Computability over an Arbitrary Structure. Sequential and Parallel Polynomial Time

Olivier Bournez[1], Felipe Cucker[2], Paulin Jacobé de Naurois[1]*, and Jean-Yves Marion[1]

[1] LORIA,
615 rue du Jardin Botanique, BP 101,
54602 Villers-lès-Nancy Cedex, Nancy, France.
{Olivier.Bournez,Paulin.De-Naurois,Jean-Yves.Marion}@loria.fr
[2] City University of Hong Kong,
Tat Chee avenue,
Kowloon, Hong Kong.
macucker@math.cityu.edu.hk

Abstract. We provide several machine-independent characterizations of deterministic complexity classes in the model of computation proposed by L. Blum, M. Shub and S. Smale. We provide a characterization of partial recursive functions over any arbitrary structure. We show that polynomial time computable functions over any arbitrary structure can be characterized in term of safe recursive functions. We show that polynomial parallel time decision problems over any arbitrary structure can be characterized in terms of safe recursive functions with substitutions.

1 Introduction

Why are we convinced by the Church Thesis? An answer is certainly that there are so many mathematical models, like partial recursive functions, lambda-calculus, or semi-Thue systems, which are equivalent to Turing machine, but are also independent from the computational machinery. When computing over arbitrary structures, like real numbers, the situation is not so clear. Seeking machine independent characterizations of complexity classes can lend further credence to the importance of the classes and models considered.

We consider here the BSS model of computation over the real numbers introduced by Blum, Shub and Smale in their seminal paper [BSS89]. The model was later on extended to a computational model over any arbitrary logical structure [Goo94, Poi95]. See the monograph [BCSS98] for a general survey about the BSS model.

First of all, we present a new characterization of computable functions that extends the one of [BSS89] to any arbitrary structure.

* This author has been partially supported by City University of Hong Kong SRG grant 7001290.

Theorem 1. *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \ldots, op_k, =, rel_1, \ldots, rel_l, \alpha)$, the set of partial recursive functions over $\mathcal{K}$ is exactly the set of decision functions computed by a BSS machine over $\mathcal{K}$.*

In the BSS model, complexity classes like PTIME and NPTIME can be defined, and complete problems in these classes can be shown to exist. On many aspects, this is an extension of the classical complexity theory since complexity classes correspond to classical complexity classes when dealing with booleans or integers. The next results strengthen this idea.

In classical complexity theory, several attempts have been done to provide nice formalisms to characterize complexity classes in a machine independent way. Such characterizations include descriptive characterization based on finite model theory like Fagin [Fag74], characterization by function algebra like [Cob62], or by combining both kinds of characterization like in [Gur83, Saz80]: see [Clo95, Imm99, EF95] for more complete references.

Despite the success of those approaches to capture major complexity classes, one may not be completely satisfied because explicit upper bounds on computational resources or restrictions on the growth rates are present. The recent works of Bellantoni and Cook [BC92] and of Leivant [Lei95, LM93] suggests another direction by mean of data tiering which is called implicit computational complexity. There are no more explicit resource bounds. It provides purely syntactic models of complexity classes which can be applied to programming languages to analyze program complexity [Jon00, Hof99, MM00].

In this paper, following these lines, we establish two "implicit" characterizations of the complexity classes. Our characterizations work over arbitrary structures, and subsume previous ones when restricted to booleans or integers.

First, we characterize polynomial time computable BSS functions. This result stems on the safe primitive recursion principle of Bellantoni and Cook [BC92].

Theorem 2. *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \ldots, op_k, =, rel_1, \ldots, rel_l, \alpha)$, the set of safe recursive functions over $\mathcal{K}$ is exactly the set of functions computed in polynomial time by a BSS machine over $\mathcal{K}$.*

Second, we capture parallel polynomial time BSS functions based on Leivant and Marion characterization of polynomial space computable functions [LM95].

Theorem 3. *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \ldots, op_k, =, rel_1, \ldots, rel_l, \alpha)$, 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 $\mathcal{K}$.*

Observe that, unlike Leivant and Marion, our proofs characterize 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 is meaningless: as pointed out by Michaux [Mic89], on some structures like $(\mathbb{R}, 0, 1, =, +, -, *)$, any computable function can be computed in constant working space.

From a programming perspective, a way of understanding all these 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 able to build methods to derive automatically computational properties of programs, in the lines of [Jon00, Hof99, MM00], is one of our main motivations on making this research.

On the other hand, we believe BSS computational model to provide new insights for understanding complexity theory when dealing with structures over other domains [BCSS98]: several nice results have been obtained for this model in the last decade, including separation of complexity classes over specific structures: see for example [Mee92, Cuc92, CSS94]. We believe these results to contribute to the understanding of complexity theory, even when restricted to classical complexity [BCSS98].

It is worth mentioning that it is not the first time that the implicit computational complexity community is interested by computations over real numbers: see the exciting paper of Cook [Coo92] on higher order functionals, or the works trying to clarify the situation such as [RIR01]. However this is the first time that implicit characterizations of this type over arbitrary structures are given.

In Section 2, we give a characterization of primitive recursive functions over an arbitrary structure. We introduce our notion of safe recursive function in Section 3. We give the proof of Theorem 2 in Section 4. We recall the notion of a family of circuits over an arbitrary structure in Section 5. Safe recursion with substitutions is defined in Section 5.2. Theorem 3 is proved in Section 5.3 and 5.4.

2 Partial Recursive and Primitive Recursive Functions

2.1 Definitions

A structure $\mathcal{K} = (\mathbb{K}, op_1, \ldots, op_k, rel_1, \ldots, rel_l, \alpha)$ is given by some underlying set $\mathbb{K}$, some operators $op_1, \ldots, op_k$ with arities, and some relations $rel_1, \ldots, rel_l$ with arities. Constants correspond to operators of arity 0. 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 equality relation $=$ is always one relation of the structure, and that there is at least one constant α in the structure. A good example for such a structure, corresponding to the original paper in [BSS89] is $\mathcal{K} = (\mathbb{R}, +, -, *, =, \leq, 0, 1)$. Another one, corresponding to classical complexity and computability theory, is $\mathcal{K} = (\{0,1\}, \vee, \wedge, =, 0, 1)$.

