# Chapter 6

# Syntactic Characterizations of Complexity Classes in Blum Shub and Smale Model

In this chapter, we focus on the Blum Shub and Smale [Blum et al., 1989] model of computations. This is a discrete time model that has been initially introduced to discuss of the complexity of problems related to polynomials [Blum et al., 1989], [Blum et al., 1998]. It has later on been extended by Poizat in [Poizat, 1995], [Goode, 1994] into a model of computations over an arbitrary logical structure.

This chapter is conceived as a catalogue of our results related to the characterization of complexity classes in this model. All the results that are presented here are in the spirit of the characterization of polynomial time of [Bellantoni and Cook, 1992] in classical complexity.

## 6.1 Introduction

Blum Shub and Smale (BSS) model is one of the approaches to study the complexity or the computability of problems over the reals. The most known approach is recursive analysis introduced by Turing [Turing, 1936], Grzegorczyk [Grzegorczyk, 1957], and Lacombe [Lacombe, 1955]. In the recursive analysis approach one considers that a real is represented by a quickly converging sequence of rationals, and one says that a function is computable when there is an effective way to transform any representation of a real into a representation of the image of the real by the function: see [Weihrauch, 2000] for a recent monograph that presents recursive analysis. Complexity notions can be defined in this model [Ko, 1991].

Blum, Shub and Smale introduced in [Blum et al., 1989] another approach to talk about the complexity of problems over the reals. Their model is sometimes called the real Turing machine. This model, in opposition to recursive analysis, measures the complexity of problems in terms of the number of arithmetic operations that are required for their resolution, independently of the representation of reals.

The model, initially introduced to deal with algebraic complexity of problems on the field of the reals, or more generally over arbitrary rings, has later on been extended by Poizat in [Poizat, 1995], [Goode, 1994] into a computational model over an arbitrary logical structure.

Depending on the considered logical structure, one can obtain a whole complexity theory, with complexity classes such as $P$, $NP$, reduction notions, complete problems, and questions like $P \neq NP$?, for which an answer can sometimes be provided for some structures.

As classical complexity can be seen as the restriction of this notion of computations for the particular case of boolean or finite structures, this model gives new insights on older problems of classical complexity, or on their links with logic.

In particular, this model gave birth to numerous papers devoted to understand which result from classical complexity can be generalized to other structures than booleans: see monograph [Blum et al., 1998].

Through the PhD of Paulin Jacobé de Naurois, also co-supervised by Felipe Cucker in Hong-Kong and Jean-Yves Marion in Nancy, we tried to understand if it was possible to characterize syntactically complexity classes in this model over an arbitrary structure.

Since there exist in classical complexity several such characterizations, the question can be seen as understanding whether these results extend to arbitrary logical structures.

Another strong motivation is the following: syntactic characterizations of complexity classes define these classes without any explicit reference to a notion of machine. Since there are several computational models over the reals, this adds to the legitimacy of considered complexity classes. Indeed, since the relations between the models over the reals are far from being all clear, being able to define complexity classes without fixing the model of computation is a way to circumvent the problem, and to legitimate the idea that the considered class is independent of all variations that can be considered on the model.

Observe in addition, that we obtain some characterizations of complexity classes that are sometimes as much natural, or even more, natural than classical definitions.

Based on the characterization by Bellantoni et Cook of polynomial time in classical complexity in [Bellantoni and Cook, 1992], we first obtained a syntactic characterization of computable functions, as well as of polynomial time computable functions in terms of safe recursive functions.

We later on obtained a characterization of functions that are computable in parallel polynomial time in terms of recursive functions with substitutions. This result generalizes the result of [Leivant and Marion, 1995] to arbitrary structures.

By generalizing the results from [Bellantoni, 1994], we also obtained a characterization of all levels of the polynomial hierarchy in terms of safe recursion with predicative minimization, and of the polynomial digital hierarchy in terms of safe recursion with digital predicative minimization.

We also characterized polynomial alternating time in terms of safe recursion with predicative substitutions, and digital polynomial alternating time in terms of safe recursion with digital predicative substitutions.

In this chapter, we recall in a synthetic (or catalogue) way the set of our results. The results that are presented here are published in journals [Bournez et al., 2006], [Bournez et al., 2005a], and in conferences [Bournez et al., 2003], [Bournez et al., 2004a], and [Bournez et al., 2004b].

All these characterizations are in the straight line of the characterizations, of the so-called implicit complexity, following the work from Bellantoni and Cook [Bellantoni and Cook, 1992]. The first characterization of polynomial time by an algebra of functions without explicit reference to a model of computation is due to Cobham in [Cobham, 1962]. However, the schemas of [Bellantoni and Cook, 1992] are really more natural.

Other machine-independent characterizations exist in classical complexity: see for example the monographs and surveys [Clote, 1998], [Ebbinghaus and Flum, 1995], [Immerman, 1999]. In particular, there is the whole domain of research that concerns the characterizations of descriptive complexity, based on relations or global methods of finite model theory.

Descriptive complexity is born from the work of Fagin [Fagin, 1974], which proved that class $NP$ can be characterized as the class of sets that can be defined in second-order existential logic. Vardi and Immerman [Vardi, 1982], [Immerman, 1987], [Immerman, 1986] used this approach to characterize complexity class $P$. Other characterizations exist for classes such as $LOGSPACE$, in [Gurevich, 1983] or for class $PSPACE$, in [Moschovakis, 1984], [Gurevich and Shelah, 1986], [Immerman, 1987], [Bonner, 1989], [Abiteboul and Vianu, 1989], [Leivant, 1990], [Abiteboul et al., 1990], [Abiteboul and Vianu, 1991], [Immerman, 1991]. A synthetic presentation of the domain can be found in [Ebbinghaus and Flum, 1995], [Immerman, 1999], [Clote, 1998]. All these characterizations are in classical complexity.

In [Grädel and Meer, 1995], the notion of $\mathbb{R}$-structure was introduced, and characterizations of classes $P$ and $NP$ over the field of reals in terms of logics over these $\mathbb{R}$-structures have been established. These results have later on been extended in [Cucker and Meer, 1999] to other complexity classes, and also in [Grädel and Gurevich, 1998] to other structures than the fields of reals.

We are also coauthors of an extension of the notion of $\mathbb{R}$-structure to arbitrary structures. However, we decided to focus in this chapter on the approaches *à la* Bellantoni and Cook. Refer to [Bournez et al., 2005b] for these characterizations.

We will add to all these arguments about motivation, that about the question why being interested in the BSS model, that chapter 1 shown several times that polynomials appear naturally when one discusses

problems over the reals, even when they are not expected. This model is clearly more natural than recursive analysis to discuss the complexity[1] of problems related to polynomials.

Furthermore, in a programming perspective, one way to interpret these results is to consider computability over an arbitrary structure as a programming language with extra-operators coming from some external library. This observation, and its potential for the construction of methods to derive automatically properties of programs in the spirit of [Hofmann, 1999], [Jones, 2001], is also one of the motivations of this work.

All the results in this chapter constitute a part of the PhD thesis work of Paulin Jacobé de Naurois. They all have been obtained in collaboration with his supervisors, Felipe Cucker, Jean-Yves Marion and myself.

