Département de formation doctorale en informatique

Institut National
Polytechnique de Lorraine

École doctorale IAEM Lorraine

Completeness Results and Syntactic Characterizations of Complexity Classes over Arbitrary Structures

THESE

soutenue publiquement le December 15, 2004

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Paulin Jacobé de Naurois

membres du jury

Président :	Pr. Zhou Ding-Xuan
Rapporteurs :	Pr. Pascal Koiran Ass. Pr. Mordecai Golin
Examinateurs :	Pr. Felipe Cucker Pr. Jean-Yves Marion Ass. Pr. Olivier Bournez
Invité :	Pr. Michel de Rougemeont (rapporteur-invité)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

A Pauline

Acknowledgements

I would like to thank Felipe Cucker for the opportunity he gave me several times to come and study with him in Hong Kong, first for my Master thesis, then for my MPhil, and eventually for this joint PhD Thesis between City University of Hong Kong and the Institut National Polytechnique de Lorraine of Nancy, France. I have particularly appreciated his kindness and the time he devoted to discuss with me, as well as the numerous advises he gave me.

I am also very grateful to Olivier Bournez and to Jean-Yves Marion, who welcomed me in Nancy for doing this PhD. They too, devoted a large amount of time to discuss with me, and helped me with their advises. I enjoyed the time we spent together, and the discussions we had.

I am grateful to Olivier Bournez, Felipe Cucker and Jean-Yves Marion for the amount of time they spent in order to make City University of Hong Kong and the INPL of Nancy reach their first joint-PhD agreement. It required a large amount of time and effort, and I would like to thank them for the energy they devoted to it.

Many thanks to Pascal Koiran, who first made it possible for me to meet Felipe Cucker as well as Olivier Bournez and Jean-Yves Marion, and who next accepted to referee my thesis. I particularly appreciated his comments on this work.

I would also like to thank Michel de Rougemont and Mordecai Golin for refereeing this work, and for their comments.

Zhou Ding-Xuan, for being in the jury, deserves to be thanked.

Peter Bürgisser also deserves my gratitude for the fruitfull discussion that we had last spring in Hong Kong, and that we continued when he invited me last autumn in Paderborn.

For her patience throughout these year, her support and all that she gave me, I want to express my gratitude to Pauline. And to all my friends and family: thank you.

Résumé

Nous nous plaçons dans le modèle de calcul BSS sur des structures arbitraires. Nous présentons de nouveaux résultats de complétude concernant des problèmes géométriques sur les nombres réels avec addition et ordre. Nous présentons aussi plusieurs caractérisations de classes de complexité indépendantes du modèle de calcul sous-jacent. Nous étendons des résultats de Grädel, Gurevich et Meer en complexité descriptive, qui caractérisent les problèmes de décisions calculables en temps polynomial déterministe et non-déterministe en termes de logique sur des structures métafinies. Nous étendons des résultats de Bellantoni et Cook pour caractériser les fonctions calculables en temps polynomial séquentiel, et de Leivant et Marion pour caractériser les fonctions calculables en temps polynomial parallèle, en termes d'algèbres de fonctions récursives. Nous présentons également des caractérisations de fonctions calculables dans la hiérarchie polynomiale et en temps polynomial alternant.

Mots-clés: modèle de calcul BSS, complexité algébrique, complexité descriptive, complexité implicite, logique, algèbres de fonctions récursives.

Abstract

We focus on the BSS model of computation over arbitrary structures. We provide new completeness results for geometrical problems when this structure is the set of real numbers with addition and order. We also provide several machine independent characterizations of complexity classes over arbitrary structures. We extend some results by Grädel, Gurevich and Meer in descriptive complexity, characterizing deterministic and non deterministic polynomial time decision problems in terms of logics over metafinite structures. We extend some results by Bellantoni and Cook, characterizing functions computable in sequential determinisitc polynomial time, and by Leivant and Marion, characterizing functions computable in parallel determinisitc polynomial time in terms of algebras of recursive functions. We also provide some characterizations of functions computable within the polynomial hierarchy and in polynomial alternating time.

Keywords: BSS model of computation, algebraic complexity, descriptive complexity, imlpicitecomplexity, logic, algebra of recursive functions.

Contents

List of Figures

Introduction

In the last decades complexity theory developed in many directions to offer a broad perspective of the complexity of computations. Two directions relevant to our work are the extension of complexity theory to domains other than finite alphabets and the characterizations of complexity classes in machine-independent terms.

A seminal paper for the first direction above is the one by Blum, Shub and Smale [BSS89], where a theory of computation and complexity that allowed an ordered ring or field as alphabet for the space of admissible inputs was developed. The authors emphasized the case when the ring is the field of real numbers, $\mathbb{R}$, bringing the theory of computation into the domain of analysis, geometry and topology. Later on extended to the more abstract level of computations over arbitrary structures in [Goo94, Poi95], this BSS model, among other things, makes use of the extensively developed subject of the theory of discrete computations, initiated by the works of Turing [Tur36] and Church [Chu41]. Indeed, when the computational structure considered in the model is a discrete set, the BSS theory of computation coincides with the classical theory of computation over finite alphabets. When the structure is specialized to the real numbers, the theory gives computable functions which are reasonable for the study of algorithms of numerical analysis, geometry and topology. It provides a formal setting for considering problems of computability -is the Mandelbrot set decidable?- and complexity -does there exist a polynomial time algorithm deciding whether a real polynomial of degree 4 has a zero? When considering an algebraic cost for the basic operations -adding or multiplying two numbers is considered as one single computation step, independently of their values- a resource -the number of computation steps performed by an algorithm- and bounds, a hierarchy of complexity classes similar to the classical one based on [Coo71, Kar72, Lev73] is defined, as well as notions of reduction and completeness, and natural complete problems for many classes are exhibited [CKM97, Koi99, Koi00]. For a reference, see [BCSS98] or [Poi95].

Along these lines, a large amount of work has been done to find structural results relating different complexity classes or different computational structures. A substructure of the field of the real numbers of [BSS89], the ring of real numbers with addition and order, received careful attention [Koi94, CK95]. Transfer theorems were proved, relating the complexity of non-determinism in this setting to the complexity of non-determinism in the classical model of computation over finite alphabets [FK98, FK00], establishing the importance of this structure, lying somehow "halfway" between discrete structures and the field of real numbers. An extensive study of the complexity of computing some geometrical invariants over this structure can been found in [BC04], where the authors use the notion of counting classes, introduced in [Val79a] over finite structures and in [Mee00] over the real numbers to prove several other transfer results. We extend some of these results in Chapter 2 and prove several new completeness results for complexity classes over this structure of real numbers with addition and order.

In the classical setting of computation over finite alphabets, a great amount of work has been done, along the second direction mentioned above, to provide machine independent characterization of the major complexity classes. Such characterizations lend further credence to the importance of the complexity classes considered, relate them to issues relevant to programming and verification, and, in general, help understand the notion of computability in a whole perspective. Several approaches for designing such characterizations have been chosen, among which one can find descriptive complexity (global methods of finite model theory) and applicative programs over algebras.

Descriptive complexity (finite model theory) began with the work of Fagin [Fag74], who proved, in the classical setting, that the class NP can be characterized as the class of sets describable within existential second-order logic. Subsequently, Vardi and Immerman [Var82, Imm83, Imm86] used this approach to characterize P. Several other characterizations exist, for classes like LOGSACE [Gur83] or PSPACE [Mos83, GS86, Imm87, Bon88, AV89, Lei90a, ASV90, AV91, Imm91]. An overview of the subject can be found in [EF95, Imm99]. These characterizations, however, applied originally only to the classical setting of computation over finite alphabets since only finite models were considered. Grädel and Gurevich [GG98] introduced later on the notion of metafinite models, allowing to consider computations over more general structures. A particular case, the $\mathbb{R}$-structures, were developed by Grädel and Meer [GM95]. Additional results along this line, characterizing the classes $\mathrm{PAR}_{\mathbb{R}}$, $\mathrm{EXP}_{\mathbb{R}}$ and $\mathrm{NC}_{\mathbb{R}}$ can be found in [CM99].

Functional schemes are another way of defining classes of functions, such as the primitive recursive functions [Kle36] and the double-recursive functions [Pet66]. Characterizations of classical complexity classes by recurrence schemes originate with Cobham's [Cob62] work, where a characterization of FP by "bounded recursion on notations" is provided. One can also mention the subrecursive characterization of linear space by Ritchie [Rit63]. Unfortunately, Cobham's characterization uses *ad hoc* initial functions and explicit bounds on the function's growth rate, inspired by Grzegorczyk's classification of the primitive recursive functions [Grz53]. Gurevich noticed that, interpreting the classical calculus of primitive recursive functions (resp. recursive functions) over finite objects gives precisely LOGSPACE (resp. FP) [Gur88]. Sazonov [Saz80] gave earlier a variant of the latter result, and Gurevich results were later on extended to other complexity classes such as PSPACE and EXPTIME [Goe89, Lo92], and NC^1 [CL90]. Yet, all these characterizations are not applicable to any computational structure, and can be improved in order to do so.

When the calculus of recursive functions is interpreted over non-finite algebras of words or trees, the use of ramified data, or data tiering, introduced independently by Simmons [Sim88], Leivant [Lei90b, Lei94b] and Bellantoni and Cook [BC92], allows to avoid the use of such explicit bounds. One underlying idea is that the data objects are used computationally in different guises. By explicitly separating the uses, and requiring that the recursion schemes respect that separation, classes of recursive functions are obtained, which correspond closely to major complexity classes. Such characterizations exist, for instance, for FP [BC92, Lei93, LM93], the extended polynomials [Lei90b], the linear space functions [Bel92, Lei93, Ngu93], NC^1 and polylog space [Blo92, LM00], PH [Bel94] the elementary functions [Lei94b], and PSPACE [LM95].

A natural research line is to combine the two directions above by looking for machine-independent characterization of complexity classes over arbitrary structures. For that matter, we use the approach by tiering, which seems more appropriate than the approach by restricting the interpretation of recursive functions over finite objects. A first step lies in the definition of a notion of primitive recursion over arbitrary structures. Such a definition over $\mathbb{R}$ can be found in [BSS89], where an explicit separation is made between real numbers, on which the arithmetical operations are performed, and natural numbers, which are used for the control of the recursion schemes. Based on this work, an extension of the Grzegorczyk hierarchy over the real numbers can be found in [Gak96, Gak97]. Steming on this definition, we propose our own notion of recursion over an arbitrary structure. We use these definition to characterize several complexity classes, namely $\mathrm{FP}_{\mathcal{K}}$, the levels of the polynomial hierarchy $\mathrm{PH}_{\mathcal{K}}$, parallel polynomial time $\mathrm{FPAR}_{\mathcal{K}}$ and polynomial alternating time $\mathrm{PAT}_{\mathcal{K}}$.