$\mathbb{K}^* = \bigcup_{i \in \mathbb{N}} \mathbb{K}^i$ will denote the set of words over alphabet $\mathbb{K}$. In our notations, words of elements in $\mathbb{K}$ will be represented with overlined letters, while simple elements in $\mathbb{K}$ will be represented by simple letters. For instance, $a.\overline{x}$ stands for the word in $\mathbb{K}^*$ whose first letter is a and which ends with the word $\overline{x}$. ϵ will denote the empty word. The length of a word $\overline{w} \in \mathbb{K}^*$ is denoted by $|\overline{w}|$.

We assume that the reader has some familiarities with the computation model of Blum Shub and Smale: see [BSS89, BCSS98] for a detailed presentation.

In particular remember that a problem $P \subset \mathbb{K}^*$ is decidable (respectively a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is computable), if there exists a machine M over structure $\mathcal{K}$ that decides P (resp. computes f). A problem $P \subset \mathbb{K}^*$ is in the class PTIME (respectively a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is in the class FPTIME), if there exists a polynomial p and a machine M over structure $\mathcal{K}$ that decides P (resp. computes f) in time p.

As for the classical settings, computable functions over any 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 that works over any arbitrary structure. See comments below for comparisons with the one, for the structure of real numbers, in the original paper [BSS89].

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 $\perp$.

Theorem 1. *Over any structure $\mathcal{K} = (\mathbb{K}, op_1, \ldots, op_k, =, rel_1, \ldots, rel_l, \alpha)$, the set of functions $(\mathbb{K}^*)^n \to \mathbb{K}^*$ computed by a BSS machine over $\mathcal{K}$ is exactly the set of partial recursive functions, that is to say the smallest set of functions containing the basic functions, and closed under the operations of composition, primitive recursion, and minimization defined below.*

The basic functions are of four kinds:

– functions making elementary manipulations of words of elements in $\mathbb{K}$. For any $a \in \mathbb{K}, \overline{x}, \overline{x_1}, \overline{x_2} \in \mathbb{K}^*$:

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

– Projections: for any $n \in \mathbb{N}$, $\imath \leq n$:

$$\mathrm{Pr}^n_\imath(\overline{x_1}, \ldots, \overline{x_\imath}, \ldots, \overline{x_n}) = \overline{x_\imath}$$

– functions of structure $\mathcal{K}$: for any operator (including the constants treated as operators of arity 0) $op_\imath$ or relation $rel_\imath$ of arity $n_\imath$ we have the following initial functions:

$$\mathrm{Op}_\imath(a_1.\overline{x_1}, \ldots, a_{n_\imath}.\overline{x_{n_\imath}}) = (op_\imath(a_1, \ldots, a_{n_\imath})).\overline{x_{n_\imath}}$$
$$\mathrm{Rel}_\imath(a_1.\overline{x_1}, \ldots, a_{n_\imath}.\overline{x_{n_\imath}}) = \begin{cases} \alpha \text{ if } rel_\imath(a_1, \ldots, a_{n_\imath}) \\ \epsilon \text{ otherwise} \end{cases}$$

– test function :

$$\mathrm{C}(\overline{x}, \overline{y}, \overline{z}) = \begin{cases} \overline{y} & \text{if } \mathrm{hd}(\overline{x}) = \alpha \\ \overline{z} & \text{otherwise} \end{cases}$$

Operations mentioned above are:

– composition: Assume f: $(\mathbb{K}^*)^n \to \mathbb{K}^*$, $g_1, \ldots, g_n$: $(\mathbb{K}^*)^m \to \mathbb{K}^*$ are given partial functions. Then the composition h: $(\mathbb{K}^*)^m \to \mathbb{K}^*$ is defined by

$$h(\overline{x_1}, \ldots, \overline{x_m}) = f(g_1(\overline{x_1}, \ldots, \overline{x_m}), \ldots, g_n(\overline{x_1}, \ldots, \overline{x_m}))$$

– 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: $(\mathbb{K}^*)^2 \to \mathbb{K}^*$

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

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

$$\mu\overline{x}\,(f(\overline{x}, \overline{y})) = \begin{cases} \perp & \text{if } \forall t \in \mathbb{N} : \text{hd}(f(0^t, \overline{y})) \neq \alpha \\ \alpha^k : \ k = \min\{t \mid \text{hd}(f(0^t, \overline{y})) = \alpha\} & \text{otherwise} \end{cases}$$

Note 1. Our definition of primitive recursion and of minimization is sightly different from the one found in [BSS89]. 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: $\mathbb{N} * \mathbb{K}^k \to \mathbb{K}^l$. Therefore, they only capture finite dimensional functions. It is known that, on the real numbers with $+, -, *$ operators, finite dimensional functions are equivalent to non-finite dimensional functions (see [Mic89]), but this is not true over other structures, for instance $\mathbb{Z}_2$. 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,. We consider it to be a more general and a more natural way to define computable functions, and moreover not restricted to structure $\mathcal{K} = (\mathbb{R}, +, -, *, =, \leq, 0, 1)$.

Observe that primitive recursion can be replaced by simultaneous primitive recursion:

Proposition 1. [BMdN02] *Simultaneous primitive recursion is definable with primitive recursive functions.*

The proof of Theorem 1, similar to the proof of Theorem 2 in section 3, is not given here.

3 Safe Recursive Functions

In this section we define the set of safe recursive functions over any arbitrary structure $\mathcal{K}$, extending the notion of safe recursive functions over the natural numbers found in [BC92].

Safe recursive functions are defined in a quite similar manner as primitive recursive functions. However, in the spirit of [BC92], safe recursive functions have two different types of arguments, each of which having different properties and different purposes. The first type of argument, called "normal" arguments, 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 is called "safe", and 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.

