mathcal{K}}^i \cup \Pi_{\mathcal{K}}^i$.

Clearly the *polynomial hierarchy* satisfies $\mathrm{PH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \Sigma_{\mathcal{K}}^i = \bigcup_{i=0}^{\infty} \Pi_{\mathcal{K}}^i = \bigcup_{i=0}^{\infty} \square_{\mathcal{K}}^i$.

The first levels of this hierarchy are known: $\Sigma_{\mathcal{K}}^0 = \Pi_{\mathcal{K}}^0 = \mathrm{P}_{\mathcal{K}}$, $\Sigma_{\mathcal{K}}^1 = \mathrm{NP}_{\mathcal{K}}$ and $\Pi_{\mathcal{K}}^1 = \mathrm{coNP}_{\mathcal{K}}$.

4.2 Safe Recursion with Predicative Minimization

Based on [1], we now introduce the notion of predicative minimization.

Definition 4.2 Given $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, we define $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}$ by *predicative minimization*:

$$f(\overline{x};\overline{a}) = \mu\overline{b}(h(\overline{x};\overline{a},\overline{b})) = \begin{cases} \mathrm{cons}(;\mathbf{1},\min_t\{\overline{b}:\ |\overline{b}| = t \wedge h(\overline{x};\overline{a},\overline{b}) = \epsilon\}) \\ \qquad\qquad \text{if there is such a } \overline{b} \in \mathbb{K}^* \\ \epsilon \qquad\qquad \text{otherwise.} \end{cases}$$

Remark 4.3 *This minimization is a non-deterministic one, as opposed to the minimization defined in [6]. It is used to capture one (possibly among several) existential witness for some property. Therefore, it can capture languages recognized by BSS-machines with different levels of quantifier alternation. This notion is in essence non-deterministic and may well, on a structure without quantifier elimination, be not computable by BSS machines without quantifier alternation. On the other hand, the minimization of [6] is a deterministic one: it can not capture any quantification, but can denote the computation length needed to reach some node in a BSS machine, hence capturing the whole class of BSS computable functions (without quantifier).*

Remark 4.4 *As stated in [1], it may seem surprising to obtain* $\mathbf{1}$ *if no minimal* $\overline{b}$ *exists. When we deal with partial recursive functions, the operation of minimization is a non-halting process when no minimal solution exists. Here however, since we use the operation of safe recursion instead of primitive recursion, we know, as proved in [6], that, when* h *is a safe recursive function, the computation time of* $h(\overline{x};\overline{a},\overline{b})$ *does not depend on* $\overline{b}$*. Therefore, on a structure with quantifier elimination, this predicative minimization is computable in finite time.*

We now introduce a new set of functions.

Definition 4.5 The set $\mu\mathrm{PH}_{\mathcal{K}}$ of functions is the closure of the set of safe recursive functions over $\mathcal{K}$ under the application of predicative minimization and safe composition.

In order to give a rigorous formalism for our statement below, let us define the following:

Definition 4.6 For any function $f : (\mathbb{K}^*)^n \times (\mathbb{K}^*)^m \to \mathbb{K}^*$,

$$\mathrm{Char}_f(\overline{x_1},\ldots,\overline{x_n};\overline{y_1},\ldots,\overline{y_m}) = \begin{cases} \mathbf{1} \ \text{ if } f(\overline{x_1},\ldots,\overline{x_n};\overline{y_1},\ldots,\overline{y_m}) \neq \epsilon \\ \epsilon \qquad\qquad \text{otherwise} \end{cases}$$

We denote by $\mathrm{CharPH}_{\mathcal{K}}$ the set $\{\mathrm{Char}_f,\ f \in \mu\mathrm{PH}_{\mathcal{K}}\}$. Theorem 1.2 is

formally the following:

Theorem 4.7 *A decision problem over $\mathcal{K}$ belongs to* $\mathrm{PH}_{\mathcal{K}}$ *if and only if its characteristic function belongs to* $\mathrm{CharPH}_{\mathcal{K}}$.