Plan of the Thesis

Our thesis is divided into seven chapters, as follows:

Chapter 1: Computability and Complexity over an Arbitrary Structure

In this chapter, we recall the notion of computation over an arbitrary structure $\mathcal{K}$. We emphasize on three main examples: computation over $\mathbb{Z}_2$, which corresponds to the classical notion of computation by a Turing machine, $\mathbb{R}$, which is the original structure considered in [BSS89], and $\mathbb{R}_{ovs}$ the real numbers with addition and order, for which many interesting results have been found. Next, we define the notions of reduction and completeness, present the main complexity classes and recall some interesting results.

Chapter 2: The Complexity of Semilinear Problems

This chapter is devoted to the study of the complexity of many natural geometrical problems over the reals with addition and order. Two topologies are considered, the Euclidian topology and the Zariski topology, and complete problems related to both of them are exhibited in several major complexity classes. All these results are original and unpublished. This chapter is a joint work with Pr. Felipe Cucker and Pr. Peter Bürgisser.

Chapter 3: Classical Characterizations of P

This chapter is purely bibliographical. We recall here some of the major machine independent characterizations of P in the classical setting of computation over finite alphabets found in the literature.

Chapter 4: Metafinite Model Theory

Logical characterizations for deterministic polynomial time P and non-deterministic polynomial time NP in the classical setting exist, see [Fag74, Var82, Imm83, Imm86]. Similar results have been proven over $\mathbb{R}$, see [GM95, CM99], and other arithmetical structures [GG98]. In this chapter, we extend some of these results to the more general context of computations over an arbitrary structure.

Chapter 5: Partial Recursion and Computability

In this chapter we recall the partial and primitive recursive functions over $\mathbb{R}$ of [BSS89], and propose our own notion of partial and primitive recursion over an arbitrary structure. Next, we prove that these partial recursive functions over an arbitrary structure $\mathcal{K}$ are exactly the functions computable by a BSS machine over $\mathcal{K}$.

Chapter 6: Safe Recursion and Poly-Time

Based on our notion of primitive recursion over an arbitrary structure $\mathcal{K}$ of the previous chapter, we extend the notion of classical safe recursion of [BC92] to the notion of safe recursion over an arbitrary structure. Next, we prove that the safe recursive functions over an arbitrary structure $\mathcal{K}$ are exactly the functions computable in polynomial time by a BSS machine over $\mathcal{K}$.

Chapter 7: Parallelism and Non-Determinism

In this chapter we extend the safe recursive functions with substitutions of [LM95] to the notion of safe recursive functions with substitutions over an arbitrary structure $\mathcal{K}$. We show that these functions are exactly the functions computable in polynomial parallel time over $\mathcal{K}$. Next, we extend the approach of [Bel94] to an arbitrary structure $\mathcal{K}$, showing that the levels of the polynomial hierarchy over $\mathcal{K}$ correspond exactly to some classes of safe recursive functions with predicative minimization over $\mathcal{K}$. We propose also a notion of safe recursive functions with predicative substitutions over $\mathcal{K}$, and show that they are exactly the functions computed in polynomial alternating time over $\mathcal{K}$. Similar results are also provided for the digital polynomial hierarchy and digital polynomial alternating time over $\mathcal{K}$.

Chapter 1

Computability and Complexity over an Arbitrary Structure

In this chapter, we describe computability and complexity in the BSS model of computation, introduced over the real numbers by Blum, Shub and Smale in [BSS89] and named after these authors. Their notions have been later on extended on arbitrary structures [Goo94]. Detailed accounts can be found in [BCSS98] for structures like real and complex numbers, or [Poi95] for considerations about arbitrary structures. Next, we introduce the main complexity classes over this setting, and define the notions of reductions and completeness. Some interesting results are recalled

1.1 Computing in the BSS model

This section is devoted to the exposition of the BSS model of computation, based on [BCSS98] and [Poi95].

1.1.1 Arbitrary Structures

Definition 1.1.1 (Structure)

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 $\mathbb{K}$, some operators $\{op_i\}_{i\in I}$, 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 with arity greater than 1 is assumed to be finite, that is, only symbols for constants may be infinitely many. We will not distinguish between operator and relation symbols and their corresponding interpretations as functions and relations of the same arity, 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

0 and **1** in our work) in the structure.

The main examples of structures are:

- $\mathbb{Z}_2 = (\{0,1\}, =, \mathbf{0}, \mathbf{1})$. When considered over this discrete structure, the notions of computability and complexity involved in our work are the classical ones. In particular, machines over this structure are Turing machines, and complexity classes defined over this structure are the classical ones. Since this structure is the classical discrete one, we will usually omit the subscript $\mathbb{Z}_2$ for denoting problems or classes, thus using the classical denominations.

- $\mathbb{R} = (\mathbb{R}, +, -, *, /, <, \{c \in \mathbb{R}\})$. This structure is the one introduced by Steve Smale, Lenore Blum and Mike Shub in their seminal paper [BSS89] for dealing with computational problems over the real numbers. This setting allows for the analysis of the computation over the real numbers and gives a nice background for numerical analysis.

- $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, <, \{c \in \mathbb{R}\})$. This structure gives a background for the analysis of many geometrical problems involving affine spaces. It is also denoted as $\mathbb{R}_{add}$, the subscript "add" being a shorthand for "additive model". Chapter 2 is devoted to the study of several geometrical problems over this structure.

Remark 1.1.1 In our exposition we have made the choice to describe the constants as operators of arity zero. While this is not the usual description of computation, as found in [Poi95], it makes practically no real difference. In the exposition of [Poi95], machines have (a finite number of) constant nodes, with corresponding constants $c_i \in \mathbb{K}$. Different machines may have different constants associated with their constant nodes, and the range of possible values is the whole set $\mathbb{K}$. Some authors then considered classes of problems decided with machines having *no* constant node, as in [FK00], where the equivalence $\mathrm{P}^0_{\mathsf{add}} = \mathrm{NP}^0_{\mathsf{add}} \Leftrightarrow \mathrm{P} = \mathrm{NP}$ is proven. This can be seen as a transfer theorem for a specific model of computation, that is BSS machines with no constant node. In our exposition, machines have also a finite number of constant nodes, but the range of possible values is determined by the structure itself. We denote the group of real numbers with addition and order by $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, <, \{c \in \mathbb{R}\})$, allowing the constants to range over the whole set $\mathbb{R}$. BSS Machines with no constant nodes will be machines over the substructure $\mathbb{R}^0_{ovs} = (\mathbb{R}, +, -, <, 0, 1)$, and Fournier and Koiran's result will be stated as $\mathrm{P}_{\mathbb{R}^0_{ovs}} = \mathrm{NP}_{\mathbb{R}^0_{ovs}} \Leftrightarrow \mathrm{P} = \mathrm{NP}$. The same result can then be seen as a transfer result for the *same* model of computation, over two different structures. Modulo a very slight reformulation of the results, as shown by the example above, the two expositions are strictly equivalent. Our motivation for choosing this one is to make the definition of algebras of recursive functions easier, and also the proof of our results concerning these algebras of functions. It allows us to have the same generic definitions, which can be applied indifferently, for instance, to the two structures $\mathbb{R}_{ovs}$ and $\mathbb{R}^0_{ovs}$ above. It also provides a safer ground for defining appropriate logics, as in Chapter 4 and [GG98] where the same presentation is given.

Remark 1.1.2 Any structure $\mathcal{K}$ as above contains a copy of $(\{0,1\}, =, \mathbf{0}, \mathbf{1})$. Indeed, by hypothesis, any structure contains at least two different constants with different interpretations, and a binary relation interpreted as the equality over $\mathbb{K}$.

We denote by $\mathbb{K}^* = \bigcup_{i \in \mathbb{N}} \mathbb{K}^i$ the set of words over the alphabet $\mathbb{K}$. The space $\mathbb{K}^*$ is the analogue to Σ^* the set of all finite sequences of zeros and ones. 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 ϵ the empty word. The length of a word $\overline{w} \in \mathbb{K}^*$ is denoted by $|\overline{w}|$.

1.1.2 BSS Machines

We now define machines over $\mathcal{K}$ following the lines of [BCSS98].

Definition 1.1.2 (Machine)

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[1] $\mathcal{S} = \mathbb{K}_*$, together with a connected directed graph whose nodes labelled

[1] 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.

$0, \ldots, N$ correspond to the set of different *instructions* of the machine. These nodes are of one of the six following types: input, output, computation, copy, branching and shift nodes. Let us describe them a bit more.

1. *Input nodes.* There is only one input node and it is labelled 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 labelled 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.

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 0 of $\mathcal{S}$ by the value $op(w_0, \ldots, w_{n-1})$ where $w_0, w_2, \ldots, w_{n-1}$ are components 0 to $n-1$ 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.

4. *Copy 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 performs one of the following actions:

   - Replace the component indexed by 0 by a copy of the component indexed by 1. This is denoted as a *copy left*.

   - Replace the component indexed by 0 by a copy of the component indexed by -1. This is denoted as a *copy right*.

   - Exchange the component indexed by 0 and the component indexed by 1 . This is denoted as a *switch*.

5. *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 $rel(w_0, \ldots, w_{n-1})$ is true and $\beta^-(m)$ otherwise, where $w_0, w_2, \ldots, w_{n-1}$ are components 0 to $n-1$ of $\mathcal{S}$ and rel is some relation of the structure $\mathcal{K}$ of arity n.

6. *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 σ 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 [BCSS98, BSS89, Poi95]. We will not dwell on these details but focus on the essential ideas in the proofs to come in the sequel.

Remark 1.1.3 The original notion of a BSS machine found in [BSS89] applies only to $\mathbb{R}$, and has been extended in [BCSS98] to the notion of computation over a ring. Accordingly to [BCSS98], computation nodes have an associated polynomial or *rational* map g_m, computing several polynomial functions. On the other hand, the exposition in [Poi95] applies to any structure and relies on an extension of the classical Turing machine [Tur36] with several tapes. Thus, in [Poi95], computation nodes perform only operations of the structure, as above, but allow one to copy an element from one cell in one tape onto one cell in one other tape. In our exposition we consider the same basic operations over one single tape. Therefore we need to introduce some ways to copy and displace elements over the tape: this is the purpose of our *copy* nodes. It is clear that the polynomial and rational maps of [BCSS98] can be achieved by composing a finite number of operations of the structure and copies of elements on the tape, and therefore that our definition leads to the same notion of computability and the same complexity classes than [BSS89, BCSS98, Goo94, Poi95].