We will use the following notations: the two different types of arguments are separated by a semicolon ";" : normal arguments (respectively: safe arguments) are placed at left (resp. at right) of the semicolon.

We define now safe recursive functions:

Definition 1. *The set of safe recursive functions over $\mathbb{K}$ is the smallest set of functions: $(\mathbb{K}^*)^k \to \mathbb{K}^*$, containing the basic safe functions, and closed under the operations of safe composition and safe recursion*

Basic safe functions are the basic functions of Section 2, their arguments defined all as safe.

Operations mentioned above are:

- safe composition: Assume f: $(\mathbb{K}^*)^m \times (\mathbb{K}^*)^n \to \mathbb{K}^*$, $g_1, g_m : (\mathbb{K}^*)^p \to \mathbb{K}^*$ and $g_{m+1}, g_{m+n} : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to \mathbb{K}^*$ are given functions. Then the composition is the function $h : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to \mathbb{K}^*$:

$$\begin{aligned} h(\overline{x_1}, \ldots, \overline{x_p}; \overline{y_1}, \ldots, \overline{y_q}) = f\,(&g_1(\overline{x_1}, \ldots, \overline{x_p}), \ldots, g_m(\overline{x_1}, \ldots, \overline{x_p}); \\ &g_{m+1}(\overline{x_1}, \ldots, \overline{x_p}; \overline{y_1}, \ldots, \overline{y_q}), \ldots, g_{m+1}(\overline{x_1}, \ldots, \overline{x_p}; \overline{y_1}, \ldots, \overline{y_q})) \end{aligned}$$

  Note 2. It is possible to move an argument from the normal position to the safe position, whereas the reverse is forbidden. By "move", we mean the following: 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: $f(\overline{x}, \overline{y}; \overline{z}) = g(\overline{x}; \overline{y}, \overline{z})$ but a definition like the following is not valid: $f(\overline{x}; \overline{y}, \overline{z}) = g(\overline{x}, \overline{y}; \overline{z})$.

- 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:

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

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

$$\vdots$$

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

Note 3. The operation of primitive recursion previously defined is a simple recursion, whereas the operation of safe recursion is a simultaneous recursion. As stated by Proposition 1, it is possible to simulate a simultaneous primitive recursion with single primitive recursion, whereas this does not seem to be true with safe recursion. As shown in the simulation of a BSS machine by safe recursive functions, we need to have a simultaneous recursion able to define simultaneously three functions in order to prove Theorem 2. In the classical setting, this is the choice made by Leivant and Marion in [LM95], while Bellantoni and cook used a smash function # to build and break t-uples [BC92]. Both choices are equivalent.

4 Proof of Theorem 2

Theorem 2 is proved in two steps. First, we prove that a safe recursive function can be computed by a BSS machine in polynomial time. Second, we prove that all functions computable in polynomial time by a BSS machine over $\mathbb{K}$ can be expressed as safe recursive functions.

4.1 Polynomial Time Evaluation of a Safe Recursive Function

This is a straightforward consequence of the following lemma.

Lemma 1. *Let $f(\overline{x_1},\ldots,\overline{x_n};\overline{y_1},\ldots,\overline{y_m})$ be any safe recursive function. If we write $T\ulcorner f(\ldots)\urcorner$ for the evaluation time of $f(\ldots)$,*

$$T\ulcorner f(\overline{x_1},\ldots,\overline{x_n};\overline{y_1},\ldots,\overline{y_m})\urcorner \leq p_f(T\ulcorner\overline{x_1}\urcorner+\ldots+T\ulcorner\overline{x_n}\urcorner)+T\ulcorner\overline{y_1}\urcorner+\ldots+T\ulcorner\overline{y_m}\urcorner$$

for some polynomial p_f.

This is proved by induction on the depth of the definition tree of the safe recursive function. Let f be a safe recursive function.

- If f is a basic safe function, the result is straightforward.
- if f is defined by safe composition from safe recursive functions g, h_1, h_2, using induction hypothesis, f is easily shown to be computable in polynomial time by a BSS machine.
- The non-trivial case is the case of a function f defined with simultaneous safe recursion. In order to simplify the notations, we assume that f is defined with a single safe recursion. The proof is in essence the same.

  Let us apply induction hypothesis to function g in the expression $f(a.\overline{z},\overline{x};\overline{y}) = g(\overline{z},\overline{x};f(\overline{z},\overline{x};\overline{y}),\overline{y})$. Assuming a "clever" strategy, $\overline{y}$ needs to be evaluated only once, even though it appears in several recursive calls. Thus, if we assume that $\overline{y}$ has already been evaluated, the time needed to evaluate $f(a.\overline{z},\overline{x};\overline{y}) = g(\overline{z},\overline{x};f(\overline{z},\overline{x};\overline{y}),\overline{y})$ is given by $p_g(T\ulcorner\overline{z}\urcorner+T\ulcorner\overline{x}\urcorner)+T\ulcorner f(\overline{z},\overline{x};\overline{y})\urcorner$.

We get:

$$\begin{aligned} & T^\ulcorner f(a.\overline{z},\overline{x};\overline{y})^\urcorner \\ \le\ & T^\ulcorner \overline{y}^\urcorner + p_g(T^\ulcorner \overline{z}^\urcorner + T^\ulcorner \overline{x}^\urcorner) + T^\ulcorner f(\overline{z},\overline{x};\overline{y})^\urcorner \\ \le\ & T^\ulcorner \overline{y}^\urcorner + p_g(T^\ulcorner \overline{z}^\urcorner + T^\ulcorner \overline{x}^\urcorner) + p_g(T^\ulcorner \overline{\mathrm{tl}(z)}^\urcorner + T^\ulcorner \overline{x}^\urcorner) + \ldots \\ & \qquad \ldots + p_g(T^\ulcorner \overline{\epsilon}^\urcorner + T^\ulcorner \overline{x}^\urcorner) + p_h(T^\ulcorner \overline{x}^\urcorner)\,. \end{aligned}$$

Assuming without loss of generality p_g monotone, we get

$$T^\ulcorner f(a.\overline{z},\overline{x};\overline{y})^\urcorner \le |a.\overline{z}| p_g(T^\ulcorner \overline{z}^\urcorner + T^\ulcorner \overline{x}^\urcorner) + p_h(T^\ulcorner \overline{x}^\urcorner) + T^\ulcorner \overline{y}^\urcorner,$$

from which the lemma follows.