## 6.2 Computations over an Arbitrary Structure

Let's start by introducing briefly computability and complexity over an arbitrary structure. For more details, refer to monograph [Blum et al., 1998], for structures in relations with real numbers, or complex numbers, or to the book [Poizat, 1995] for considerations on more general structures.

**Definition 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* $op_i$*, and a finite number of relations* $rel_1, \ldots, rel_l$*. Constants correspond to operators of arity 0. While the index set* $I$ *may be infinite, the number of operators of arity greater than zero 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 the sequel) in the structure.*

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

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 $\Sigma^*$ the set of all finite sequences of zeros and ones. It provides the inputs for machines over $\mathcal{K}$. For technical reasons we shall also consider the bi-infinite direct sum $\mathbb{K}_*$. Elements of this space have the form

$$(\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots)$$

where $x_i \in \mathbb{K}$ for all $i \in \mathbb{Z}$ and $x_k = 0$ for $k$ sufficiently large in absolute value. The space $\mathbb{K}_*$ has natural shift operations, shift left $\sigma_\ell : \mathbb{K}_* \to \mathbb{K}_*$ and shift right $\sigma_r : \mathbb{K}_* \to \mathbb{K}_*$ where

$$\sigma_\ell(x)_i = x_{i-1} \qquad \text{and} \qquad \sigma_r(x)_i = x_{i+1}.$$

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 $\epsilon$ the empty word. The length of a word $\overline{w} \in \mathbb{K}^*$ is denoted by $|\overline{w}|$.

We now define machines over $\mathcal{K}$ following the lines of [Blum et al., 1998].

**Definition 2** *A* machine over $\mathcal{K}$ *consists of an input space* $\mathcal{I} = \mathbb{K}^*$*, an output space* $\mathcal{O} = \mathbb{K}^*$*, and a register space*[2] $\mathcal{S} = \mathbb{K}_*$*, together with a connected directed graph whose nodes labeled* $0, \ldots, N$ *correspond to the set of different* instructions *of the machine. These nodes are of one of the five following types: input, output, computation, branching and shift nodes. Let us describe them a bit more.*

1. *Input nodes. There is only one input node and is labeled with 0. Associated with this node there is a next node* $\beta(0)$*, and the input map* $g_I : \mathcal{I} \to \mathcal{S}$*.*

2. *Output nodes. There is only one output node which is labeled with* 1*. It has no next nodes, once it is reached the computation halts, and the output map* $g_O : \mathcal{S} \to \mathcal{O}$ *places the result of the computation in the output space.*

[1] and not necessarily the computability.

[2] In the original paper by Blum, Shub and Smale, this is called the *state* space. We rename it *register* space to avoid confusions with the notion of 'state' in a Turing machine.

3. *Computation nodes. Associated with a node $m$ of this type there are a next node $\beta(m)$ and a map $g_m : \mathcal{S} \to \mathcal{S}$. The function $g_m$ replaces the component indexed by 1 of $\mathcal{S}$ by the value $op(w_1, \ldots, w_n)$ where $w_1, w_2, \ldots, w_n$ are components 1 to $n$ of $\mathcal{S}$ and $op$ is some operation of the structure $\mathcal{K}$ of arity $n$. The other components of $\mathcal{S}$ are left unchanged. When the arity $n$ is zero, $m$ is a constant node. A given machine uses only a finite number of constants, however, in order to compare different machines and to denote the notion of reduction between them and completeness, we need to include all possible constants in the underlying structure $\mathcal{K}$. Thus the possibly infinite index set $I$.*

4. *Branch nodes. There are two nodes associated with a node $m$ of this type: $\beta^+(m)$ and $\beta^-(m)$. The next node is $\beta^+(m)$ if $rel(w_1, \ldots, w_n)$ is true and $\beta^-(m)$ otherwise. Here $w_1, w_2, \ldots, w_n$ are components 1 to $n$ of $\mathcal{S}$ and $rel$ is some relation of the structure $\mathcal{K}$ of arity $n$.*

5. *Shift nodes. Associated with a node $m$ of this type there is a next node $\beta(m)$ and a map $\sigma : \mathcal{S} \to \mathcal{S}$. The $\sigma$ is either a left or a right shift.*

*Several conventions for the contents of the register space at the beginning of the computation have been used in the literature [Blum et al., 1998], [Blum et al., 1989], [Poizat, 1995]. We will not dwell on these details but focus on the essential ideas in the proofs to come in the sequel.*

In other words, a machine over $\mathcal{K}$ is essentially a Turing Machine, which is able to perform the basic operations $\{op_i\}$ and the basic tests $rel_1, \ldots, rel_l$ at unit cost, and whose tape cells can hold arbitrary elements of the underlying set $\mathbb{K}$ [Poizat, 1995], [Blum et al., 1998]. Note that the register space $\mathcal{S}$ above has the function of the tape and that its component with index 1 plays the role of the scanned cell.

**Definition 3** *For a given machine $M$, the function $\varphi_M$ associating its output to a given input $x \in \mathbb{K}^*$ is called the* input-output function. *We shall say that a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is* computable *when there is a machine $M$ such that $f = \varphi_M$.*

*Also, a set $A \subseteq \mathbb{K}^*$ is* decided *by a machine $M$ if its characteristic function $\chi_A : \mathbb{K}^* \to \{\mathbf{0}, \mathbf{1}\}$ coincides with $\varphi_M$.*

*A* configuration *of a machine $M$ over $\mathcal{K}$ is given by an instruction $q$ of $M$ along with the position of the head of the machine and two words $w_l, w_r \in \mathbb{K}^*$ that give the contents of the tape at left and right of the head.*

We can now define some central complexity classes.

**Definition 4** *A set $S \subset \mathbb{K}^*$ is in class $\mathrm{P}_\mathcal{K}$ (respectively a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is in class $\mathrm{FP}_K$), if there exist a polynomial $p$ and a machine $M$, so that for all $\overline{w} \in \mathbb{K}^*$, $M$ stops in time $p(|\overline{w}|)$ and $M$ accepts iff $\overline{w} \in S$ (respectively, $M$ computes function $f(\overline{w})$).*

This notion of computability corresponds to the classical one for structures over the booleans or the integers, and corresponds to the one of Blum Shub and Smale in [Blum et al., 1989] over the real numbers.

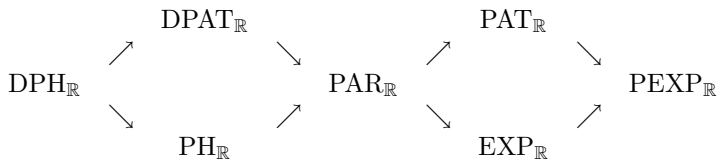
**Proposition 1** **(i)** *The class $\mathrm{P}_\mathcal{K}$ is the classical* P *when $\mathcal{K} = (\{0, 1\}, =, \mathbf{0}, \mathbf{1})$.*

**(ii)** *The class $\mathrm{P}_\mathcal{K}$ is the class $\mathrm{P}_\mathbb{R}$ of [Blum et al., 1989] when $\mathcal{K} = (\mathbb{R}, +, -, *, =, \leq, \{c \in \mathbb{R}\})$.*

It is also possible to consider the analogue of all the other classes known in classical complexity.