Remark 1.1.4 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 $\mathbb{K} \cup \#$, the symbol "$\#$" being the "blank" symbol [Poi95, BCSS98]. 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. In what follows we will freely use the common expressions "tape", "scanning head", etc., the translation between these concepts and a shifting register space with a designated 1st position being obvious.

Definition 1.1.3 For a given machine M, the function φ_M associating its output to a given input $\overline{x} \in \mathbb{K}^*$ 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 φ_M. Such subsets of $\mathbb{K}^*$ are also denoted as *languages* or *problems* over the structure $\mathcal{K}$.

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. It gives an instantaneous description of the computation of the machine at a given step.

Remark 1.1.5 By Remark 1.1.2, it is clear that, over any structure $\mathcal{K}$, machines have at least the computational power of classical Turing machines. It follows that there exists universal machines over any structure $\mathcal{K}$: a universal BSS machine $\mathcal{U}$ over a structure $\mathcal{K}$ takes as input the encoding of any machine M over $\mathcal{K}$ and any $\overline{x} \in \mathbb{K}^*$, and outputs $\varphi_M(\overline{x})$ when it is defined. To give an idea, M is encoded as an element in $\{\mathbf{0}, \mathbf{1}\}^*$ describing the finite directed graph and the type of the nodes, together with an element $(c_1, \ldots, c_r) \in \mathbb{K}^*$ providing the values of the constant nodes of M. $\mathcal{U}$ simulates a constant node of M by accessing the corresponding value c_i and copying it to the appropriate location. Therefore, $\mathcal{U}$, which has only a finite number of constants, can simulate any machine M, even in the case of a structure $\mathcal{K}$ with an infinite number of constant operations. It is also easy to see that the classical tricks, such as clocking a machine or simulating an algorithm, can be applied in this general setting.

1.2 Computing with Circuits

In this section we introduce the notion of circuit over $\mathcal{K}$, as found in [Poi95, BCSS98, Gat86], and recall some links of this computational device with the BSS model of computation. While some definitions of parallel machines are found in the literature (see [BCSS98]), circuits provide a simple formalism for describing parallel computations. Circuits differ from machines in that they make computations over inputs of constant size. Thus, for dealing with decision problems of variable input size, we shall consider families of circuits.

1.2.1 Circuits

Definition 1.2.1 (Circuit)

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

The evaluation of a circuit on a given assignment of values of $\mathbb{K}$ to its input gates is defined in a straightforward way: an input gate reads one element of the input, a computation gate computes an operator of the structure, of arity k, over its k parent gates, a selection gate tests whether its first parent is labeled with $\mathbf{1}$, and returns the evalutation of its second parent if this is true or the evaluation of its third parent if not, and an output gate provides one element of the output. This evaluation defines a function from $\mathbb{K}^n$ to $\mathbb{K}^m$ where n is the

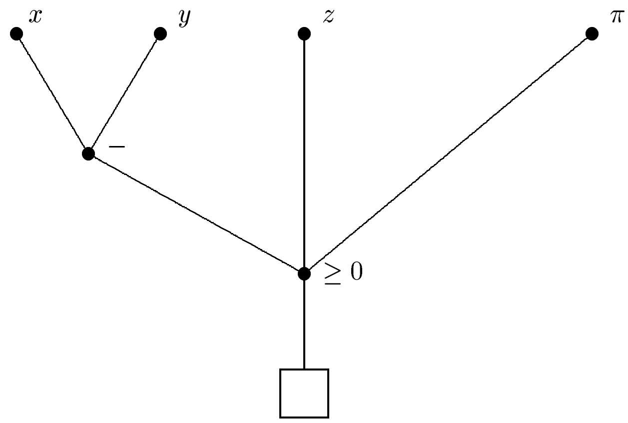

Figure 1.1: A Circuit over $\mathbb{R}_{ovs}$ Computing: if $x \geq y$ then z else π.

number of input gates and m that of output gates. See [Poi95, BCSS98] for formal details.

We say that a family $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ of circuits computes a function $f : \mathbb{K}^* \to \mathbb{K}^*$ when the function computed by the nth circuit of the family is the restriction of f to $\mathbb{K}^n$. We say that this family is *P-uniform* when there exists a deterministic Turing machine M satisfying the following. There exist constants $\alpha_1, \ldots, \alpha_m \in \mathbb{K}$ such that, for every $n \in \mathbb{N}$, the constant gates of $\mathscr{C}_n$ have associated constants in the set $\{\alpha_1, \ldots, \alpha_m\}$ and M computes a description of the ith gate of the nth circuit in time polynomial in n (if the ith gate is a constant gate with associated constant α_k then M returns k instead of α_k). When the machine M computing a description of the gates of the circuits works in time $t(n)$ for a non-decreasing time-constructible function $t : \mathbb{N} \to \mathbb{N}$, we say simply that family $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ of circuits is *$t(n)$-uniform*.

Remark 1.2.1 It is usually assumed that gates are numbered consecutively with the first gates being the input gates and the last ones being the output gates. In addition, if gate i has parents $j_1, \ldots, j_r$ then one must have $j_1, \ldots, j_r < i$. Unless otherwise stated we will assume this enumeration applies.

1.2.2 Circuits and Machines

The first link between boolean circuits (circuits over $\mathbb{Z}_2$) and computability is due to Shannon [Sha49]. More recently, Savage compared circuit complexity and algorithmic complexity [Sav72, Sav76]. Some developments can also be found in [Weg87, BS90]. Following these lines, some computational links between circuits over a structure $\mathcal{K}$ and BSS machines have been exhibited [Goo94, Poi95]. By Remark 1.1.5, it is easy to simulate a circuit, or even a

$t(n)$-uniform family of circuits by a BSS machine. We shall see that the reverse holds under certain circumstances. In [Poi95] the following result is proved, relating the computational power of circuits and machines over an arbitrary structure.

Proposition 1.2.1 *Assume M is a BSS machine over $\mathcal{K}$ computing a function f_M. Denote by $\alpha_1, \ldots, \alpha_m \in \mathbb{K}$ the constants used by M. Assume moreover that, for all inputs of size n, the computation time of M is bounded by some time-constructible function $t(n) \geq n$, and that the length of an output depends only on the size of its input. Then, there exists a $t(n)^2$-uniform family of circuits $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ such that $\mathscr{C}_n$ has $n+m$ inputs $(x_1, \ldots, x_n, y_1, \ldots, y_m)$, has size polynomial in $t(n)$, and, for all $\overline{x} = x_1 \ldots x_n$, the evaluation of $\mathscr{C}_n$ on input $(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m)$ equals the output of f_M on input $\overline{x}$.*

Remark 1.2.2 The requirements that the computation time bound and output size depend only on the size of the input, and not on its value, are not too strong: From an arbitrary machine M with n nodes, computing a function ϕ_M, it suffices to add a clock and some extra idle characters to the output to build in time $O(n)$ a machine M' complying with these requirements and which computes a reasonable encoding of ϕ_M. For a machine deciding a set, clocking this machine is enough to comply with these requirements.

1.3 Complexity Classes

In this section we introduce the main complexity classes that we will have to deal with in this work. Complexity classes will be defined with respect to the BSS model of computation exposed above. We will only assume a *unit cost* measure for every computation step, i.e. every computation step of a BSS machine will be considered to have cost 1, independently of the value of its arguments. Also, the resource we use for measuring the complexity of problems is time, i.e. the number of computation steps required by a machine to halt on an input. We do *not* consider complexity classes defined with space as a resource, since there exist some structures where space is not a relevant resource, as shown by the following results, due to Christian Michaux:

Proposition 1.3.1 *[Mic89] Every algorithm over $\mathbb{R}_{ovs}$ can be carried out by a machine working in constant space, up to an exponential increase of the time cost.*

Moreover,

Proposition 1.3.2 *[Mic89] Every polynomial time algorithm over* $(\mathbb{R}, +, -, /2, <, \{c \in \mathbb{R}\})$ *can be carried out by a machine working in constant space and polynomial time.*

As a consequence, a complexity class defined with respect to space as a resource is likely to be trivial over some structures.

1.3.1 Sequential Polynomial Time

Definition 1.3.1 (Deterministic Polynomial Time)
A set $S \subset \mathbb{K}^*$ is in class $\mathrm{P}_{\mathcal{K}}$ (respectively a function $f : \mathbb{K}^* \rightarrow \mathbb{K}^*$ is in class 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 the function $f(\overline{w})$).

Proposition 1.3.3 *[Poi95]* $\mathrm{P}_{\mathcal{K}}$ *is the class of problems decided by P-uniform families of circuits of polynomial size.*

Polynomial time is the usual measure of tractability: a natural problem in $\mathrm{P}_{\mathcal{K}}$ is usually solved by an algorithm working in time bounded by a polynomial of low-degree, and can therefore be implemented and run in a reasonable amount of time. Non-determinism allows one to define several other polynomial time complexity classes as follows:

Definition 1.3.2 (Non-Deterministic 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.

In other words, $\mathrm{NP}_{\mathcal{K}}$ is the class of problems that are difficult to solve, but easy to check: an input $\overline{x}$ belongs to the problem if and only if there exists a witness, or proof, $\overline{y}$, which allows to check the membership in polynomial time. Over most of the natural structures, in particular over our three main structures $\mathbb{Z}_2$, $\mathbb{R}$ and $\mathbb{R}_{ovs}$, it is unknown whether $\mathrm{P}_{\mathcal{K}}$ equals $\mathrm{NP}_{\mathcal{K}}$. This question over $\mathbb{R}$ and $\mathbb{R}_{ovs}$ may be related to the classical question over $\mathbb{Z}_2$, P = NP. Studying this question over $\mathbb{R}$ or $\mathbb{Z}_2$ is therefore helpful to understand the classical setting [FK00]. Also, the reader should keep in mind that one can exhibit some structures where $\mathrm{P}_{\mathcal{K}} \neq \mathrm{NP}_{\mathcal{K}}$ such as $\mathbb{R}_{vs} = (\mathbb{R}, +, -, =, \{c \in \mathbb{R}\})$ ([Mee92]).

1.3.2 Computing with Oracles

Let $\mathcal{K}$ be a structure, and let $L \subseteq \mathbb{K}^*$ be a problem over $\mathcal{K}$. A machine M with oracle L over $\mathcal{K}$ is a machine as in Definition 1.1.2 with the additional type of node:

7. *Oracle nodes.* Associated with a node m of this type there are a next node $\beta(m)$ and a map $g_m : \mathcal{S} \to \mathcal{S}$. When the content of $\mathcal{S}$ is $(\ldots, w_{-1}, w_0, w_1, \ldots, w_n)$ for any $n \in \mathbb{N}$, $(w_1, \ldots, w_n) \in \mathbb{K}^*$ is taken as an input to the problem L. If $(w_1, \ldots, w_n) \in L$ The function g_m replaces the component indexed by 0 of $\mathcal{S}$ by $\mathbf{1}$, otherwise it replaces it by $\mathbf{0}$. The other components of $\mathcal{S}$ are left unchanged. When $n = 0$, we shall consider that the node tests the membership of the empty word to L.