4.2 Simulation of a Polynomial Time BSS Machine

Let M be a BSS machine over the structure $\mathcal{K}$. In order to simplify our exposition we assume, without any loss of generality, that M has a single tape. M computes a partial function f_M from $\mathbb{K}^*$ to $\mathbb{K}^*$. Moreover, we assume that M stops after $c|\overline{x}|^r$ computation steps, where $\overline{x}$ denotes the input of the machine M. Our goal is to prove that f_M can be defined as a safe recursive function.

In what follows, we represent the tape of the machine M by a couple a variables $(\overline{y_1},\overline{y_2})$ in $(\mathbb{K}^*)^2$ such that the non-empty part is given by $\overline{y_1}^R.\overline{y_2}$, where $\overline{y_1}^R$ is the reversed word of $\overline{y_1}$, and that the head of the machine is on the first letter of $\overline{y_2}$.

We also assume that the m nodes in M are numbered with natural numbers, node 0 being the initial node and node 1 the terminal node. We assume that the terminal node is a loopback node, i.e., its only next node is itself. In the following definitions, node number q will be coded by the word α^q of length q.

Let q ($q \in \mathbb{N}$) be a move node. Three functions are associated with this node:

$$\begin{aligned} \mathcal{G}_\imath(;\overline{y_1},\overline{y_2}) &= \alpha^{q'} \\ \mathcal{H}_\imath(;\overline{y_1},\overline{y_2}) &= \mathrm{tl}(;\overline{y_1}) \qquad \text{or } \mathrm{hd}(;\overline{y_2}).\overline{y_1} \\ \mathcal{I}_\imath(;\overline{y_1},\overline{y_2}) &= \mathrm{hd}(;\overline{y_1}).\overline{y_2} \;\; \text{or } \mathrm{tl}(;\overline{y_2}) \end{aligned}$$

according if one moves right or left. Function $\mathcal{G}_\imath$ returns the encoding of the following node in the computation tree of M, function $\mathcal{H}_\imath$ returns the encoding of the left part of the tape, and function $\mathcal{I}_\imath$ returns the encoding of the right part of the tape.

Let q ($q \in \mathbb{N}$) be a node associated to some operation op of arity n of the structure. We also write Op for the corresponding basic operation. Functions $\mathcal{G}_\imath$, $\mathcal{H}_\imath$, $\mathcal{I}_\imath$ associated with this node are now defined as follows:

$$\begin{aligned} \mathcal{G}_\imath(;\overline{y_1},\overline{y_2}) &= \alpha^{q'} \\ \mathcal{H}_\imath(;\overline{y_1},\overline{y_2}) &= \overline{y_1} \\ \mathcal{I}_\imath(;\overline{y_1},\overline{y_2}) &= \mathrm{cons}(;\mathrm{Op}(;\mathrm{hd}(;\overline{y_2}),\ldots,\mathrm{hd}(;\mathrm{tl}^{(n-1)}(;\overline{y_2}))),\mathrm{tl}^{(n)}(;\overline{y_2})) \end{aligned}$$

Let q ($q \in \mathbb{N}$) be a node corresponding to a relation *rel* of arity n of the structure. The three functions associated with this node are now:

$$\begin{aligned}
\mathcal{G}_\imath(;\overline{y_1},\overline{y_2}) &= \mathrm{C}(;rel(;\mathrm{hd}(;\overline{y_1}),\ldots,\mathrm{hd}(;\mathrm{tl}^{(n-1)}(;\overline{y_2}))),\alpha^{q'},\alpha^{q''})\\
\mathcal{H}_\imath(;\overline{y_1},\overline{y_2}) &= \overline{y_1}\\
\mathcal{I}_\imath(;\overline{y_1},\overline{y_2}) &= \overline{y_2}
\end{aligned}$$

One can define easily without safe recursion, for any integer k, a function $Equal_k$ such that:

$$Equal_k(;\overline{y_1}) = \begin{cases} \alpha \text{ if } \overline{y_1} = \alpha^k \\ \epsilon \text{ otherwise} \end{cases}$$

We can now define the safe recursive functions $next_{state}$, $next_{left}$ and $next_{right}$ which, given the encoding of a state and of the tape of the machine, return the encoding of the next state in the computation tree, the encoding of the left part of the tape and the encoding of the right part of the tape:

$$\begin{aligned}
next_{state}(;\overline{s},\overline{y_1},\overline{y_2}) &= \mathrm{C}\,(;Equal_0(;\overline{s}),\mathcal{G}_0(;\overline{y_1},\overline{y_2}),\mathrm{C}\,(;Equal_1(;\overline{s}),\mathcal{G}_1(;\overline{y_1},\overline{y_2}),\\
&\quad\ldots\mathrm{C}\,(;Equal_m(;\overline{s}),\mathcal{G}_m(;\overline{y_1},\overline{y_2}),\epsilon)\ldots))\\
next_{left}(;\overline{s},\overline{y_1},\overline{y_2}) &= \mathrm{C}\,(;Equal_0(;\overline{s}),\mathcal{H}_0(;\overline{y_1},\overline{y_2}),\mathrm{C}\,(;Equal_1(\overline{s}),\mathcal{H}_1(;\overline{y_1},\overline{y_2}),\\
&\quad\ldots\mathrm{C}\,(;Equal_m(;\overline{s}),\mathcal{H}_m(;\overline{y_1},\overline{y_2}),\epsilon)\ldots))\\
next_{right}(;\overline{s},\overline{y_1},\overline{y_2}) &= \mathrm{C}\,(;Equal_0(;\overline{s}),\mathcal{I}_0(;\overline{y_1},\overline{y_2}),\mathrm{C}\,(;Equal_1(;\overline{s}),\mathcal{I}_1(;\overline{y_1},\overline{y_2}),\\
&\quad\ldots\mathrm{C}\,(;Equal_m(;\overline{s}),\mathcal{I}_m(;\overline{s},\overline{y_1},\overline{y_2}),\epsilon)\ldots))
\end{aligned}$$