For non-deterministic classes, one subtlety is that 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 $\{0, 1\}$. 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, i.e., over a finite structure, these two notions of nondeterminism coincide and they yield the same polynomial hierarchy and class of polynomial alternating time. Moreover, polynomial alternating time coincides with PSPACE and with PAR (the class

of sets decided in parallel polynomial time). This needs not to be so over infinite structures. For instance, over $(\mathbb{R}, +, -, *, /, \leq)$, we have the following inclusions of complexity classes [Cucker, 1993]




where an arrow means inclusion, $\mathrm{EXP}_\mathbb{R}$ denotes exponential time, $\mathrm{PEXP}_\mathbb{R}$ parallel exponential time, $\mathrm{PH}_\mathbb{R}$ is the polynomial hierarchy, and $\mathrm{PAT}_\mathbb{R}$ polynomial alternating time. In addition 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. Refer to [Blum et al., 1998] for a complete discussion.

In what follows, we will present syntactic characterizations of main classes of this diagram. The rest of this chapter is voluntary a catalogue of our results.

## 6.3 Computable Functions

Let's start by characterizing computable functions. As in the classical setting, computable functions over an arbitrary structure $\mathcal{K}$ can be characterized algebraically, in terms of the smallest set of functions containing some initial functions and closed by composition, primitive recursion and minimization. In the rest of this section we present such a characterization.

### 6.3.1 Partial Recursive Functions and Primitive Recursive Functions

We consider functions $(\mathbb{K}^*)^n \to \mathbb{K}^*$, taking as inputs arrays of words of elements in $\mathbb{K}$, and returning as output a word of elements in $\mathbb{K}$. When the output of a function is undefined, we use the symbol $\perp$.

**Definition 5** *We call* basic functions *the following four kinds of functions:*

**(i)** *Functions making elementary manipulations of words over $\mathbb{K}$. For any $a \in \mathbb{K}, \overline{x}, \overline{x_1}, \overline{x_2} \in \mathbb{K}^*$*

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

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

$$\mathsf{Pr}_i^n(\overline{x_1}, \ldots, \overline{x_i}, \ldots, \overline{x_n}) = \overline{x_i}.$$

**(iii)** *Functions of structure. For any operator (including the constants treated as operators of arity 0) $op_i$ or relation $rel_i$ of arity $n_i$ we have the following initial functions:*

$$\begin{array}{rcl} \mathsf{Op}_i(a_1.\overline{x_1}, \ldots, a_{n_i}.\overline{x_{n_i}}) & = & (op_i(a_1, \ldots, a_{n_i})).\overline{x_{n_i}} \\ \mathsf{Rel}_i(a_1.\overline{x_1}, \ldots, a_{n_i}.\overline{x_{n_i}}) & = & \begin{cases} \mathbf{1} \text{ if } rel_i(a_1, \ldots, a_{n_i}) \\ \epsilon \text{ otherwise.} \end{cases} \end{array}$$

**(iv)** *Selection function*

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

*The set of* partial recursive functions *over $\mathcal{K}$ is the smallest set of functions $f : (\mathbb{K}^*)^k \to \mathbb{K}^*$ containing the basic functions and closed under the following operations:*

**(1)** Composition. *Assume* $g : (\mathbb{K}^*)^n \to \mathbb{K}^*$, $h_1, \ldots, h_n : \mathbb{K}^* \to \mathbb{K}^*$ *are given partial functions. Then the composition* $f : \mathbb{K}^* \to \mathbb{K}^*$ *is defined by*

$$f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_n(\overline{x})).$$

**(2)** Primitive recursion. *Assume* $h : \mathbb{K}^* \to \mathbb{K}^*$ *and* $g : (\mathbb{K}^*)^3 \to \mathbb{K}^*$ *are given partial functions. Then we define* $f : (\mathbb{K}^*)^2 \to \mathbb{K}^*$

$$\begin{array}{rcl} f(\epsilon, \overline{x}) & = & h(\overline{x}) \\ f(a.\overline{y}, \overline{x}) & = & \left\{ \begin{array}{ll} g(\overline{y}, f(\overline{y}, \overline{x}), \overline{x}) & \text{if } f(\overline{y}, \overline{x}) \neq \perp \\ \perp & \text{otherwise.} \end{array} \right. \end{array}$$

**(3)** Minimization. *Assume* $g : (\mathbb{K}^*)^2 \to \mathbb{K}^*$ *is given. Function* $f : \mathbb{K}^* \to \mathbb{K}^*$ *is defined by minimization on the first argument of* $g$*, written by* $f(\overline{y}) = \mu\overline{x}\,(g(\overline{x}, \overline{y}))$*, if:*

$$\mu\overline{x}\,(g(\overline{x}, \overline{y})) = \left\{ \begin{array}{ll} \perp & \text{if } \forall t \in \mathbb{N} \ : \mathsf{hd}(g(0^t, \overline{y})) \neq \mathbf{1} \\ \mathbf{1}^k : \ k = \min\{t \mid \mathsf{hd}(g(0^t, \overline{y})) = \mathbf{1}\} & \text{otherwise.} \end{array} \right.$$

*Partial recursive functions defined without using the minimization operator are called* primitive recursive.

Let's do some remarks about theses schemas.

**(i)** The formal definition of function $\mathsf{tl}$ is actually a primitive recursive definition with no recurrence argument. However, when we introduce the notion of safe recursion in Section 6.4, this function $\mathsf{tl}$ needs to be given as *a priori* functions in order to be applied to safe arguments, and not only to normal arguments. For the sake of coherence, we give it here as an *a priori* function as well.

**(ii)** Note that primitive recursive functions are total functions whereas partial recursive functions may be partial functions.

**(iii)** The operation of minimization on the first argument of $g$ returns the smallest word in $\{\mathbf{1}\}^*$ satisfying a given property. The reason why it does not return a smallest word made of *any* letter in $\mathbb{K}$ is to ensure determinism, and therefore computability. On a structure where NP is not decidable, such a non-deterministic minimization may not be computable by a BSS machine, which is in essence deterministic.

**(iv)** In the definition of composition, primitive recursion, and minimization above we have taken arguments $\overline{x}, \overline{y} \in \mathbb{K}^*$. This is to simplify notations. To be fully formal, we should allow for arguments in $(\mathbb{K}^*)^p$ with $p \geq 1$. We will adopt these simplifications all throughout this paper since the proofs for the fully formal case would not be different: just notationally more involved.

**(v)** In the definition of primitive recursion, the variable $a$ in front of the recurrence argument $a.\overline{y}$ does not appear as argument of the function $g$. The first reason for this is the need of consistency among argument types: $a$ is a single element in $\mathbb{K}$ whereas all arguments need to be words in $\mathbb{K}^*$. The second reason is that $g$ may still depend on the value of the first element of $\overline{y}$.