Recall the analogy of Remark 1.1.4: a machine with oracle is essentially a Turing with one additional node, denoted as *oracle* node. When entering an oracle node, we shall consider that the machine reads the right part of the tape, and replaces its first character with $\mathbf{0}$ or $\mathbf{1}$, depending on whether this word belongs to the oracle language. Several other conventions, using an additional tape, denoted as "oracle tape" exist. Since they are equivalent to the one above, up to a polynomial increase in the computation time, we use the one above, which simplifies our proofs.

Note that this is performed by one single node of M, thus in one computation step independently of the complexity of L. A machine with oracle L can be seen as an algorithm which has access for free to a subroutine deciding L. The oracle node is a call to this subroutine, and when the answer comes, the computation resumes.

Remark 1.3.1 Following Remark 1.1.2, we shall sometimes consider machines over $\mathcal{K}$ with discrete oracles, i.e oracles over $\mathbb{Z}_2$. The correspondence is made by identifying $0 \in \mathbb{Z}_2$ with $\mathbf{0} \in \mathbb{K}$ and $1 \in \mathbb{Z}_2$ with $\mathbf{1} \in \mathbb{K}$. This is useful for relating complexity classes defined over different structures, such as in Theorem 1.4.3 hereafter.

Definition 1.3.3 (Complexity Classes with Oracles)

Given a complexity class $\mathcal{C}_{\mathcal{K}}$ over $\mathcal{K}$, and a problem L over $\mathcal{K}$, $\mathcal{C}_{\mathcal{K}}^L$ is the set of problems decided by the machines of $\mathcal{C}_{\mathcal{K}}$ with oracle L.

Given a complexity class $\mathcal{C}_\mathcal{K}$ over $\mathcal{K}$, and a set $\mathcal{L}$ of problems over $\mathcal{K}$,

$\mathcal{C}_\mathcal{K}^\mathcal{L} = \bigcup_{L\in\mathcal{L}} \mathcal{C}_\mathcal{K}^L$.

1.3.3 Higher Sequential Complexity Classes

Now that we can define complexity classes with oracles to other complexity classes, we can define inductively the *polynomial hierarchy*, first introduced by [Sto76] over $\mathbb{Z}_2$.

Definition 1.3.4 (Polynomial Hierarchy)
Let $\Sigma^0_\mathcal{K} = \Pi^0_\mathcal{K} = \Delta^0_\mathcal{K} = \mathrm{P}_\mathcal{K}$ and, for $i \geq 1$,

$$\begin{aligned}\Delta^i_\mathcal{K} &= \mathrm{P}_\mathcal{K}^{\Sigma^{i-1}_\mathcal{K}}\\ \Sigma^i_\mathcal{K} &= \mathrm{NP}_\mathcal{K}^{\Sigma^{i-1}_\mathcal{K}}\\ \Pi^i_\mathcal{K} &= \mathrm{coNP}_\mathcal{K}^{\Sigma^{i-1}_\mathcal{K}}\end{aligned}$$

The *polynomial time hierarchy* over $\mathcal{K}$ is

$$\mathrm{PH}_\mathcal{K} = \bigcup_{i=0}^{\infty} \Sigma^i_\mathcal{K} = \bigcup_{i=0}^{\infty} \Pi^i_\mathcal{K} = \bigcup_{i=0}^{\infty} \Delta^i_\mathcal{K}.$$

A function in $\mathrm{F}\Delta^i_\mathcal{K}$ is a polynomial time function over $\mathcal{K}$ which queries $\Sigma^i_\mathcal{K}$ oracles:

$$\mathrm{F}\Delta^i_\mathcal{K} = \mathrm{FP}_\mathcal{K}^{\Sigma^{i-1}_\mathcal{K}} = \mathrm{FP}_\mathcal{K}^{\Pi^{i-1}_\mathcal{K}}.$$

The *functional polynomial time hierarchy* over $\mathcal{K}$ is

$$\mathrm{FPH}_\mathcal{K} = \bigcup_{i=0}^{\infty} \mathrm{F}\Delta^i_\mathcal{K}.$$

The first levels of the polynomial hierarchy are previously defined complexity classes: $\Sigma^0_\mathcal{K} = \mathrm{P}_\mathcal{K}, \Sigma^1_\mathcal{K} = \mathrm{NP}_\mathcal{K}, \Pi^1_\mathcal{K} = \mathrm{coNP}_\mathcal{K}$.

Clearly, the following inclusions hold:

$$\begin{aligned}\forall i \in \mathbb{N},\quad \Sigma^i_\mathcal{K} &\subseteq \Delta^{i+1}_\mathcal{K} &&\subseteq \Sigma^{i+1}_\mathcal{K}\\ \Pi^i_\mathcal{K} &\subseteq \Delta^{i+1}_\mathcal{K} &&\subseteq \Pi^{i+1}_\mathcal{K}\end{aligned}$$

On our three main structures, it is still unknown whether these inclusions are strict or not. This is an extension of the open question $\mathrm{P}_\mathcal{K} = \mathrm{NP}_\mathcal{K}$, and, if two classes in the hierarchy coincide, then the higher levels of the hierarchy collapse to these classes [BCSS98, Poi95].

Recall that in our general setting we do not consider complexity classes defined with space as a resource. In particular, we do not have a notion of polynomial space. However, the following non deterministic complexity class, when considered over $\mathbb{Z}_2$, coincides with PSPACE.

Definition 1.3.5 (Polynomial Alternating Time)

A set $S \subseteq \mathbb{K}^*$ belongs to $\mathrm{PAT}_{\mathcal{K}}$ 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 $\overline{x} \in \mathbb{K}^*$,

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

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

The inclusion $\mathrm{PH}_{\mathcal{K}} \subseteq \mathrm{PAT}_{\mathcal{K}}$ is clear, and, once again, on $\mathbb{Z}_2$ and on $\mathbb{R}_{ovs}$, it is unknown whether it is strict or not. On $\mathbb{R}$, it has been proved to be strict in [Cuc93]. All inclusions are summarized on the next Figure 1.2.

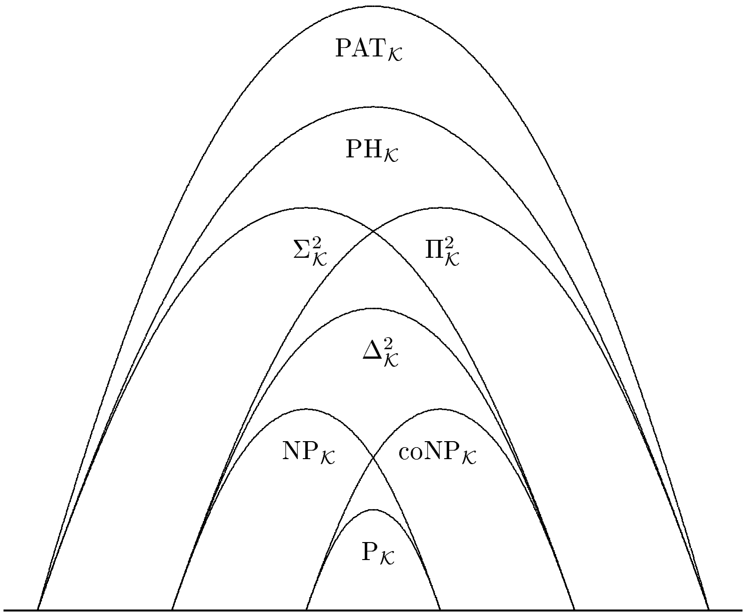

Figure 1.2: Inclusion Relations between Sequential Complexity Classes over $\mathcal{K}$

1.3.4 Digital Non-Determinisitc Sequential Classes

Using existential and universal quantifiers over $\{\mathbf{0}, \mathbf{1}\}$ instead of quantifiers over $\mathbb{K}$ allows us to define digital versions of the complexity classes above as follows.

Definition 1.3.6 (Digital Non-Deterministic Polynomial Time)

A decision problem A is in $\mathrm{DNP}_{\mathcal{K}}$ 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 $\overline{y} \in \{\mathbf{0},\mathbf{1}\}^*$ with $|\overline{y}| \le p_B(|\overline{x}|)$ satisfying $(\overline{x},\overline{y}) \in B$.

A decision problem A is in $\mathrm{coDNP}_{\mathcal{K}}$ 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 $\overline{y} \in \{\mathbf{0},\mathbf{1}\}^*$ with $|\overline{y}| \le P_B(|\overline{x}|)$, $(\overline{x},\overline{y})$ is in B.

Definition 1.3.7 (Digital Polynomial Hierarchy)
Let $\mathrm{D\Sigma}^0_{\mathcal{K}} = \mathrm{D\Pi}^0_{\mathcal{K}} = \mathrm{D\Delta}^0_{\mathcal{K}} = \mathrm{P}_{\mathcal{K}}$ and, for $i \ge 1$,

$$\begin{aligned} \mathrm{D\Delta}^i_{\mathcal{K}} &= \mathrm{P}_{\mathcal{K}}^{\Sigma^{i-1}_{\mathcal{K}}} \\ \mathrm{D\Sigma}^i_{\mathcal{K}} &= \mathrm{DNP}_{\mathcal{K}}^{\Sigma^{i-1}_{\mathcal{K}}} \\ \mathrm{D\Pi}^i_{\mathcal{K}} &= \mathrm{coDNP}_{\mathcal{K}}^{\Sigma^{i-1}_{\mathcal{K}}} \end{aligned}$$

The *digital polynomial time hierarchy* over $\mathcal{K}$ is

$$\mathrm{DPH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{D\Sigma}^i_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{D\Pi}^i_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{D\Delta}^i_{\mathcal{K}}.$$

A function in $\mathrm{DF\Delta}^i_{\mathcal{K}}$ is a polynomial time function over $\mathcal{K}$ which queries $\mathrm{D\Sigma}^i_{\mathcal{K}}$ oracles:

$$\mathrm{DF\Delta}^i_{\mathcal{K}} = \mathrm{FP}_{\mathcal{K}}^{\mathrm{D\Sigma}^{i-1}_{\mathcal{K}}} = \mathrm{FP}_{\mathcal{K}}^{\mathrm{D\Pi}^{i-1}_{\mathcal{K}}}.$$

The *functional digital polynomial time hierarchy* over $\mathcal{K}$ is

$$\mathrm{DFPH}_{\mathcal{K}} = \bigcup_{i=0}^{\infty} \mathrm{DF\Delta}^i_{\mathcal{K}}.$$

And, similarly,

Definition 1.3.8 *A set $S \subseteq \mathbb{K}^*$ belongs to* $\mathrm{DPAT}_{\mathcal{K}}$ (digital 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 $\overline{x} \in \mathbb{K}^*$,*

$$\overline{x} \in S \quad \Leftrightarrow \quad \exists a_1 \in \{\mathbf{0},\mathbf{1}\}\, \forall b_1 \in \{\mathbf{0},\mathbf{1}\} \,\ldots\, \exists a_{q(|\overline{x}|)} \in \{\mathbf{0},\mathbf{1}\}\, \forall b_{q(|\overline{x}|)} \in \{\mathbf{0},\mathbf{1}\}$$
$$M_S \text{ accepts } (\overline{x}, a_1.b_1 \ldots a_{q(|\overline{x}|)}.b_{q(|\overline{x}|)}).$$

In addition, we define $\mathrm{DFPAT}_{\mathcal{K}} = \mathrm{FP}_{\mathcal{K}}^{\mathrm{DPAT}_{\mathcal{K}}}$.