From now on, we define with safe recursion the encoding of the state of the machine reached after k computation nodes, where k is encoded by the word $\alpha^k \in \mathbb{K}^*$, and we also define the encoding of the left part and the right part of the tape. All this is done with functions $comp_{state}$, $comp_{left}$ and $comp_{right}$ as follows:

$$\begin{aligned}
comp_{state}(\epsilon;\overline{y_1},\overline{y_2}) &= \epsilon\\
comp_{state}(\alpha^{k+1};\overline{y_1},\overline{y_2}) &= next_{state}\left(;comp_{state}(\alpha^k;\overline{y_1},\overline{y_2}),comp_{left}(\alpha^k;\overline{y_1},\overline{y_2}),\right.\\
&\qquad\left.comp_{right}(\alpha^k;\overline{y_1},\overline{y_2})\right)\\
comp_{left}(\epsilon;\overline{y_1},\overline{y_2}) &= \overline{y_1}\\
comp_{left}(\alpha^{k+1};\overline{y_1},\overline{y_2}) &= next_{left}\left(;comp_{state}(\alpha^k;\overline{y_1},\overline{y_2}),comp_{left}(\alpha^k;\overline{y_1},\overline{y_2}),\right.\\
&\qquad\left.comp_{right}(\alpha^k;\overline{y_1},\overline{y_2})\right)\\
comp_{right}(\epsilon;\overline{y_1},\overline{y_2}) &= \overline{y_2}\\
comp_{right}(\alpha^{k+1};\overline{y_1},\overline{y_2}) &= next_{right}\left(;comp_{state}(\alpha^k;\overline{y_1},\overline{y_2}),comp_{left}(\alpha^k;\overline{y_1},\overline{y_2}),\right.\\
&\qquad\left.comp_{right}(\alpha^k;\overline{y_1},\overline{y_2})\right)
\end{aligned}$$

In order to simplify the notations, we write the above as follows:

$$\begin{aligned}
comp(\epsilon;\overline{y_1},\overline{y_2}) &= \epsilon\\
comp(\alpha^{k+1};\overline{y_1},\overline{y_2}) &= next\left(;comp(\alpha^k;\overline{y_1},\overline{y_2})\right)
\end{aligned}$$

On input $\overline{x}$, with the head originally on the first letter of $\overline{x}$, the final state of the computation of M is then reached after t computation steps, where

$$t = c|\overline{x}|^r$$

The reachability of this final state is given by the following lemma:

Lemma 2. [BMdN02] *For any $c, d \in \mathbb{N}$, one can write a safe recursive function $P_{c,d}$ such that, on any input $\overline{x}$ with $|\overline{x}| = n$, $|P_{c,d}(\overline{x};)| = cn^d$.*

Let us apply this lemma and define such a $P_{c,r}$. Then, we define $finalcomp(\overline{x};) = comp(P_{c,r}(\overline{x};); \epsilon, \overline{x})$. The encoding of the tape at the end of the computation is then given by $finalcomp_{left}(\overline{x};)$ and $finalcomp_{right}(\overline{x};)$ ending here our simulation of the BSS machine M.

5 A Characterization of the Parallel Class PAR$_{\mathcal{K}}$

5.1 A Parallel Model of Computation

In this section, we assume that our structure $\mathcal{K}$ has at least two different constant, denoted by α and β. This is necessary to respect the P-uniformity provisio as below: One needs to describe an exponential gate number with a polynomially long codification.

Recall the notion of circuit over an arbitrary structure $\mathcal{K}$ [Poi95, BCSS98].

Definition 2. *A circuit over the structure $\mathcal{K}$ is an acyclic directed graph whose nodes are labeled either as input nodes of in-degree 0, output nodes of out-degree 0, test nodes 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 valuation of input nodes is defined in the straightforward way, all nodes behaving as one would expect: any test node tests whether its first parent is labeled with α, and returns the label of its second parent if this is true or the label of its third parent if not. See [Poi95, BCSS98]) for formal details.

We say that a family C_n, $n \in \mathbb{N}$ of circuits is P-uniform if and only if there exists a polynomial time deterministic function describing each gate of each circuit.

The reader can find in [BCSS98] the definition of parallel machine over a structure $\mathcal{K}$. We will not give formal definitions here, since we will actually use the alternative characterization given by Proposition 2 below, proved in [BCSS98].

Proposition 2. *The PAR$_{\mathcal{K}}$ class of problems decidable in polynomial time by a parallel machine using an exponentially bounded number of processors is exactly the class of problem decidable by a P-uniform family of circuits of polynomial depth.*

The rest of this section is devoted to prove that, over any structure $\mathcal{K}$, class PAR$_{\mathcal{K}}$ also corresponds to the class of decision functions definable with safe recursion with substitution.