**(vi)** Our definition of primitive recursion and of minimization is slightly different from the one found in [Blum et al., 1989]. In this paper, the authors introduce a special integer argument for every function, which is used to control recursion and minimization, and consider the other arguments as simple elements in $\mathbb{K}$. Their functions are of type $f : \mathbb{N} \times \mathbb{K}^k \to \mathbb{K}^l$. Therefore, they only capture finite dimensional functions. It is known that over the real numbers with $+, -, *$ operators finite dimensional functions are equivalent to non-finite dimensional functions (see [Michaux, 1989]). But this is not true over other structures, for instance $\mathbb{Z}/2\mathbb{Z}$. Our choice is to consider arguments as words of elements in $\mathbb{K}$, and to use the length of the arguments to control recursion and minimization. This allows us to capture non-finite dimensional functions over arbitrary structures.

Observe that over structure $\{\{0, 1\}, =, \mathbf{0}, \mathbf{1}\}$, our partial recursive (resp. primitive recursive) functions coincide with the classical partial recursive (resp. primitive recursive) functions.

### 6.3.2 Characterization of Computable Functions

We proved the following result in [Bournez et al., 2002], [Bournez et al., 2003]. This is reprinted in journal [Bournez et al., 2005a].

**Theorem 1** *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i\in I}, \mathbf{0}, \mathbf{1})$, a function is BSS computable if and only if it can be defined as a partial recursive function over $\mathcal{K}$.*

## 6.4 Polynomial Time

We now present a characterization of polynomial time. To do so, we extend to an arbitrary structure $\mathcal{K}$ the notion of safe recursive function over the natural numbers defined by Bellantoni and Cook in their seminal paper [Bellantoni and Cook, 1992].

### 6.4.1 Safe Recursive Functions

**Example 1** *Consider the following function*

$$\exp(\overline{x}) = \mathbf{1}^{2^{|\overline{x}|}}$$

*which computes in unary the exponential. It can be easily defined with primitive recursion by*

$$\begin{aligned} \mathsf{Cons}(\epsilon, \overline{y}) &= \overline{y} \\ \mathsf{Cons}(a.\overline{x}, \overline{y}) &= \mathsf{cons}(a, \mathsf{Cons}(\overline{x}, \overline{y})) \\ \exp(\epsilon) &= \mathbf{1} \\ \exp(a.\overline{x}) &= \mathsf{Cons}(\exp(\overline{x}), \exp(\overline{x})). \end{aligned}$$

*On the other hand, note that* $\exp \notin \mathrm{FP}_{\mathcal{K}}$ *since the computed value is exponentially large in the size of its argument. The goal of this section is to introduce a restricted version of recursion not allowing for this exponential growth.*

Safe recursive functions are defined in a similar manner as primitive recursive functions. However, following [Bellantoni and Cook, 1992], safe recursive functions have two different types of arguments, each of them having different properties and purposes. The first type of argument, called *normal*, is similar to the arguments of the previously defined partial recursive and primitive recursive functions, since it 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. In a recursion, the recurrence argument can only be in safe position. We will see that 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 $\overline{x}$ and $\overline{y}$ are these arguments, we will write $f(\overline{x}; \overline{y})$ separating them by a semicolon ";". Normal arguments are placed at the left of the semicolon and safe arguments at its right.

We define now safe recursive functions. To do so, we consider the set of *basic safe functions* that are the basic functions of Definition 5 with the feature that their arguments are all safe.

**Definition 6** *The set of* safe recursive functions *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 the following operations:*

**(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* $h : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ *and* $g : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ *are given functions. Function* $f : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ *can then be defined by safe recursion:*

$$\begin{aligned} f(\epsilon, \overline{x}; \overline{y}) &= h(\overline{x}; \overline{y}) \\ f(a.\overline{z}, \overline{x}; \overline{y}) &= g(\overline{z}, \overline{x}; f(\overline{z}, \overline{x}; \overline{y}), \overline{y}) \end{aligned}$$

Let's do some remarks.

**(i)** Using safe composition it is possible to "move" an argument from a normal position to a safe position, whereas the reverse is forbidden. For example, assume $g : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ is a given function. One can then define with safe composition a function $f$ given by $f(\overline{x}, \overline{y}; \overline{z}) = g(\overline{x}; \overline{y}, \overline{z})$ but a definition like $f(\overline{x}; \overline{y}, \overline{z}) = g(\overline{x}, \overline{y}; \overline{z})$ is not valid.

**(ii)** Note that it is impossible to turn the definition of exp in Example 1 into a safe recursion scheme. $\mathsf{Cons}(x; y)$ and $\mathsf{Cons}(x, y;)$ can be defined as follows:

$$\begin{aligned} \mathsf{Cons}(\epsilon; \overline{y}) &= \overline{y} \\ \mathsf{Cons}(a.\overline{x}; \overline{y}) &= \mathsf{cons}(; a, \mathsf{Cons}(\overline{x}; \overline{y})) \\ \mathsf{Cons}(\epsilon, \overline{y};) &= \overline{y} \\ \mathsf{Cons}(a.\overline{x}, \overline{y};) &= \mathsf{cons}(; a, \mathsf{Cons}(\overline{x}, \overline{y};)) \end{aligned}$$

However, exp can not be defined, since this would need the use of the recurrence argument $\exp(\overline{x})$ as a normal argument of function $\mathsf{Cons}$, which is forbidden. This obstruction is at the basis of the equivalence between safe recursion and polynomial time computability.

### 6.4.2 Characterization of Polynomial Time

We proved the following result in conferences [Bournez et al., 2002], [Bournez et al., 2003], and in journal [Bournez et al., 2005a], that extends [Bellantoni and Cook, 1992] to arbitrary structures.

**Theorem 2** *Over any structure* $\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i \in I}, \mathbf{0}, \mathbf{1})$*, a function is computed in polynomial time by a BSS machine if and only if it can be defined as a safe recursive function over* $\mathcal{K}$*.*

## 6.5 Parallel Polynomial Time

### 6.5.1 Definitions

The reader can find in [Blum et al., 1998] the definition of parallel machine over a structure $\mathcal{K}$. We will not give formal definitions here[3], and we let the reader refer to [Blum et al., 1998].

**Definition 7** *$FPAR_{\mathcal{K}}$ is the class of functions $f$ computable in polynomial time by a parallel machine using an exponentially bounded number of processors and such that $|f(\overline{x})| = |\overline{x}|^{\mathcal{O}(1)}$ for all $\overline{x} \in \mathbb{K}^*$.*

Refer to [Bournez et al., 2005a] for a definition of this class in terms of circuits, without reference to a notion of parallel machine.

[3]The exposition on parallelism in [Blum et al., 1998, Chapter 18] is for $\mathcal{K}$ the real numbers but the definition of parallel machine carry on to arbitrary structures.