It is clear that over the discrete structure $\mathbb{Z}_2$, non-determinism is always digital, thus $\mathrm{DNP}_{\mathbb{Z}_2} = \mathrm{NP}$, etc... Moreover, over $\mathbb{R}_{ovs}$, we have the following:

Theorem 1.3.1 *[CK95]*

$$\begin{array}{rrcl} \forall i \in \mathbb{N} & \mathrm{D}\Delta_{\mathsf{add}} & = & \Delta_{\mathsf{add}} \\ & \mathrm{D}\Sigma_{\mathsf{add}} & = & \Sigma_{\mathsf{add}} \\ & \mathrm{D}\Pi_{\mathsf{add}} & = & \Pi_{\mathsf{add}} \end{array}$$

The proof of this striking result is based on the following result, proved in [Koi94]:

Proposition 1.3.4 *Let P be a polyhedron of $\mathbb{R}^n$, defined by polynomial inequalities of the kind $\Sigma_{i=1}^n a_i x_i + a_0 < 0$ and $\Sigma_{i=1}^n a_i x_i + a_0 \leq 0$, where the a_i are rational numbers of bit-size bounded by L. If $P \neq \emptyset$, P contains at least one point whose coordinates are rational numbers of bit-size bounded by $(Ln)^{O(1)}$.*

Over $\mathbb{R}$, no such result exists, and it is an open question whether any of the non-deterministic classes defined above coincides with its digital version.

1.3.5 A Parallel Model of Computation

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 in terms of circuits given by Proposition 1.3.5 below.[2]

Definition 1.3.9 $\mathrm{FPAR}_{\mathcal{K}}$ 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 $\overline{x} \in \mathbb{K}^*$. $\mathrm{PAR}_{\mathcal{K}}$ is the class of problems decidable in polynomial time by a parallel machine using an exponentially bounded number of processors.

Proposition 1.3.5 *[BCSS98] A function $f : \mathbb{K}^* \to \mathbb{K}^*$ is in $\mathrm{FPAR}_{\mathcal{K}}$ if and only if $f(\overline{x})$ is computed by a P-uniform family of circuits $\{\mathscr{C}_n(x_1, \ldots, x_n)|\ n \in \mathbb{N}\}$ of polynomial depth.*

We also have the following relations:

$$\mathrm{PAT}_{\mathbb{Z}_2} = \mathrm{PAR}_{\mathbb{Z}_2} = \mathrm{PSPACE},$$

and [Cuc93]:

$$\mathrm{PH}_{\mathbb{R}} \subset \mathrm{PAR}_{\mathbb{R}} \subset \mathrm{PAT}_{\mathbb{R}}.$$

[2] The exposition on parallelism in [BCSS98, Chapter 18] is for $\mathcal{K}$ the real numbers but the definition of parallel machine as well as the proof of Proposition 1.3.5 carry on to arbitrary structures.

From the first relation $\mathrm{PAR}_{\mathbb{Z}_2} = \mathrm{PAT}_{\mathbb{Z}_2} = \mathrm{PSPACE}$, one can consider $\mathrm{PAR}_{\mathcal{K}}$ and $\mathrm{PAT}_{\mathcal{K}}$ as two different class over an arbitrary structure which are analogies of the classical PSPACE, defined independently of the notion of space as a resource. The same holds with $\mathrm{DPAT}_{\mathcal{K}}$ since $\mathrm{DPAT}_{\mathbb{Z}_2} = \mathrm{PAT}_{\mathbb{Z}_2}$. An natural question arising on any structure $\mathcal{K}$ is whether $\mathrm{PAT}_{\mathcal{K}}$, $\mathrm{DPAT}_{\mathcal{K}}$ and $\mathrm{PAR}_{\mathcal{K}}$ coincide as they do when $\mathcal{K} = \mathbb{Z}_2$. The answer is unknown on $\mathbb{R}_{ovs}$ as well as on $\mathbb{R}$.

1.4 Reductions and Completeness

All complexity classes introduced so far contain an infinity of problems. However, as shown on Figure 1.2, some problems may belong to a class, and not to one of its subclasses. These problems seem then to be more difficult to solve, or harder, than the ones in the subclass. The notions of reductions introduced below, and the afferent notion of complete problems, give a formal and precise description of this concept of a problem being more difficult than another. Reducing a problem A to a problem B means that A is "easier" to solve than B: given an efficient algorithm solving B, one can deduce an efficient algorithm solving A. We shall consider two types of reductions.

1.4.1 Reductions

Definition 1.4.1 (Many-one Reduction)
Let $A \subset \mathbb{K}^*$ and $B \subset \mathbb{K}^*$ be two decisions problems over $\mathcal{K}$. We say that there exists a *many-one reduction* from A to B, written as $A \preceq_m B$, if there exists a function $f : \mathbb{K}^* \to \mathbb{K}^*$ in $\mathrm{FP}_{\mathcal{K}}$ such that, for all $\overline{x} \in \mathbb{K}^*$, $\overline{x} \in A$ if and only if $f(\overline{x}) \in B$.

Definition 1.4.2 (Turing Reduction)
Let $A \subset \mathbb{K}^*$ and $B \subset \mathbb{K}^*$ be two decisions problems over $\mathcal{K}$. We say that there exists a *Turing reduction* from A to B, written as $A \preceq_T B$, if $A \in \mathrm{P}_{\mathcal{K}}^B$.

Clearly, $A \preceq_m B$ implies $A \preceq_T B$.

1.4.2 Completeness

Definition 1.4.3 (Hardness)
Consider a complexity class $\mathcal{C}_{\mathcal{K}}$ over $\mathcal{K}$, and a reduction $\preceq$. A problem $B \subset \mathbb{K}^*$ is $\mathcal{C}_{\mathcal{K}}$-hard if and only if

$$\forall A \in \mathcal{C}_{\mathcal{K}}, \quad A \preceq B.$$

Definition 1.4.4 (Completeness)

Consider a complexity class $\mathcal{C}_\mathcal{K}$ over $\mathcal{K}$, and a reduction $\preceq$. A problem $B \subset \mathbb{K}^*$ is $\mathcal{C}_\mathcal{K}$-complete if and only if it is in $\mathcal{C}_\mathcal{K}$ and is $\mathcal{C}_\mathcal{K}$-hard.

Over any structure $\mathcal{K}$, a natural $\mathrm{NP}_\mathcal{K}$-complete problem under many-one reductions is the following [Goo94, Poi95]:

Definition 1.4.5 (Circuit Satisfiability)

Given a circuit $\mathscr{C}$ over $\mathcal{K}$ with n input gates, decide whether there exists $\overline{x} \in \mathbb{K}^n$ such that $\mathscr{C}(\overline{x}) = \mathbf{1}$. This problem is denoted as $\mathrm{CSAT}_\mathcal{K}$.

Quantified versions of this problem yield completeness results for any class in the polynomial hierarchy, as follows:

Definition 1.4.6 ($\Sigma^k\mathrm{CSAT}_\mathcal{K}$ and $\Pi^k\mathrm{CSAT}_\mathcal{K}$)

For any $k \geq 1$,

$\Sigma^k\mathrm{CSAT}_\mathcal{K}$ is the following problem: Given a circuit $\mathscr{C}$ over $\mathcal{K}$ with $n_1 + \ldots + n_k$ input gates, decide whether

$$\exists\overline{x_1} \in \mathbb{K}^{n_1}\, \forall\overline{x_2} \in \mathbb{K}^{n_2} \ldots Q\overline{x_k} \in \mathbb{K}^{n_k},\ \mathscr{C}(\overline{x_1}, \ldots, \overline{x_k}) = \mathbf{1},$$

where $Q \in \{\exists, \forall\}$ differs from the previous quantifier. $\Sigma^k\mathrm{CSAT}_\mathcal{K}$ is $\Sigma^k_\mathcal{K}$-complete under many-one reductions.

$\Pi^k\mathrm{CSAT}_\mathcal{K}$ is the following problem: Given a circuit $\mathscr{C}$ over $\mathcal{K}$ with $n_1 + \ldots + n_k$ input gates, decide whether

$$\forall\overline{x_1} \in \mathbb{K}^{n_1}\, \exists\overline{x_2} \in \mathbb{K}^{n_2} \ldots Q\overline{x_k} \in \mathbb{K}^{n_k},\ \mathscr{C}(\overline{x_1}, \ldots, \overline{x_k}) = \mathbf{1},$$

where $Q \in \{\exists, \forall\}$ differs from the previous quantifier. $\Pi^k\mathrm{CSAT}_\mathcal{K}$ is $\Pi^k_\mathcal{K}$-complete under many-one reductions.

In any of our main three structures, many other natural $\mathrm{NP}_\mathcal{K}$-complete problems under many-one reductions have been found. Over $\mathbb{Z}_2$, the notion of NP-completeness was first introduced by Cook [Coo71] and Levin [Lev73], and developed by Karp [Kar72] who exhibited 21 NP-complete problems. For a guide on classical NP-completeness, see [GJ79]. Moreover, complete problems under many-one reductions for all $\Delta^i_\mathcal{K}$, $\Sigma^i_\mathcal{K}$, $\Pi^i_\mathcal{K}$, $\mathrm{PAT}_\mathcal{K}$ and their digital versions, as well as for $\mathrm{PAR}_\mathcal{K}$, exist for any structure $\mathcal{K}$. See [Goo94] for $\mathrm{NP}_\mathcal{K}$-completeness results.