5 A Characterization of $\mathrm{DPH}_{\mathcal{K}}$

5.1 Digital Polynomial hierarchy over a structure $\mathcal{K}$

In this section, we assume that our structure $\mathcal{K}$ contains at least two constants, denoted by $\mathbf{0}$ and $\mathbf{1}$. Then we can define the notion of digital polynomial hierarchy over $\mathcal{K}$:

Definition 5.1 The digital polynomial time hierarchy over $\mathcal{K}$ is $\mathrm{DPH}_{\mathcal{K}} = \bigcup_{\imath=0}^{\infty} \mathrm{D}\Sigma_{\mathcal{K}}^{\imath}$, where $\mathrm{D}\Sigma_{\mathcal{K}}^{\imath}$ is the class of problems defined in the following way: $S \in \mathrm{D}\Sigma_{\mathcal{K}}^{\imath}$ if and only if

- there exists a polynomial time BSS machine M_p over $\mathcal{K}$
- there exist polynomial functions $p_1, \ldots, p_\imath : \mathbb{N} \to \mathbb{N}$
- $\overline{x} \in S \Leftrightarrow \exists \overline{y_1} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_1(|x|)} \forall \overline{y_2} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_2(|x|)} \ldots \mathrm{Q}\overline{y_\imath} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_\imath(|x|)}$, M_p accepts $(\overline{x}, \overline{y_1}, \ldots, \overline{y_\imath})$
  where $\{\mathbf{0},\mathbf{1}\}^{\leq n} = \{x \in \{\mathbf{0},\mathbf{1}\}^* \mid |x| \leq n\}$, $\mathrm{Q} = \exists$ if $\imath$ is odd and $\mathrm{Q} = \forall$ otherwise.

The class $\mathrm{D}\Pi_{\mathcal{K}}^{\imath}$ is defined replacing the third line above by $\overline{x} \in S \Leftrightarrow \forall \overline{y_1} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_1(|x|)} \exists \overline{y_2} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_2(|x|)} \ldots \mathrm{Q}\overline{y_\imath} \in \{\mathbf{0},\mathbf{1}\}^{\leq p_\imath(|x|)}$, M_p accepts $(\overline{x}, \overline{y_1}, \ldots, \overline{y_\imath})$.

In addition, we define $\mathrm{D}\Box_{\mathcal{K}}^{\imath} = \mathrm{D}\Sigma_{\mathcal{K}}^{\imath} \cup \mathrm{D}\Pi_{\mathcal{K}}^{\imath}$ and the *digital polynomial hierarchy* satisfies $\mathrm{DPH}_{\mathcal{K}} = \bigcup_{\imath=0}^{\infty} \mathrm{D}\Sigma_{\mathcal{K}}^{\imath} = \bigcup_{\imath=0}^{\infty} \mathrm{D}\Pi_{\mathcal{K}}^{\imath} = \bigcup_{\imath=0}^{\infty} \mathrm{D}\Box_{\mathcal{K}}^{\imath}$.

5.2 Safe Recursion with Digital Predicative Minimization

Similarly to the notion of predicative minimization of previous section, we introduce the notion of digital predicative minimization:

Definition 5.2 Given $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, we define $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}$ by *digital predicative minimization*:

$$f(\overline{x};\overline{a}) = \mathrm{d}\mu\overline{b}(h(\overline{x};\overline{a},\overline{b})) = \begin{cases} \mathrm{cons}(;\mathbf{1}, \min_t\{\overline{b} : \ |\overline{b}| = t \& h(\overline{x};\overline{a},\overline{b}) = \epsilon\}) \\ \qquad\qquad \text{if there is such a } \overline{b} \in \{\mathbf{0},\mathbf{1}\}^* \\ \epsilon \qquad\qquad \text{otherwise.} \end{cases}$$

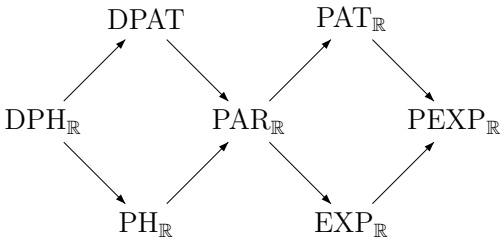
Theorem 1.3 is formally the following:

Theorem 5.3 *A decision problem over $\mathcal{K}$ belongs to* $\mathrm{DPH}_{\mathcal{K}}$ *if and only if its characteristic function belongs to* $\mathrm{CharDPH}_{\mathcal{K}}$.