### 6.5.2 Safe Recursion with Substitutions

**Definition 8** *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 substitutions. *Let $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, $g : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^{l+1} \to \mathbb{K}^*$, and $\sigma_j : \emptyset \times \mathbb{K}^* \to \mathbb{K}^*$ for $0 < j \leq l$ be safe recursive functions.*

*Function $f : (\mathbb{K})^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ is defined by safe recursion with substitutions as follows:*

$$\begin{aligned} f(\epsilon, \overline{x}; \overline{u}, \overline{y}), &= h(\overline{x}; \overline{u}, \overline{y}) \\ f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) &= \begin{cases} g\left(\overline{z}, \overline{x}; f(\overline{z}, \overline{x}; \sigma_1(;\overline{u}), \overline{y}), \ldots, f(\overline{z}, \overline{x}; \sigma_l(;\overline{u}), \overline{y}), \overline{y}\right) \\ \qquad\qquad \text{if } \forall j \ f(\overline{z}, \overline{x}; \sigma_j(;\overline{u}), \overline{y}) \neq \perp \\ \perp \qquad\qquad \text{otherwise.} \end{cases} \end{aligned}$$

*The functions $\sigma_j$ are called* substitution functions.

### 6.5.3 Characterization of Parallel Polynomial Time

We proved the following result in papers [Bournez et al., 2003], [Bournez et al., 2005a]. These results extends [Leivant and Marion, 1995] to an arbitrary structure.

**Theorem 3** *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i\in I}, \mathbf{0}, \mathbf{1})$, a function is computed in parallel polynomial time by a BSS machine if and only if it can be defined as a safe recursive function with substitutions over $\mathcal{K}$.*

Observe that, unlike Leivant and Marion, Theorem 4 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 [Michaux, 1989], on some structures like $(\mathbb{R}, +, -, *, \leq, \mathbf{0}, \mathbf{1})$, any computable function can be computed in constant working space. However, in the classical case, one has FPAR= FPSPACE.

## 6.6 Polynomial Hierarchy

### 6.6.1 Definitions

As in the classical setting, the polynomial hierarchy over a given structure $\mathcal{K}$ can be defined in several equivalent ways, including syntactic descriptions, or semantic definitions by successive relativizations of non-deterministic polynomial time (see [Blum et al., 1998]).

Recall some basic complexity classes:

- $\mathrm{P}_{\mathcal{K}}$ is the class of problems over $\mathcal{K}$ decided in polynomial time. We denote by $\mathrm{FP}_{\mathcal{K}}$ the class of functions over $\mathcal{K}$ computed in polynomial time.
- A decision problem $A$ is in $\mathrm{NP}_{\mathcal{K}}$ if and only if there exists a decision problem $B$ in $\mathrm{P}_{\mathcal{K}}$ and a polynomial $p_B$ such that $\overline{x} \in A$ if and only if there exists $\overline{y} \in \mathbb{K}^*$ with $|\overline{y}| \leq p_B(|\overline{x}|)$ satisfying $(\overline{x}, \overline{y}) \in B$.
- A decision problem $A$ is in $\mathrm{coNP}_{\mathcal{K}}$ if and only if there exists a decision problem $B$ in $\mathrm{P}_{\mathcal{K}}$ and a polynomial $p_B$ such that $\overline{x} \in A$ if and only if for all $\overline{y} \in \mathbb{K}^*$ with $|\overline{y}| \leq P_B(|\overline{x}|)$, $(\overline{x}, \overline{y})$ is in $B$.

**Definition 9** *Let $\Sigma^0_{\mathcal{K}} = \mathrm{P}_{\mathcal{K}}$ and, for $i \geq 1$, $\Sigma^i_{\mathcal{K}} = \mathrm{NP}^{\Sigma^{i-1}_{\mathcal{K}}}_{\mathcal{K}}$, $\Pi^i_{\mathcal{K}} = \mathrm{coNP}^{\Sigma^{i-1}_{\mathcal{K}}}_{\mathcal{K}}$.*

*The* polynomial time hierarchy *over $\mathcal{K}$ is $\mathrm{PH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \Sigma^i_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \Pi^i_{\mathcal{K}}$.*

*A function is in $\mathrm{F}\Delta^i_{\mathcal{K}}$ if it computable in polynomial time by a machine over $\mathcal{K}$ which queries an oracle in $\Sigma^i_{\mathcal{K}}$. That is, $\mathrm{F}\Delta^i_{\mathcal{K}} = \mathrm{FP}^{\Sigma^i_{\mathcal{K}}}_{\mathcal{K}} = \mathrm{FP}^{\Pi^i_{\mathcal{K}}}_{\mathcal{K}}$.*

*The* functional polynomial time hierarchy *over $\mathcal{K}$ is $\mathrm{FPH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{F}\Delta^i_{\mathcal{K}}$.*

Extending the classical notion of polynomial time reduction between decision problems, complete problems for every of the $\Sigma^i_{\mathcal{K}}$ and $\Pi^i_{\mathcal{K}}$ have been shown to exist [Blum et al., 1998], [Poizat, 1995].

### 6.6.2 Safe Recursion with Predicative Minimization

In the spirit of [Bellantoni, 1994], , we now introduce the notion of predicative minimization.

**Definition 10** *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 *as follows*

$$f(\overline{x};\overline{a}) = \mathbf{\exists}\overline{b}(h(\overline{x};\overline{a},\overline{b})) = \begin{cases} \mathbf{1} & \text{if there exists } \overline{b} \in \mathbb{K}^* \text{ such that } h(\overline{x};\overline{a},\overline{b}) = \mathbf{0} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

We now introduce new sets of functions.

**Definition 11** *Let* $F$ *be a class of BC functions. The set of* restricted safe recursive functions relative to F *over* $\mathcal{K}$*, denoted by* $RSR_{\mathcal{K}}(F)$*, is the smallest set of functions containing the basic safe functions and* $F$*, and closed under the following* restricted safe composition *scheme*

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

*where the* $h_i$ *belong to* $RSR_{\mathcal{K}}(F)$ *and* $g$ *to* $\mathrm{SR}_{\mathcal{K}}$*, and the following* restricted safe recursion *scheme*

$$\begin{aligned} f(\epsilon,\overline{x};\overline{y}) &= h(\overline{x};\overline{y}) \\ f(a.\overline{z},\overline{x};\overline{y}) &= g(\overline{z},\overline{x};f(\overline{z},\overline{x};\overline{y}),\overline{y}) \end{aligned}$$

*where* $h$ *belongs to* $RSR_{\mathcal{K}}(F)$ *and* $g$ *to* $\mathrm{SR}_{\mathcal{K}}$*. This implies that no function in* $F\backslash\mathrm{SR}_{\mathcal{K}}$ *may be involved in the definition of* $g$*.*

**Definition 12** *Assume* $F$ *is a class of functions: a function* $f$ *is in* $\mathbf{\exists}F$ *if it is defined with one predicative minimization over a function* $h$ *of* $F$*.*

*We define by induction the following sets:*

- $F^0_{\mathcal{K}} = \mathrm{SR}_{\mathcal{K}}$.
- $F^{i+1}_{\mathcal{K}} = RSR_{\mathcal{K}}(F^i_{\mathcal{K}} \bigcup \mathbf{\exists}F^i_{\mathcal{K}})$*, for* $i \geq 0$.