1.4.3 Counting Problems

So far we have seen two types of problems and of complexity classes: decision problems, which ask whether a desired solution exists, and computation problems, which require that a function be computed. But there is a third kind of problems, introduced over the discrete structure by Valiant [Val79b], and later on extended over the reals by Meer [Mee00] and Bürgisser and Cucker [BC04] denoted as counting problems, which ask how many solutions to a decision problem exist.

The main corresponding complexity class is the following:

Definition 1.4.7 (Counting Polynomial Time)
Let $\mathcal{K}$ be a structure. $\#\mathrm{P}_{\mathcal{K}}$ is the class of functions $\#L : \mathbb{K}^* \to \mathbb{N} \cup \{\infty\}$ such that there exists a polynomial time machine M_L over $\mathcal{K}$ with polynomial time bound $p(n)$, and such that for all $\overline{x} \in \mathbb{K}^*$,

$$\#L(\overline{x}) = \mathrm{card}\left(\left\{\overline{y} \in \mathbb{K}^{p(|\overline{x}|)} : \ M_L \text{ accepts on input } (\overline{x}, \overline{y})\right\}\right).$$

As above, reductions between counting problems have been introduced, and complete problems have been found.

Definition 1.4.8 (Parsimonious Reductions)
Let $A : \mathbb{K}^* \to \mathbb{N} \cup \{\infty\}$ and $B : \mathbb{K}^* \to \mathbb{N} \cup \{\infty\}$ be two counting problems over $\mathcal{K}$. We say that there exists a *parsimonious reduction* from A to B, written as $A \preceq_p B$, if there exists a function $f : \mathbb{K}^* \to \mathbb{K}^*$ in $\mathrm{FP}_{\mathcal{K}}$ such that, for all $\overline{x} \in \mathbb{K}^*$, $A(\overline{x}) = B(f(\overline{x}))$.

Unless otherwise stated, all hardness and completeness results for counting problems will be assumed to hold for parsimounious reductions. An interesting result, showing the relevance of this class in the discrete setting, is the following.

Theorem 1.4.1 *[Val79a] Computing the permanent of a matrix over $\mathbb{Z}$ is $\#\mathrm{P}$-complete.*

Over an arbitrary structure, the following natural problem is $\#\mathrm{P}_{\mathcal{K}}$-complete [BC04]:

Definition 1.4.9 ($\#\mathrm{CSAT}_{\mathcal{K}}$)
Given a circuit $\mathscr{C}$ over $\mathcal{K}$ with n input gates, count the number of $\overline{x} \in \mathbb{K}^n$ such that $\mathscr{C}(\overline{x}) = \mathbf{1}$.

Many other completeness results for natural problems over $\mathbb{Z}_2$ have been found, and Toda [Tod90] proved the following

Theorem 1.4.2 *[Tod90]*

$$\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}.$$

This theorem establishes the computational power of counting. Moreover, Toda and Watanabe [TW92] introduced a principle for assigning to any class of decision problems a corresponding class of counting problems:

Definition 1.4.10 Given a set $A \in \mathbb{K}^*$ and a polynomial p, $\#_A^p : \mathbb{K}^* \to \mathbb{N} \cup \infty$ is the function satisfying, for all $\overline{x} \in \mathbb{K}^*$,

$$\#_A^p(\overline{x}) = |\{\overline{y} \in \mathbb{K}^{p(|\overline{x}|)} : (\overline{x}, \overline{y}) \in A\}|.$$

If $\mathcal{C}$ is a class of decision problems,

$$\# \cdot \mathcal{C} = \{\#_A^p : A \in \mathcal{C} \text{ and } p \text{ a polynomial } \}.$$

Similarly, one assigns $\mathrm{D}\# \cdot \mathcal{C}$ to a digital non-deterministic class $\mathcal{C}$ over $\mathcal{K}$.

Based on previous results by Meyer auf der Heide [Mey84, Mey88] and Fournier and Koiran [FK98, FK00], Bürgisser and Cucker have later on extended the study of counting problems to the additive setting $\mathbb{R}_{ovs}$, and proved several transfer theorems between classical complexity classes and classes over $\mathbb{R}_{ovs}$. The main interesting feature in theses results is that they link complexity classes over $\mathbb{R}_{ovs}$ with classical ones over $\mathbb{Z}_2$, through oracle calls as described in Remark 1.3.1.

Theorem 1.4.3 *[BC04] The following statements hold ($k \geq 0$):*

1. $\Sigma^k_{\mathsf{add}} \subseteq \mathrm{P}^{\Sigma^k}_{\mathsf{add}}$, $\Pi^k_{\mathsf{add}} \subseteq \mathrm{P}^{\Pi^k}_{\mathsf{add}}$, $\mathrm{PH}_{\mathsf{add}} \subseteq \mathrm{P}^{\mathrm{PH}}_{\mathsf{add}}$.
2. $\mathrm{D}\# \cdot \Sigma^k_{\mathsf{add}} \subseteq \mathrm{FP}^{\#\Sigma^k}_{\mathsf{add}}$, $\mathrm{D}\# \cdot \Pi^k_{\mathsf{add}} \subseteq \mathrm{FP}^{\#\Pi^k}_{\mathsf{add}}$, $\mathrm{D}\# \cdot \mathrm{PH}_{\mathsf{add}} \subseteq \mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$.
3. $\mathrm{PAR}_{\mathsf{add}} = \mathrm{P}^{\mathrm{PSPACE}}_{\mathsf{add}}$.
4. $\mathrm{FPAR}_{\mathsf{add}} = \mathrm{FP}^{\mathrm{PAR}_{\mathsf{add}}}_{\mathsf{add}} = \mathrm{FP}^{\mathrm{PSPACE}}_{\mathsf{add}}$.
5. $\mathrm{FP}^{\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}} = \mathrm{FP}^{\mathrm{D}\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}} = \mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$.

As emphasized in the introduction, these transfer results show the relevance of the structure $\mathbb{R}_{ovs}$, which allows to express many geometrical and topological properties in the set of the real numbers, but which is also intrisincly related to classical Turing machines over finite structures in terms of complexity.

Chapter 2

The Complexity of Semilinear Problems

In this chapter we focus on the study of *semilinear sets* $S \subseteq \mathbb{R}^n$, which are derived from closed halfspaces by taking a finite number of unions, intersections, and complements. Moreover, we assume that the closed halfspaces are given by linear inequalities of "mixed-type" $a_1x_1 + \ldots + a_nx_n \leq b$, with integer coefficients a_i and real right hand side b.

Semilinear sets can be represented in a very compact way by additive circuits, i.e. by circuits over $\mathbb{R}_{ovs}$ as in Definition 1.2.1. The set of inputs accepted by an additive circuit is semilinear, and any semilinear set can be described this way. Therefore, we only consider the structure $\mathbb{R}_{ovs}$ in this chapter. Complexity results in this setting shed some light on their equivalent over $\mathbb{R}$, but are also closely related to the classical setting $\mathbb{Z}_2$, as shown by the transfer results of Theorem 1.4.3.

The basic problem $\mathrm{CSAT}_{\mathsf{add}}$ to decide whether the semilinear set S given by an additive circuit is nonempty turns out to be $\mathrm{NP}_{\mathsf{add}}$-complete [BCSS98]. By contrast, the feasibility question for a system of linear inequalities of the above mixed type is solvable in $\mathrm{P}_{\mathsf{add}}$. This is just a rephrasing of a well known result by Tardos [Tar86]. To understand this statement, one should note that an additive circuit of size $O(n)$ can actually decide a semilinear set defined with $O(2^n)$ linear inequalities of the mixed type above. Moreover, the integer coefficients of these linear inequalities have then a bit-size polynomial in n.

It is also noticeable that any $\overline{x} \in \mathbb{R}^*$ can be trivially encoded by an additive circuit $\mathscr{C}_{\overline{x}}$, with constant gates x_i. Therefore, circuits can be seen as natural inputs for additive machines, and many geometrical properties of the corresponding semilinear sets can be studied. Our purpose here is to provide a large landscape of natural complete problems for many of the complexity classes defined over $\mathbb{R}_{ovs}$. The results exposed in this chapter are all original and unpublished.

2.1 Notations and Conventions

If $\mathscr{C}$ is an additive circuit we denote by $F_{\mathscr{C}} : \mathbb{R}^n \to \mathbb{R}^m$ the function computed by the circuit.

A decision circuit $\mathscr{C}$ is an additive circuit with one output gate. The set $\overline{x} \in \mathbb{R}^n$ such that $F_{\mathscr{C}}(x_1, \ldots, x_n) = 1$ is the semilinear set decided by $\mathscr{C}$, and is denoted by $S_{\mathscr{C}}$.

If E is a set of equalities of inequalities with n variables, we denote by $\mathcal{S}(E)$ its set of solutions in $\mathbb{R}^n$.

In order to simplify our proofs, we will consider the selection nodes of the circuits to have indegree 4 (instead of 3 as above) and compute, with input (x, a, b, c),

$$\text{if } x < 0 \text{ then } a, \text{ elsif } x = 0 \text{ then } b, \text{ else } c.$$

This is w.l.o.g. since one can pass from one form of circuit to the other in polynomial time.

Following the lines of [BCSS98] and [ZS79], we define the Zariski topology for semilinear sets.

Definition 2.1.1 (Zariski Closure)
A semilinear set $S \subseteq \mathbb{R}^n$ is *Zariski closed* if it is a finite union of affine subspaces of $\mathbb{R}^n$. The *Zariski closure* of a set $V \subseteq \mathbb{R}^n$, denoted by $\overline{V}^Z$, is the smallest Zariski-closed semilinear subset of $\mathbb{R}^n$ containing V.

Remark 2.1.1 The use of the words "closed" or "closure" is adequate. The Zariski-closed sets satisfy the axioms of the closed sets of a topology.

Definition 2.1.2 (Zariski Irreducibility)
A set $S \subseteq \mathbb{R}^n$ is *Zariski-irreducible* when its Zariski closure is an affine space.

For any nonempty set $S \subseteq \mathbb{R}^n$, its Zariski closure is a non-redundant finite union of nonempty affine subspaces $A_1, \ldots, A_s$ of $\mathbb{R}^n$. We call $A_1, \ldots, A_s$ the *irreducible components* of $\overline{S}$ and we call the sets $S \cap A_i$ the *irreducible components* of S.