5.2 Safe Recursion with Substitutions

Definition 3. *The set of functions defined with safe recursion with substitutions over $\mathbb{K}$ is the smallest set of functions: $(\mathbb{K}^*)^n \to \mathbb{K}^*$, containing the basic safe functions, and closed under the operations of safe composition and safe recursion with substitutions.*

Basic safe functions are defined in Section 3, as well as the operation of safe composition. We only need to define the notion of safe recursion with substitution : Let $h_1, \dots, h_k : \mathbb{K} \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ and $g_1, \dots, g_k : (\mathbb{K})^2 \times (\mathbb{K}^*)^{kl+1} \to \mathbb{K}^*$. Let the following safe recursive functions $\sigma_{\imath,\jmath} : \mathbb{K}^* \to \mathbb{K}^*, 0 < \imath \leq k, 0 < \jmath \leq l$ for an arbitrary l, called substitution functions. These functions need to be instanciated in the scheme. Here we assume that the arguments of these substitution functions are all safe. Functions $f_1, \dots, f_k : (\mathbb{K})^2 \times (\mathbb{K}^*)^2 \to \mathbb{K}^*$ can then be defined by safe recursion with substitutions:

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

$$\begin{aligned} & f_1(a.\overline{x}, \overline{z}; \overline{u}, \overline{y}) \\ & = \begin{cases} g_1\left(\overline{x}, \overline{z}; f_1(\overline{x}, \overline{z}; \sigma_{1,1}(;\overline{u}), \overline{y}), \dots, f_1(\overline{x}, \overline{z}; \sigma_{1,l}(;\overline{u}), \overline{y}), \dots \right. \\ \qquad\qquad \left. f_k(\overline{x}, \overline{z}; \sigma_{k,1}(;\overline{u}), \overline{y}), \dots, f_k(\overline{x}, \overline{z}; \sigma_{k,l}(;\overline{u}), \overline{y}), \overline{y}\right) \\ \qquad\qquad\qquad \text{if } \forall \imath, \jmath \ f_\imath(\overline{x}, \overline{z}; \sigma_{\imath,\jmath}(;\overline{u}), \overline{y}) \neq \perp \\ \perp \qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \\ & \vdots \\ & f_k(a.\overline{x}, \overline{z}; \overline{u}, \overline{y}) \\ & = \begin{cases} g_k\left(\overline{x}, \overline{z}; f_1(\overline{x}, \overline{z}; \sigma_{1,1}(;\overline{u}), \overline{y}), \dots, f_1(\overline{x}, \overline{z}; \sigma_{1,l}(;\overline{u}), \overline{y}), \dots \right. \\ \qquad\qquad \left. f_k(\overline{x}, \overline{z}; \sigma_{k,1}(;\overline{u}), \overline{y}), \dots, f_k(\overline{x}, \overline{z}; \sigma_{k,l}(;\overline{u}), \overline{y}), \overline{y}\right) \\ \qquad\qquad\qquad \text{if } \forall \imath, \jmath \ f_\imath(\overline{x}, \overline{z}; \sigma_{\imath,\jmath}(;\overline{u}), \overline{y}) \neq \perp \\ \perp \qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \end{aligned}$$

5.3 Simulation of a P-uniform Family of Circuits

By hypothesis, the family of circuits we want to simulate here is P-uniform. This means that there exists a polynomial time deterministic function which, given n the length of the input of the circuit and m the gate number, gives the description of the mth gate of circuit C_n.

We detail now how a gate is described:

- The single (remember, we simulate a decision function) output node is numbered 0. It is represented by $\epsilon \in \mathbb{K}^*$.
- Let r be the maximal arity of a relation or a function of the structure. Let $s = \lceil \lg(\max\{r, 3\}) \rceil$ be the size necessary to write the binary encoding of $0, 1, \dots, \max\{r, 3\}$, the maximum number of parents for a given node. Assume that a gate is represented by $\overline{y}$. Its parents nodes, from its first to

its mth, are represented by $\overline{a_0}.\overline{y}, \ldots, \overline{a_{m-1}}.\overline{y}$, where $\overline{a_\imath} \in \{\beta, \alpha\}^*$ represents the binary encoding of $\imath$ in $\mathbb{K}^*$, β being put in place of 0 and α in place of 1. We add as many βs as needed in front such that these $a_\imath$ have length s.

We define the safe recursive functions $\sigma_\imath$ such that $\sigma_\imath(\overline{y}) = a_\imath.\overline{y}$ where $a_\imath$ is defined above.

Note 4. We assume here a "tree" viewpoint for the description of the circuit, by opposition to the (more classical) "Directed Acyclic Graph" (DAG) viewpoint. In this tree viewpoint, the representations of a gate are codifications of the paths from the output node to it. In particular, a gate may have several representations. P-uniformity ensures that the translation can be done in polynomial time by a deterministic function.

Theorem 2 proved before in this paper ensures that this description can be computed with safe recursive functions. Therefore, we assume that we have the following safe recursive function *Gate*, which returns a code for the label of the node $\overline{y}$ in the circuit C_n, where $n = |\overline{x}|$ is the size of the input, and $\overline{x} = x_1 . \ldots . x_n$:

$$Gate(\overline{x}; \overline{y}) = \begin{cases} \beta.x_\imath & \text{for an input gate corresponding} \\ & \text{to the } \imath^{th} \text{ coordinate of the input} \\ \alpha^\imath & \text{for a gate labeled with op}_\imath \\ \alpha^{k+\imath} & \text{for a gate labeled with rel}_\imath \\ \alpha^{k+l+1} & \text{for a } \textit{test} \text{ node} \end{cases}$$

Remember the functions $Equal_k$ defined in Section 4, and denote with $\overline{t}$, the current depth in the simulation of the circuit. The simulation of the circuit is done with the function *Eval* defined as follows, where $k_\imath$ is the arity of $op_\imath$ and $l_\imath$ the arity of $rel_\imath$:

$$\begin{aligned}
Eval(\epsilon, \overline{x}; \overline{y}) &= \mathrm{C}(; Gate(\overline{x}; \overline{y}), \mathrm{tl}(; Gate(\overline{x}; \overline{y})), \epsilon) \\
Eval(t_1.\overline{t}, \overline{x}; \overline{y}) &= \mathrm{C}\big(; Equal_1(; Gate(\overline{x}; \overline{y})), \mathrm{Op}_1\ (; Eval(\overline{t}, \overline{x}; \sigma_0(; \overline{y})), \\
&\qquad \ldots, Eval(\overline{t}, \overline{x}; \sigma_{k_1}(; \overline{y}))\big), \\
&\quad \vdots \\
&\quad \ldots \mathrm{C}\big(; Equal_{k+1}(; Gate(\overline{x}; \overline{y})), \mathrm{Rel}_1\ (; Eval(\overline{t}, \overline{x}; \sigma_0(; \overline{y})), \\
&\qquad \ldots, Eval(\overline{t}, \overline{x}; \sigma_{l_1}(; \overline{y}))\big), \\
&\quad \vdots \\
&\quad \ldots \mathrm{C}\big(; Equal_{k+l}(; Gate(\overline{x}; \overline{y})), \mathrm{Rel}_l\ (; Eval(\overline{t}, \overline{x}; \sigma_0(; \overline{y})), \\
&\qquad \ldots, Eval(\overline{t}, \overline{x}; \sigma_{l_l}(; \overline{y}))\big), \\
&\quad \mathrm{C}\,(; Equal_{k+l+1}(; Gate(\overline{x}; \overline{y})), \\
&\quad \mathrm{C}\big(Eval(\overline{t}, \overline{x}; \sigma_0(\overline{y};)), Eval(\overline{t}, \overline{x}; \sigma_1(\overline{y};)), Eval(\overline{t}, \overline{x}; \sigma_2(\overline{y};))\big), \\
&\qquad \mathrm{C}(; Gate(\overline{x}; \overline{y}), \mathrm{tl}(; Gate(\overline{x}; \overline{y})), \epsilon)) \ldots) \ldots))
\end{aligned}$$

Assume $p(n) = cn^d$ is a polynomial bounding the depth of the circuit C_n. The evaluation of C_n on input $\overline{x}$ of length n is then given by $Eval(\overline{t}, \overline{x}; \epsilon)$ where $|\overline{t}| = cn^d$. Lemma 2 gives the existence of a safe recursive function $P_{c,d}$ such that $|P_{c,d}(\overline{x};)| = cn^d$. The evaluation of the P-uniform family of circuits $\mathrm{C}_n, n \in \mathbb{N}$ is then given by the function $Circuit(\overline{x};) = Eval(P_{c,d}(\overline{x};), \overline{x}; \epsilon)$ defined with safe recursion with substitutions.

5.4 Evaluation of a Function Defined with Safe Recursion with Substitutions

Let f be a function defined with safe recursion with substitutions, and denote by f_n the restriction of f on the set of inputs of size at most n. We need to prove that f can be simulated by a P-uniform family of circuits $\mathcal{C}(f)_n, n \in \mathbb{N}$ of polynomial depth, each $\mathcal{C}(f)_n$ simulating f_n.

Let us prove by induction on the definition tree of f the following lemma:

Lemma 3. *For any function $f : (\mathbb{K}^*)^r \times (\mathbb{K}^*)^s$ defined with safe recursion with substitutions, let us denote by $D\ulcorner f(\ldots)\urcorner$ the depth of the circuit $\mathcal{C}_n$ simulating $f(\ldots)$ on inputs of size at most n. Then,*

$$\begin{aligned} &D\ulcorner f(\overline{x_1}, \ldots, \overline{x_r}; \overline{y_1}, \ldots, \overline{y_s})\urcorner \\ &\leq p_f(\max\{D\ulcorner\overline{x_1}\urcorner, \ldots, D\ulcorner\overline{x_r}\urcorner\}) + \max\{D\ulcorner\overline{y_1}\urcorner, \ldots, D\ulcorner\overline{y_s}\urcorner\} \end{aligned}$$

for some polynomial p_f, and

$$|f(\overline{x_1}, \ldots, \overline{x_r}; \overline{y_1}, \ldots, \overline{y_s})| \leq q_f(|\overline{x_1}| + \ldots + |\overline{x_s}|)$$

for some polynomial q_f.

Proof.

- If f is a basic function, the result is straightforward.
- If f is defined with the operation of safe composition:

$$\begin{aligned} &f(\overline{x_1} \ldots, \overline{x_r}, \overline{y_1}, \ldots, \overline{y_{p_1}}, \overline{z_1}, \ldots, \overline{z_{p2}}; \overline{t_1}, \ldots, \overline{t_m}) \\ &= g(h_1(\overline{x_1}, \ldots, \overline{x_r}, \overline{y_1}, \ldots, \overline{y_{p_1}};); h_2(\overline{x_1}, \ldots, \overline{x_n}, \overline{z_1}, \ldots, \overline{z_{p2}}; \overline{t_1}, \ldots, \overline{t_m})) \end{aligned}$$

  then, $\mathcal{C}(f)_n$ is the obtained by plugging $\mathcal{C}(h_1)_n$ and $\mathcal{C}(h_2)_n$ in the input nodes of $\mathcal{C}(g)_{q_{h_1}(n)+q_{h_2}(n)}$. Thus, when we apply the induction hypothesis to g:

$$\begin{aligned} p_f(n) &\leq p_g\left(p_{h_1}(n)\right) + \max\{p_{h_1}(n), p_{h_2}(n)\} \\ q_f(n) &\leq q_g\left(q_{h_1}(n)\right) \end{aligned}$$

- If f is defined with the operation of safe recursion with substitutions:
- The non-trivial case is the case of a function f defined with safe recursion, as in Definition 3.