*We denote by* $\mathbf{\exists}\mathrm{PH}_{\mathcal{K}} = \bigcup_{i\in\mathbb{N}} F^i_{\mathcal{K}}$ *the closure of the basic safe functions over* $\mathcal{K}$ *under the application of restricted safe recursion, predicative minimization and safe composition.*

### 6.6.3 Characterization of the Polynomial Hierarchy

We proved the following result in papers [Bournez et al., 2003], [Bournez et al., 2005a].

**Theorem 4** *A function:* $(\mathbb{K}^*)^n \times \emptyset \to \mathbb{K}^*$ *belongs to* $\mathrm{F}\Delta^i_{\mathcal{K}}$ *if and only if it is defined in* $\mathrm{F}^i_{\mathcal{K}}$.

**Corollary 1** *A decision problem over* $\mathcal{K}$ *belongs to* $\mathrm{PH}_{\mathcal{K}}$ *if and only if its characteristic function is defined in* $\mathbf{\exists}\mathrm{PH}_{\mathcal{K}}$.

## 6.7 Digital Polynomial Hierarchy

### 6.7.1 Definitions

In this digital version of the polynomial hierarchy, witnesses for a given problem are discrete choices among given values, and not arbitrary elements of the structure. As in the previous section, complete problems have been shown to exist for every level of this hierarchy [Blum et al., 1998].

**Definition 13** *A set $S \subseteq \mathbb{K}^*$ belongs to* $\mathrm{DNP}_{\mathcal{K}}$ *if and only if there exist a polynomial $p$ and a polynomial time BSS machine $M$ over $\mathcal{K}$ such that, for all $\overline{x} \in \mathbb{K}^*$,*

$$\overline{x} \in S \Leftrightarrow \exists \overline{y} \in \{\mathbf{0},\mathbf{1}\}^* \text{ s.t. } |\overline{y}| \leq p(|\overline{x}|) \text{ and } M \text{ accepts } (\overline{x},\overline{y}).$$

*Let* $\mathrm{D}\Sigma^0_{\mathcal{K}} = \mathrm{P}_{\mathcal{K}}$ *and, for* $i \geq 1$, $\mathrm{D}\Sigma^i_{\mathcal{K}} = \mathrm{DNP}^{\mathrm{D}\Sigma^{i-1}_{\mathcal{K}}}_{\mathcal{K}}$, $\mathrm{D}\Pi^i_{\mathcal{K}} = \mathrm{coDNP}^{\mathrm{D}\Sigma^{i-1}_{\mathcal{K}}}_{\mathcal{K}}$.

*The* digital polynomial time hierarchy *is* $\mathrm{DPH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{D}\Sigma^i_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{D}\Pi^i_{\mathcal{K}}$.

*A function is in* $\mathrm{DF}\Delta^i_{\mathcal{K}}$ *if it computable in polynomial time by a machine over $\mathcal{K}$ which queries an oracle in* $\mathrm{D}\Sigma^i_{\mathcal{K}}$. *That is,* $\mathrm{DF}\Delta^i_{\mathcal{K}} = \mathrm{FP}^{\mathrm{D}\Sigma^i_{\mathcal{K}}}_{\mathcal{K}} = \mathrm{FP}^{\mathrm{D}\Pi^i_{\mathcal{K}}}_{\mathcal{K}}$.

*The* functional digital polynomial time hierarchy *is* $\mathrm{DFPH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{DF}\Delta^i_{\mathcal{K}}$.

### 6.7.2 Safe Recursion with Digital Predicative Minimization

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

**Definition 14** *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 *as follows*

$$f(\overline{x};\overline{a}) = \mathbf{\exists}_{\mathrm{D}}\overline{b}(h(\overline{x};\overline{a},\overline{b})) = \begin{cases} \mathbf{1} & \text{if there exists } \overline{b} \in \{\mathbf{0},\mathbf{1}\}^* \text{ such that } h(\overline{x};\overline{a},\overline{b}) = \mathbf{0} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

**Definition 15** *Let $F$ be a class of functions. A function $f$ is in $\mathbf{\exists}_{\mathrm{D}}F$ if it is defined with one predicative minimization over a function $h$ of $F$.*

*We define by induction the following sets:*

- $dF^0_{\mathcal{K}} = \mathrm{SR}_{\mathcal{K}}$
- $dF^{i+1}_{\mathcal{K}} = RSR_{\mathcal{K}}(dF^i_{\mathcal{K}} \bigcup \mathbf{\exists}_{\mathrm{D}} F^i_{\mathcal{K}})$, *for* $i \geq 0$.

*We denote by $\mathbf{\exists}_{\mathrm{D}}\mathrm{PH}_{\mathcal{K}}$ the closure of the basic safe functions over $\mathcal{K}$ under the application of projections, restricted safe recursion, digital predicative minimization and safe composition.*

### 6.7.3 Characterization of Polynomial Digital Hierarchy

We proved the following result in [Bournez et al., 2003], [Bournez et al., 2005a].

**Theorem 5** *A function: $(\mathbb{K}^*)^n \times \emptyset \to \mathbb{K}^*$ belongs to* $\mathrm{DF}\Delta^i_{\mathcal{K}}$ *if and only if it is defined in $dF^i_{\mathcal{K}}$.*

**Corollary 2** *A decision problem over $\mathcal{K}$ belongs to* $\mathrm{DPH}_{\mathcal{K}}$ *if and only if its characteristic function is defined in* $\mathbf{\exists}_{\mathrm{D}}\mathrm{DPH}_{\mathcal{K}}$.

When considering finite structures, this naturally yields a characterization of the classical polynomial hierarchy alternative to the one found in [Bellantoni, 1994].

**Corollary 3** *A decision problem belongs to* PH *if and only if its characteristic function is defined in* $\mathbf{\exists}_{\mathrm{D}}\mathrm{DPH}_{\{\mathbf{0},\mathbf{1}\}}$.

## 6.8 Polynomial Alternating Time

### 6.8.1 Definitions

**Definition 16** *A set $S \subseteq \mathbb{K}^*$ belongs to* $\mathrm{PAT}_{\mathcal{K}}$ *(*polynomial alternating time*) if and only if there exist a polynomial function $q : \mathbb{N} \to \mathbb{N}$ and a polynomial time BSS machine $M_S$ over $\mathcal{K}$ such that, for all $\overline{x} \in \mathbb{K}^*$,*

$$\begin{aligned} \overline{x} \in S \quad \Leftrightarrow \quad & \exists a_1 \in \mathbb{K}\, \forall b_1 \in \mathbb{K}\, \ldots\, \exists a_{q(|\overline{x}|)} \in \mathbb{K}\, \forall b_{q(|\overline{x}|)} \in \mathbb{K} \\ & M_S \text{ accepts } (\overline{x}, a_1.b_1 \ldots a_{q(|\overline{x}|)}.b_{q(|\overline{x}|)}). \end{aligned}$$

*In addition, we define* $\mathrm{FPAT}_{\mathcal{K}} = \mathrm{FP}_{\mathcal{K}}^{\mathrm{PAT}_{\mathcal{K}}}$.

When $\mathcal{K}$ is the structure $\{\{0,1\}, =, \mathbf{0}, \mathbf{1}\}$, $\mathrm{PAT}_{\mathcal{K}}$ is PSPACE.