Let $\mathscr{C}$ be an additive circuit and let s be the number of its selection gates. A *path* is an element in $\{-1, 0, 1\}^s$. We say that $x \in \mathbb{R}^n$ *follows* a path γ when, for $j = 1, \ldots, s$, the result of the test performed at the jth selection gate is γ_j (i.e., -1 if the tested value is < 0, 0 if it is $= 0$, and 1 if it is > 0). Given a path γ, its *leaf set* is

$$D_\gamma = \{x \in \mathbb{R}^n \mid x \text{ follows } \gamma\}.$$

Let $I_\gamma = \{i \leq s \mid \gamma_i \neq 0\}$ and $E_\gamma = \{j \leq s \mid \gamma_j = 0\}$. That is, D_γ is defined by a set of affine inequalities $F_i < 0$, $i \in I_\gamma$, and a set of equalities $F_j = 0$, $j \in E_\gamma$. Therefore, D_γ is convex, and, if $D_\gamma \neq \emptyset$, its Zariski closure is the affine space defined by the equalities $F_j = 0$, $j \in E_\gamma$, (the strict inequalities $F_i < 0$ define a subset of $\mathbb{R}^n$ of dimension n (since $D_\gamma \neq \emptyset$) and therefore they can not possibly be relevant for the definition of the closure of D_γ).

Moreover, by Theorem 1.3.1, the following problem is $\mathrm{NP}_{\mathsf{add}}$-complete under many-one reductions.

Definition 2.1.3 **(Circuit Boolean Satisfiability -** $\mathrm{CBS}_{\mathsf{add}}$**)**
$\mathrm{CBS}_{\mathsf{add}}$ is the following problem: given an additive circuit $\mathscr{C}$ with n input gates decide whether there exists $x \in \{0,1\}^n$ such that $\mathscr{C}(x) = 1$.

The following problem $\mathrm{DIM}_{\mathsf{add}}$ is also $\mathrm{NP}_{\mathsf{add}}$-complete under many-one reductions, see [BC04, Theorem 5.1].

Definition 2.1.4 **(Dimension)**
$\mathrm{DIM}_{\mathsf{add}}$ is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates $k \in \{0, \ldots, n\}$, decide whether the dimension of $S_{\mathscr{C}}$ is greater or equal than k.

We will often use the following functions

$$\mathsf{sign} : \mathbb{R} \to \{-1, 0, 1\}, \qquad \mathsf{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and

$$\mathsf{pos} : \mathbb{R} \to \{0, 1\}, \qquad \mathsf{pos}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0. \end{cases}$$

For $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, abusing notations we define $\mathsf{sign}(x) = (\mathsf{sign}(x_1), \ldots, \mathsf{sign}(x_n))$, and similarly, $\mathsf{pos}(x) = (\mathsf{pos}(x_1), \ldots, \mathsf{pos}(x_n))$.

We will also often use the following notion:

Definition 2.1.5 A *quadrant* of $\mathbb{R}^n$ is a subset of $\mathbb{R}^n$ containing all points x such that:

$$x = ((-1)^{s_1}\lambda_1, \ldots, (-1)^{s_n}\lambda_n) \quad , s_i \in \{0,1\}, \lambda_i > 0.$$

Note that $\mathbb{R}^2$ is, modulo a sparse set of points, divided equally into four quadrants. Our notion of quadrant is a generalization of this concept to $\mathbb{R}^n$.

The following lemma states a technical result which will be used later on in many proofs.

Lemma 2.1.1 *Given a decision additive circuit $\mathscr{C}$, an accepting path γ and a point $x \in D_\gamma$, one can compute the dimension of D_γ in time polynomial in the size of $\mathscr{C}$.*

PROOF. Since $\mathscr{C}$ is an additive circuit, all affine inequalities in I_γ and equalities in E_γ defining D_γ have integer coefficients of polynomial bit-size. Then, computing linear combination of equalities of E_γ, with integer coefficients of polynomial bit-size, can be performed in polynomial time. Therefore, one can use gaussian elimination in to compute the dimension of the solution set of E_γ. Since D_γ is not empty by hypothesis, its dimension is the dimension of the solution set of E_γ.

□

2.2 Properties for the Euclidean Topology

In this section we study the complexity of several topological properties of semilinear sets, when the topology at hand is the euclidian topology.

2.2.1 Euclidian Adherence

Lemma 2.2.1 EADH$_{\mathsf{add}}$ (Euclidean Adherence) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates and a point $x \in \mathbb{R}^n$ decide whether x belongs to the Euclidean closure of $S_\mathscr{C}$. The problem* EADH$_{\mathsf{add}}$ *is* NP$_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Let us give first an NP$_{\mathsf{add}}$ algorithm solving EADH$_{\mathsf{add}}$. Let $\mathscr{C}$ be an additive circuit with n input gates and r selection gates, and denote by A the set of accepting paths of $\mathscr{C}$. Clearly,

$$\overline{S_\mathscr{C}} = \overline{\bigcup_{\gamma \in \mathcal{A}} D_\gamma} = \bigcup_{\gamma \in \mathcal{A}} \overline{D_\gamma}.$$

Therefore, x belongs to the Euclidean closure of $S_\mathscr{C}$ if and only if it belongs to the closure of a leaf set D_γ of $S_\mathscr{C}$. Such a leaf set is defined by a set of inequalities $F_i < 0$, $i \in I_\gamma$, and a set of equalities $F_i = 0$, $i \in E_\gamma$. Its Euclidean closure is the intersection of the solution set of the equalities and the Euclidean closures of the solution sets of the inequalities. Therefore, x belongs to this closure if and only if satisfies the conditions $F_i \leq 0$, $i \in I_\gamma$, and $F_i = 0$, $i \in E_\gamma$.

Our NP$_{\mathsf{add}}$ algorithm solving EADH$_{\mathsf{add}}$ is then the following,

input $(\mathscr{C}, x)$
guess a path $\gamma \in \{-1, 0, 1\}^r$
check that γ is accepting
check that D_γ is non empty
check that $F_i(x) \leq 0$, $i \in I_\gamma$, and $F_i(x) = 0$, $i \in E_\gamma$

Let us now prove the hardness: we reduce $\mathrm{CBS}_{\mathsf{add}}$ to $\mathrm{EADH}_{\mathsf{add}}$. Assume $\mathscr{C}$ is an additive circuit with n input gates. Consider the circuit $\mathscr{C}'$ computing the function

$$\begin{aligned} F_{\mathscr{C}} \circ \mathsf{pos}: \quad & \mathbb{R}^n \to \{0,1\} \\ & x \mapsto F_{\mathscr{C}}(\mathsf{pos}(x)). \end{aligned}$$

The mapping $\mathscr{C} \mapsto (\mathscr{C}', 0)$ reduces $\mathrm{CBS}_{\mathsf{add}}$ to $\mathrm{EADH}_{\mathsf{add}}$. Indeed, if $S_{\mathscr{C}} = \emptyset$ then $S_{\mathscr{C}'} = \emptyset$ as well and $0 \notin \overline{S_{\mathscr{C}'}}$. On the other hand, if $S_{\mathscr{C}} \neq \emptyset$ then $S_{\mathscr{C}'}$ contains at least one quadrant and $0 \in \overline{S_{\mathscr{C}'}}$.

$\square$

2.2.2 Euclidian Closedness

Lemma 2.2.2 $\mathrm{ECLOSED}_{\mathsf{add}}$ (Euclidean Closed) *is the following problem: given an additive decision circuit* $\mathscr{C}$ *decide whether* $S_{\mathscr{C}}$ *is closed under the Euclidean topology. The problem* $\mathrm{ECLOSED}_{\mathsf{add}}$ *is* $\mathrm{coNP}_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Let us show first that $\mathrm{ECLOSED}_{\mathsf{add}}$ belongs to $\mathrm{coNP}_{\mathsf{add}}$. Given an additive circuit $\mathscr{C}$ with n input gates, denote by $\mathcal{A}$ its accepting paths. Then, $S_{\mathscr{C}}$ is closed if and only if

$$\begin{aligned} & \forall x \in \mathbb{R}^n \quad x \in \overline{S_{\mathscr{C}}} \Rightarrow x \in S_{\mathscr{C}} \\ \Leftrightarrow \quad & \forall x \in \mathbb{R}^n \quad \left(\exists \gamma \in \mathcal{A}\; x \in \overline{D_\gamma}\right) \Rightarrow x \in S_{\mathscr{C}} \\ \Leftrightarrow \quad & \forall x \in \mathbb{R}^n \quad \left(\forall \gamma \in \mathcal{A}\; x \notin \overline{D_\gamma}\right) \vee x \in S_{\mathscr{C}} \end{aligned}$$

The predicate $x \notin \overline{D_\gamma}$ can be checked in $\mathrm{P}_{\mathsf{add}}$, as in the proof of Lemma 2.2.1.

For the hardness, we reduce $\mathrm{CBS}_{\mathsf{add}}$ to the complement of $\mathrm{ECLOSED}_{\mathsf{add}}$ with a reduction similar to the one of Lemma 2.2.1.

Assume $\mathscr{C}$ is an additive circuit with n input gates. Consider the circuit $\mathscr{C}'$ computing the function

$$F_{\mathscr{C}'}(x) = \begin{cases} 0 & \text{if } x = 0 \\ F_{\mathscr{C}}(\mathsf{pos}(x)) & \text{otherwise.} \end{cases}$$

Clearly, $S_{\mathscr{C}} = \emptyset$ implies $S_{\mathscr{C}'} = \emptyset$. And if $S_{\mathscr{C}'} \neq \emptyset$ then $S_{\mathscr{C}'}$ is not closed since the origin is in its closure but is not a solution.

$\square$

2.2.3 Euclidian Denseness

Lemma 2.2.3 $\mathrm{EDENSE}_{\mathsf{add}}$ (Euclidian Denseness) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates decide whether $S_{\mathscr{C}}$ is dense in $\mathbb{R}^n$. The problem* $\mathrm{EDENSE}_{\mathsf{add}}$ *is* $\mathrm{coNP}_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Given an additive circuit $\mathscr{C}$ with n input gates, $S_{\mathscr{C}}$ is dense in $\mathbb{R}^n$ if and only if its complement has dimension strictly less than n. This is in $\mathrm{coNP}_{\mathsf{add}}$ since $\mathrm{DIM}_{\mathsf{add}}$ is in $\mathrm{NP}_{\mathsf{add}}$ by [BC04, Theorem 5.1]. The hardness follows from the reduction of Lemma 2.2.1: $\mathscr{C}$ belongs to $\mathrm{CBS}_{\mathsf{add}}$ if and only if its complement is not dense in $\mathbb{R}^n$.