Let us apply induction hypothesis to function $g_\imath$ in the expression $f_\imath(a.\overline{x},\overline{z};\overline{u},\overline{y})$. $\overline{u}$ and $\overline{y}$ are given by their respective circuits plugged at the right position. The depth of the circuit evaluating

$$f_\imath(a.\overline{x},\overline{z};\overline{u},\overline{y}) = g_\imath\left(\overline{x},\overline{z}; f_1(\overline{x},\overline{z};\sigma_{1,1}(;\overline{u}),\overline{y}),\ldots,f_1(\overline{x},\overline{z};\sigma_{1,l}(;\overline{u}),\overline{y}),\ldots\right.$$
$$\left. f_k(\overline{x},\overline{z};\sigma_{k,1}(;\overline{u}),\overline{y}),\ldots,f_k(\overline{x},\overline{z};\sigma_{k,l}(;\overline{u}),\overline{y}),\overline{y}\right)$$

is given by $p_{g_\imath}(\max\{D\ulcorner\overline{x}\urcorner,D\ulcorner\overline{z}\urcorner\}) + \max_{\imath,\jmath}\{D\ulcorner f_\imath(\overline{x},\overline{z};\sigma_{\imath,\jmath}(;\overline{u}),\overline{y})\urcorner, D\ulcorner\overline{y}\urcorner\}$. Assume: for any $\imath$, $p_{g_\imath}$ is bounded by some polynomial p_g. Assume moreover: for any $\imath,\jmath$, $p_{\sigma_{\imath,\jmath}}$ is bounded by some p_σ. We get:

$$\begin{aligned}
&D\ulcorner f(a.\overline{x},\overline{z};\overline{u},\overline{y})\urcorner\\
&\le D\ulcorner\overline{y}\urcorner + p_g(\max\{D\ulcorner\overline{x}\urcorner,D\ulcorner\overline{z}\urcorner\}) + \max_{\imath,\jmath}\{D\ulcorner f_\imath(\overline{x},\overline{z};\sigma_{\imath,\jmath}(;\overline{u}),\overline{y})\urcorner\}\\
&\le D\ulcorner\overline{y}\urcorner + p_g(\max\{D\ulcorner\overline{x}\urcorner,D\ulcorner\overline{z}\urcorner\}) + p_g(\max\{D\ulcorner\overline{\mathrm{tl}(x)}\urcorner,D\ulcorner\overline{z}\urcorner\}) + \ldots\\
&\quad \ldots + p_g(\max D\ulcorner\overline{\epsilon}\urcorner,D\ulcorner\overline{z}\urcorner\}) + p_h(D\ulcorner\overline{z}\urcorner) + \max\{|a.\overline{x}|p_\sigma() + D\ulcorner\overline{u}\urcorner, D\ulcorner\overline{y}\urcorner\}
\end{aligned}$$

Assuming without loss of generality p_g monotone, we get

$$\begin{aligned}
D\ulcorner f(a.\overline{x},\overline{z};\overline{u},\overline{y})\urcorner \le{}& |a.\overline{z}|p_g(\max\{D\ulcorner\overline{x}\urcorner,D\ulcorner\overline{z}\urcorner\})\\
& D\ulcorner\overline{y}\urcorner + p_h(D\ulcorner\overline{z}\urcorner) + \max\{|a.\overline{x}|p_\sigma() + D\ulcorner\overline{u}\urcorner, D\ulcorner\overline{y}\urcorner\},
\end{aligned}$$

from which the result follows for p_f. For q_f, we need:
$|f_\imath(a.\overline{x},\overline{z},\overline{u};\overline{y})| \le q_{g_\imath}\left(|\overline{x}| + |\overline{x}|\right)$. This ends the proof of our lemma.

It follows from the lemma that every circuit of the family $\mathcal{C}(f)_n$ has a polynomial depth in n. The P-uniformity is given by the description of the circuit as above.

In the classical setting (see [LM95]), safe recursion with substitution characterizes the class PSPACE. However, in the general setting, this notion of working space is meaningless, as pointed in [Mic89]: on some structures like $(\mathbb{R},0,1,=,+,-,*)$, any computation can be done in constant working space. However, since we have in the classical setting PAR = PSPACE, our result extends the classical one from [LM95].

References

[BC92] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer Verlag, 1998.

[BMdN02] Olivier Bournez, Jean-Yves Marion, and Paulin de Naurois. Safe recursion over an arbitrary structure: Deterministic polynomial time. Technical report, LORIA, 2002.

[BSS89] Lenore Blum, Mike Shub, and Steve 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.

[Clo95] P. Clote. Computational models and function algebras. In D. Leivant, editor, *LCC'94*, volume 960, pages 98–130, 1995.

[Cob62] 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.

[Coo92] S. A. Cook. Computability and complexity of higher-type functions. In Y. Moschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer-Verlag, New York, 1992.

[CSS94] F. Cucker, M. Shub, and S. Smale. Separation of complexity classes in Koiran's weak model. *Theoretical Computer Science*, 133(1):3–14, 11 October 1994.

[Cuc92] F. Cucker. $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NC}_{\mathbb{R}}$. *Journal of Complexity*, 8:230–238, 1992.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[Fag74] R. Fagin. Generalized first order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computation*, pages 43–73. SIAM-AMS, 1974.

[Goo94] J. B. Goode. Accessible telephone directories. *The Journal of Symbolic Logic*, 59(1):92–105, March 1994.

[Gur83] Y. Gurevich. Algebras of feasible functions. In *Twenty Fourth Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, 1983.

[Hof99] M. Hofmann. Type systems for polynomial-time computation, 1999. Habilitation.

[Imm99] N. Immerman. *Descriptive Complexity*. Springer, 1999.

[Jon00] N. Jones. The expressive power of higher order types or, life without cons. 2000.

[Lei95] D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, number 960, pages 177–194, 1995.

[LM93] D. Leivant and J-Y Marion. Lambda calculus characterizations of poly-time. *fi*, 19(1,2):167,184, September 1993.

[LM95] Daniel Leivant and Jean-Yves 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.

[Mee92] K. Meer. A note on a $P \neq NP$ result for a restricted class of real machines. *Journal of Complexity*, 8:451–453, 1992.

[Mic89] Christian Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. In *C. R. Acad. Sc. de Paris*, volume 309 of *1*, pages 435–437. 1989.

[MM00] J-Y Marion and J-Y Moyen. Efficient first order functional program interpreter with time bound certifications. In *LPAR*, volume 1955, pages 25–42. Springer, Nov 2000.

[Poi95] Bruno Poizat. *Les petits cailloux*. aléas, 1995.

[RIR01] B. Kapron R. Irwin and J. Royer. On characterizations of the basic feasible functionals. 11:117–153, 20001.

[Saz80] V. Sazonov. Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung und Kybernetik*, 7:319–323, 1980.