It is important to note that the number of quantifier alternations is not fixed, but depends on the length of the input and is polynomial in that length. It follows that $\mathrm{PH}_{\mathcal{K}} \subseteq \mathrm{PAT}_{\mathcal{K}}$.

### 6.8.2 Safe Recursion with Predicative Substitutions

**Definition 17** *Given $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, we define $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}$ by* predicative substitution *as follows*

$$f(\overline{x}; \overline{a}) = \mathfrak{Z}^{[1]} c(h(\overline{x}; \overline{a}, c)) = \begin{cases} \mathbf{1} & \text{if there exists } c \in \mathbb{K} \text{ such that } h(\overline{x}; \overline{a}, c) = \mathbf{0} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

**Definition 18** *Assume $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ and $g : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ are given functions. The function $f : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ is defined by* safe recursion with predicative substitutions *as follows*

$$\begin{aligned} f(\epsilon, \overline{x}; \overline{u}, \overline{y}) &= h(\overline{x}; \overline{u}, \overline{y}) \\ f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) &= g(\overline{z}, \overline{x}; \mathfrak{Z}^{[1]} c(f(\overline{z}, \overline{x}; c.\overline{u}, \overline{y})), \overline{y}). \end{aligned}$$

**Definition 19** *The set $\mathfrak{Z}^{[1]}\mathrm{PAT}_{\mathcal{K}}$ of* safe recursive functions with predicative substitutions *over $\mathcal{K}$ is the closure of the basic safe functions under the application of safe composition, safe recursion and safe recursion with predicative substitutions.*

### 6.8.3 Characterization of Polynomial Alternating Time

We proved the following result in papers [Bournez et al., 2004b], [Bournez et al., 2006].

**Theorem 6** *Over any structure*

$$\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i \in I}, \mathbf{0}, \mathbf{1}),$$

*A function is computed in* $\mathrm{FPAT}_{\mathcal{K}}$ *if and only if it can be defined in* $\mathfrak{Z}^{[1]}\mathrm{PAT}_{\mathcal{K}}$.

## 6.9 Digital Polynomial Alternating Time

### 6.9.1 Definitions

The class $\mathrm{DPAT}_{\mathcal{K}}$ is defined similarly to $\mathrm{PAT}_{\mathcal{K}}$ but with all quantified variables belonging to $\{\mathbf{0}, \mathbf{1}\}$. Similarly, we can define $\mathrm{DFPAT}_{\mathcal{K}} = \mathrm{FP}_{\mathcal{K}}^{\mathrm{DPAT}_{\mathcal{K}}}$.

### 6.9.2 Safe Recursion with Digital Predicative Substitutions

Similarly to the notion of predicative substitution, we define the notion of digital predicative substitution.

**Definition 20** *Given $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$, we define $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}$ by* predicative substitution*,*

$$f(\overline{x};\overline{a}) = \mathfrak{Z}_{\mathsf{D}}^{[1]} c(h(\overline{x};\overline{a},c)) = \begin{cases} \mathbf{1} & \text{if there exists } c \in \{\mathbf{0},\mathbf{1}\} \text{ such that } h(\overline{x};\overline{a},c) = \mathbf{0} \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

**Definition 21** *Assume $h : \mathbb{K}^* \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ and $g : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ are given functions. The function $f : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ is defined by* safe recursion with digital predicative substitutions *as follows*

$$\begin{aligned} f(\epsilon,\overline{x};\overline{u},\overline{y}) &= h(\overline{x};\overline{u},\overline{y}) \\ f(a.\overline{z},\overline{x};\overline{u},\overline{y}) &= g(\overline{z},\overline{x};\mathfrak{Z}_{\mathsf{D}}^{[1]} c f(\overline{z},\overline{x};c.\overline{u},\overline{y}),\overline{y}). \end{aligned}$$

**Definition 22** *The set $\mathfrak{Z}_{\mathsf{D}}^{[1]}\mathrm{PAT}_{\mathcal{K}}$ of* safe recursive functions with digital predicative substitutions *over $\mathcal{K}$ is the closure of the basic safe functions under the application of safe composition, safe recursion and safe recursion with digital predicative substitutions.*

### 6.9.3 Characterization of Digital Polynomial Alternating Time

We proved the following in papers [Bournez et al., 2004b], [Bournez et al., 2006].

**Theorem 7** *Over any structure*

$$\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i\in I}, \mathbf{0}, \mathbf{1}),$$

*A function is computed in* $\mathrm{DFPAT}_{\mathcal{K}}$ *if and only if it can be defined in* $\mathfrak{Z}^{[1]}\mathrm{DPAT}_{\mathcal{K}}$*.*

When restricted to finite structures, this yields another characterization of PSPACE.

**Corollary 4** *A set $S \subset \{0,1\}^*$ is in* PSPACE *if and only if its characteristic function can be defined in* $\mathfrak{Z}^{[1]}\mathrm{DPAT}_{\{\mathbf{0},\mathbf{1}\}}$*.*

### 6.9.4 An Alternative Characterization

An alternative characterization is possible, based on our previous characterization of $\mathrm{PAR}_{\mathcal{K}}$. Some small restrictions on the type of the functions involved in the recursion scheme yield another characterization of $\mathrm{DPAT}_{\mathcal{K}}$.

**Definition 23** *We call* pseudo logical function *any function in the closure of operations of arity 0 (constants), projections and the selector function* Select *under the application of safe composition.*

Since no recursion and no $\mathsf{tl}$ function is involved in the definition of a pseudo logical function, its output depends only on the value of the first letter of its arguments, more precisely, on whether these are **1** or not since no relation is allowed either.

**Definition 24** *Assume* $h : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ *is a given function,* $g : (\mathbb{K}^*)^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ *is a pseudo logical function, and* $\sigma_1, \sigma_2 : \emptyset \times \mathbb{K}^* \to \mathbb{K}^*$ *are safe recursive functions. Function* $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ *can then be defined by* safe recursion with digital substitutions*:*

$$\begin{aligned} f(\epsilon, \overline{x}; \overline{u}) &= h(\overline{x}; \overline{u}) \\ f(a.\overline{z}, \overline{x}; \overline{u}) &= g(\overline{z}, \overline{x}; f(\overline{z}, \overline{x}; \sigma_1(;\overline{u})), f(\overline{z}, \overline{x}; \sigma_2(;\overline{u}))). \end{aligned}$$

We define the set of *safe recursive functions with digital substitutions* to be the closure of the basic safe functions under the application of safe composition, safe recursion and safe recursion with digital substitutions.

An alternative characterization is given by the following theorem presented in [Bournez et al., 2004b], [Bournez et al., 2006].

**Theorem 8** *Over any structure*

$$\mathcal{K} = (\mathbb{K}, op_1, \cdots, op_k, rel_1, \cdots, rel_l, \{c_i\}_{i \in I}, \mathbf{0}, \mathbf{1}),$$

*a function is in* $\mathrm{DFPAT}_{\mathcal{K}}$ *if and only if it can be defined as a safe recursive functions with digital substitutions over* $\mathcal{K}$*.*

When $\mathcal{K}$ is a finite structure, i.e., when considering classical complexity, this characterization coincides with our previous characterization of $\mathrm{PAR}_{\mathcal{K}}$ in [Bournez et al., 2003], and captures PSPACE.