6 Conclusion and discussion

In this paper, we proved that parallel polynomial time, as well as the polynomial hierarchy, including the digital version can be characterized in a machine independent way.

Future work include extending these results to other classes. In particular, when dealing with the real numbers, the following inclusions are known [3]:



where an arrow means inclusion, $\mathrm{EXP}_{\mathbb{R}}$ denotes exponential time, $\mathrm{PEXP}_{\mathbb{R}}$ parallel exponential time, $\mathrm{PAT}_{\mathbb{R}}$ polynomial alternating time, and $\mathrm{DPAT}_{\mathbb{R}}$ digital polynomial alternating time. The two inclusions $\mathrm{PAR}_{\mathbb{R}} \subset \mathrm{PAT}_{\mathbb{R}}$ and $\mathrm{PAR}_{\mathbb{R}} \subset \mathrm{EXP}_{\mathbb{R}}$ are known to be strict [3].

Observe that $\mathrm{PSPACE}, \mathrm{PAT}_{\mathcal{K}}, \mathrm{DPAT}_{\mathcal{K}}, \mathrm{PAR}_{\mathcal{K}}$ yield the same class when restricted to finite structure (i.e. in classical complexity), whereas over arbitrary structure they are a priori all distinct.

Can we characterize $\mathrm{PAT}_{\mathcal{K}}$ and $\mathrm{DPAT}_{\mathcal{K}}$ implicitly over arbitrary structures?

References

[1] S. Bellantoni. Predicative recursion and the polytime hierarchy. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II, Perspectives in Computer Science*. Birkhaüser, 1994.

[2] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[3] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[4] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21:1–46, 1989.

[5] Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Safe recursion over an arbitrary structure. sequential and parallel polynomial time. In *Foundations of Software Science and Computational Structures 6th International Conference, FOSSACS 2003*, number 2620 in Lecture Notes in Computer Science, pages 185–199. Springer, 2003.

[6] Olivier Bournez, Paulin Jacobé de Naurois and Jean-Yves Marion Safe recursion and calculus over an arbitrary structure. In *Implicit Computational Complexity, ICC 2002*, Copenhagen, Denmark, 20-21 June 2002.

[7] P. Clote. Computational models and function algebras. In D. Leivant, editor, *LCC'94*, volume 960 of *Lecture Notes in Computer Science*, pages 98–130. Springer-Verlag, 1995.

[8] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

[9] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[10] R. Fagin. Generalized first order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computation*, pages 43–73. SIAM-AMS, 1974.

[11] J. B. Goode. Accessible telephone directories. *Journal of Symbolic Logic*, 59(1):92–105, 1994.

[12] Erich Grädel and Yuri Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 10 January 1998.

[13] Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 315–324, Las Vegas, Nevada, 29 May–1 June 1995.

[14] Y. Gurevich. Algebras of feasible functions. In *Twenty Fourth Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, 1983.

[15] M. Hofmann. Type systems for polynomial-time computation, 1999. Habilitation.

[16] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

[17] N. Jones. The expressive power of higher order types. *J. of Functional Programming*, 11:55–94, 2001.

[18] D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, volume 960 of *Lecture Notes in Computer Science*, pages 177–194. Springer-Verlag, 1995.

[19] D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer-Verlag.

[20] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 25–42. Springer-Verlag, Nov 2000.

[21] C. Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *C. R. Acad. Sci. Paris*, 309, Série I:435–437, 1989.

[22] B. Poizat. *Les Petits Cailloux*. Aléas, 1995.

[23] V. Sazonov. Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung und Kybernetik*, 7:319–323, 1980.