# Bibliography


[Abiteboul et al., 1990] Abiteboul, S., Simon, E., and Vianu, V. (1990). Non-deterministic languages to express deterministic transformations. In ACM, editor, *PODS '90. Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems: April 2–4, 1990, Nashville, Tennessee*, volume 51(1) of *Journal of Computer and Systems Sciences*, pages 218–229, New York, NY 10036, USA. ACM Press.

[Abiteboul and Vianu, 1989] Abiteboul, S. and Vianu, V. (1989). Fixpoint extensions of first-order logic and Datalog-like languages. In *Proceedings 4th Annual IEEE Symposium on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–9 June 1989*, pages 71–79. IEEE Computer Society Press, Los Alamitos, CA.

[Abiteboul and Vianu, 1991] Abiteboul, S. and Vianu, V. (1991). Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124.

[Bellantoni, 1994] Bellantoni, S. (1994). Predicative recursion and the polytime hierarchy. In Clote, P. and Remmel, J., editors, *Feasible Mathematics II*, *Perspectives in Computer Science*. Birkhäuser.

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

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

[Blum et al., 1989] Blum, L., Shub, M., and Smale, S. (1989). 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):1–46.

[Bonner, 1989] Bonner, A. J. (1989). Hypothetical Datalog: Negation and linear recursion. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 286–300, Philadelphia, Pennsylvania. ACM Press.

[Bournez et al., 2003] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2003). Computability over an arbitrary structure. sequential and parallel polynomial time. In Gordon, A. D., editor, *Foundations of Software Science and Computational Structures, 6th International Conference (FOSSACS'2003)*, volume 2620 of *Lecture Notes in Computer Science*, pages 185–199, Warsaw. Springer.

[Bournez et al., 2004a] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2004a). Tailoring recursion to characterize non-deterministic complexity classes over arbitrary structures. In *In 2nd APPSEM II Workshop (APPSEM'04)*.

[Bournez et al., 2004b] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2004b). Tailoring recursion to characterize non-deterministic complexity classes over arbitrary structures. In *3rd IFIP International Conference on Theoretical Computer Science - TCS'2004*, Toulouse, France. Kluwer Academic Press.

[Bournez et al., 2005a] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2005a). Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time. *Journal of Logic and Computation*, 15(1):41–58.

[Bournez et al., 2005b] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2005b). Logical characterizations of $p_{\mathcal{K}}$ and $np_{\mathcal{K}}$ over an arbitrary structure $k$. In *3rd APPSEM II Workshop (APPSEM'05), Frauenchiemsee, Germany, 2005. Also accepted for presentation at CIE 2005: New Computational Paradigms.*

[Bournez et al., 2006] Bournez, O., Cucker, F., de Naurois, P. J., and Marion, J.-Y. (2006). Implicit complexity over an arbitrary structure: Quantifier alternations. *Information and Computation*, 202(2):210–230.

[Bournez et al., 2002] Bournez, O., de Naurois, P., and Marion, J.-Y. (2002). Safe recursion and calculus over an arbitrary structure. In *Implicit Computational Complexity - ICC'02*, Copenhagen, Denmark.

[Clote, 1998] Clote, P. (1998). Computational models and function algebras. In Griffor, E. R., editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam.

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

[Cucker, 1993] Cucker, F. (1993). On the complexity of quantifier elimination: the structural approach. *The Computer Journal*, 36:400–408.

[Cucker and Meer, 1999] Cucker, F. and Meer, K. (1999). Logics which capture complexity classes over the reals. *Journal of Symbolic Logic*, 64(1):363–390.

[Ebbinghaus and Flum, 1995] Ebbinghaus, H. and Flum, J. (1995). *Finite Model Theory*. Perspectives in Mathematical Logic, Omega Series. Springer–Verlag, Heidelberg.

[Fagin, 1974] Fagin, R. (1974). Generalized first-order spectra and polynomial-time recognizable sets. In Karp, R. M., editor, *Complexity in Computer Computations*, pages 43–73. American Mathematics Society, Providence R.I.

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

[Grädel and Gurevich, 1998] Grädel, E. and Gurevich, Y. (1998). Metafinite model theory. *Information and Computation*, 140(1):26–81.

[Grädel and Meer, 1995] Grädel, E. and Meer, K. (1995). 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. ACM Press.

[Grzegorczyk, 1957] Grzegorczyk, A. (1957). On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71.

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

[Gurevich and Shelah, 1986] Gurevich, Y. and Shelah, S. (1986). Fixed-point extensions of first order logic. *Annals of Pure and Applied Logic*, 32:265–280.

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

[Immerman, 1986] Immerman, N. (1986). Relational queries computable in polynomial time. *Information and Control*, 68(1–3):86–104.

[Immerman, 1987] Immerman, N. (1987). Languages that capture complexity classes. *SIAM Journal of Computing*, 16(4):760–778.

[Immerman, 1991] Immerman, N. (1991). $DSPACE[n^k] = VAR[k+1]$. In Balcázar, José; Borodin, Alan; Gasarch, Bill; Immerman, Neil; Papadimitriou, Christos; Ruzzo, Walter; Vitányi, Paul; Wilson, C., editor, *Proceedings of the 6th Annual Conference on Structure in Complexity Theory (SCTC '91)*, pages 334–340, Chicago, IL, USA. IEEE Computer Society Press.

[Immerman, 1999] Immerman, N. (1999). *Descriptive Complexity*. Springer.

[Jones, 2001] Jones, N. D. (2001). The expressive power of higher-order types or, life without CONS. *Journal of Functionnal Programming*, 11(1):5–94.

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

[Lacombe, 1955] Lacombe, D. (1955). Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles iii. *Comptes Rendus de l'Académie des Sciences Paris*, 241:151–153.

[Leivant, 1990] Leivant, D. (1990). Inductive definitions over finite structures. *Information and Computation*, 89(2):95–108.

[Leivant and Marion, 1995] Leivant, D. and Marion, J.-Y. (1995). Ramified recurrence and computational complexity II: substitution and poly-space. In Pacholski, L. and Tiuryn, J., editors, *Computer Science Logic, 8th Workshop, CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500, Kazimierz,Poland. Springer.

[Michaux, 1989] Michaux, C. (1989). Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *Comptes Rendus de l'Académie des Sciences de Paris*, 309:435–437.

[Moschovakis, 1984] Moschovakis, Y. N. (1984). Abstract recursion as a foundation for the theory of algorithms. In *Computation and Proof Theory*, volume 1104 of *Lecture Notes in Mathematics*, pages 289–364, Berlin. Springer-Verlag.

[Poizat, 1995] Poizat, B. (1995). *Les petits cailloux*. aléas.

[Turing, 1936] Turing, A. (1936). On computable numbers, with an application to the Ëntscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265.

[Vardi, 1982] Vardi, M. Y. (1982). The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing (STOC)*, pages 137–146. ACM Press.

[Weihrauch, 2000] Weihrauch, K. (2000). *Computable Analysis*. Springer.