□

2.3 Other Basic Properties

2.3.1 Unboundedness

Lemma 2.3.1 $\mathrm{UNBOUNDED}_{\mathsf{add}}$ (Unboundedness) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates decide whether $S_{\mathscr{C}}$ is unbounded in $\mathbb{R}^n$. The problem* $\mathrm{UNBOUNDED}_{\mathsf{add}}$ *is* $\mathrm{NP}_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. In this proof, for $x \in \mathbb{R}$, $|x|$ denotes the absolute values of x. For the hardness we reduce $\mathrm{CSAT}_{\mathsf{add}}$ to $\mathrm{UNBOUNDED}_{\mathsf{add}}$. Let $\mathscr{C}$ be a circuit. Define $\mathscr{C}'$ by adding a dummy variable to $\mathscr{C}$. Then $S_{\mathscr{C}'}$ is a cylinder of base $S_{\mathscr{C}}$ satisfying that $S_{\mathscr{C}}$ is non-empty if and only if $S_{\mathscr{C}'}$ is so and in this case the latter is unbounded.

For the membership, assume $S_{\mathscr{C}}$ is bounded. Then, every accepting path γ defines a bounded leaf set D_γ. Then, $\overline{S_{\mathscr{C}}} = \bigcup_{\gamma \in \mathcal{A}} \overline{D_\gamma}$. Denote by y a point of $\overline{S_{\mathscr{C}}}$ which is at the greatest distance from the origin. y is then a vertex of one of the $\overline{D_\gamma}$. This vertex is therefore defined by n equalities: y is the solution of a system:

$$Ay = b,$$

where $A = (a_{ij})$ is a $n \times n$ matrix, $a_{ij} \in \mathbb{Z}$, and $|a_{ij}| \leq 2^{p(n)}$ for some polynomial p. The b_i are linear combinations of the constants $c_1, \ldots, c_k \in \mathbb{R}$ of the circuit $\mathscr{C}$ defining S,

$$b_i = \sum_{j=1}^{k} \beta_{ij} c_j$$

where $k \leq p(n)$, $\beta_{ij} \in \mathbb{Z}$, $|\beta_{ij}| \leq 2^{p(n)}$, and c_j denotes a constant of $\mathscr{C}$.

By Cramer's rule, for $i = 1, \ldots, n$,

$$y_i = \frac{\det(A_i^b)}{\det A}$$

where A_i^b denotes the matrix obtained by replacing in A the ith column by b. Since A is invertible, $\det A \neq 0$. Also, since $a_{ij} \in \mathbb{Z}$, $|\det A| \geq 1$. Denote by $(A)_{ij}$ the submatrix of A obtained by removing the i^{th} column and the j^{th} line. Denote also by σ_{ij}^n the signature of the permutation $\{1, \ldots, n\} \to \{1, \ldots, n\}$ exchanging i and j.

Developing $\det(A_i^b)$ along the ith column, we get

$$\det(A_i^b) = \sum_{j=1}^{n} \sigma_{ij} b_j \det((A_i^b)_{ij}).$$

Thus,

$$|\det(A_i^b)| \leq \sum_{j=1}^{n} |b_j| |\det((A_i^b)_{ij})|.$$

Clearly, $(A_i^b)_{ij} = A_{ij}$, therefore, $|\det(A_{ij})| \leq n! 2^{np(n)}$.

It follows by Stirling that $|y_i| \leq c \cdot p(n) 2^{p(n)} n! 2^{np(n)} \leq c \cdot p(n) 2^{p(n)+p(n)^4}$ for $c = \max |c_j|$. This bound is clearly computable in $\mathrm{P}_{\mathsf{add}}$.

An $\mathrm{NP}_{\mathsf{add}}$ algorithm for UNBOUNDED$_{\mathsf{add}}$ now easily follows. Given a circuit $\mathscr{C}$, guess a point $y \in \mathbb{R}^n$ with $\|y\|_\infty \geq c \cdot p(n) 2^{p(n)+p(n)^4}$. Then accept if and only if $\mathscr{C}(y) = 1$.

□

2.3.2 Isolated points

Lemma 2.3.2 ISOLATED$_{\mathsf{add}}$ (Isolatedness) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates and a point $x \in \mathbb{R}^n$ decide whether x is isolated in $S_{\mathscr{C}}$. The problem* ISOLATED$_{\mathsf{add}}$ *is* $\mathrm{coNP}_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Membership easily follows from the equivalence

$$x \text{ not isolated in } S \Leftrightarrow x \notin S \;\vee\; x \in \overline{S \setminus \{x\}}$$

and the membership of EADH$_{\mathsf{add}}$ to $\mathrm{NP}_{\mathsf{add}}$.

Hardness follows from the equivalence

$$x \in \overline{S} \Leftrightarrow x \in S \;\vee\; x \text{ not isolated in } S \cup \{x\}$$

which gives a many-one reduction from EADH$_{\mathsf{add}}$ to the complement of ISOLATED$_{\mathsf{add}}$ and from Lemma 2.2.1.

□

Corollary 2.3.1 $\text{EXISTISO}_{\mathsf{add}}$ (Existence of Isolated Points) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates decide whether there exists $x \in \mathbb{R}^n$ isolated in $S_{\mathscr{C}}$. The problem* $\text{EXISTISO}_{\mathsf{add}}$ *is Σ^2_{add}-complete under many-one reductions.*

□

Theorem 2.3.1 $\#\text{ISO}_{\mathsf{add}}$ (Counting Isolated Points) *is the following problem: given an additive decision circuit $\mathscr{C}$ count the number of isolated points in $S_{\mathscr{C}}$. The problem* $\#\text{ISO}_{\mathsf{add}}$ *is Turing-complete in* $\text{FP}_{\mathsf{add}}^{\#\text{P}_{\mathsf{add}}}$.

PROOF. Let us first describe briefly a $\text{D}\Sigma^2_{\mathsf{add}}$ algorithm for $\text{EXISTISO}_{\mathsf{add}}$, such that its existential digital witnesses are in one-to-one correspondence with the isolated points of the solution set of the input.

input $\mathscr{C}$
guess a path $\gamma \in \{-1, 0, 1\}^n$
check that $\dim(D_\gamma) = 0$, otherwise HALT and REJECT
denote by x the point in D_γ
check that x is isolated in $S_{\mathscr{C}}$, otherwise HALT and REJECT
ACCEPT

The predicate $\dim(D_\gamma) = 0$ can be checked in deterministic polynomial time. Indeed, use Lemma 2.1.1 to compute the dimension of the solution set of E_γ. If this solution set has dimension 0, then it is reduced to one point x computable in polynomial time and it suffices to check that x satisfies all inequalities of I_γ. This can also be performed in polynomial time.

The predicate "x is isolated in $S_{\mathscr{C}}$" is in $\text{coNP}_{\mathsf{add}}$ by Lemma 2.3.2, and thus in $\text{coDNP}_{\mathsf{add}}$ by Theorem 1.3.1. Therefore the whole algorithm above is in $\text{D}\Sigma^2_{\mathsf{add}}$. Clearly, a point x isolated in $S_{\mathscr{C}}$ is a leaf set of $\mathscr{C}$, which shows the required one-to-one correspondence. It follows that

$$\#\text{ISO}_{\mathsf{add}} \in \text{D}\# \cdot \text{coNP}_{\mathsf{add}} \subseteq \text{FP}_{\mathsf{add}}^{\#\text{P}_{\mathsf{add}}}$$

the last by Theorem 1.4.3 (2).

For the hardness, by Theorem 1.4.3 (5), it is enough to reduce $\#\text{SAT}$ to $\#\text{ISO}_{\mathsf{add}}$. To do so, assume ϕ is an input for $\#\text{SAT}$. It is easy to compute from ϕ an additive circuit $\mathscr{C}$ which accepts only the subset of $\{0, 1\}^n$ consisting of the satisfying assignments of ψ. Then, by definition, every point in $S_{\mathscr{C}}$ is isolated, and the number of points in $S_{\mathscr{C}}$ is the number of satisfying assignments for ϕ.

□

2.3.3 Local Topological properties

Let us briefly recall the notion of local dimension of a set: given $S \subseteq \mathbb{R}^n$, and $x \in S$, for any open ball $\mathcal{B}_r(x)$ of radius r centered on x, we denote by d_r the dimension of $S \cap \mathcal{B}_r(x)$. The local dimension of S at the point x is then

$$\dim_x S = \min_{r>0} d_r,$$

i.e. the dimension of S in the closest neighborhoods of x.

Lemma 2.3.3 $\textsc{LocDim}_{\mathsf{add}}$ (Local Dimension) *is the following problem: given an additive decision circuit $\mathscr{C}$, a point $x \in S_{\mathscr{C}}$ and an integer $d \in \mathbb{N}$, decide whether the local dimension of $S_{\mathscr{C}}$ at the point x, $\dim_x S_{\mathscr{C}}$, is greater or equal than d. The problem* $\textsc{LocDim}_{\mathsf{add}}$ *is* $\mathrm{NP}_{\mathsf{add}}$*-complete under many-one reductions.*

Proof. The membership follows from the remark that, for $x \in S$,

$$\dim_x S \geq d \Leftrightarrow \exists \gamma \in \mathcal{A}, \exists y \in D_\gamma, \quad x \in \overline{D_\gamma} \wedge \dim D_\gamma \geq d.$$

Clearly, the predicate $x \in \overline{D_\gamma}$ can be checked in $\mathrm{P}_{\mathsf{add}}$. The same holds for the predicate $\dim D_\gamma \geq d$ by Lemma 2.1.1. The hardness follows from the equivalence, for $x \in S$,

$$(x, S) \in \textsc{Isolated}_{\mathsf{add}} \Leftrightarrow \dim_x S < 1.$$

$\square$

Lemma 2.3.4 $\textsc{LocCont}_{\mathsf{add}}$ (Local Continuity) *is the following problem: given an additive circuit $\mathscr{C}$ with n input gates and a point $x \in \mathbb{R}^n$, decide whether $F_{\mathscr{C}}$ is continuous for the Euclidean topology at x. The problem* $\textsc{LocCont}_{\mathsf{add}}$ *is* $\mathrm{coNP}_{\mathsf{add}}$*-complete under many-one reductions.*

Proof.

Let us focus on the membership. for $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, denote by $\|x\|_\infty$ the norm $\max_i |x_i|$. Given a circuit $\mathscr{C}$, the local continuity of $F_{\mathscr{C}}$ in a point $x \in \mathbb{R}^n$ can be expressed in the following way:

$$\forall \epsilon > 0 \; \exists \eta > 0 \; \forall y \in \mathbb{R}^n \quad \|x - y\|_\infty < \eta \quad \Rightarrow \quad \|F_{\mathscr{C}}(x) - F_{\mathscr{C}}(y)\|_\infty < \epsilon. \tag{2.1}$$

Given $\epsilon > 0$, we define the following semilinear set S_x^ϵ by

$$y \in S_x^\epsilon \Leftrightarrow \|F_{\mathscr{C}}(x) - F_{\mathscr{C}}(y)\|_\infty < \epsilon.$$

The local continuity of $F_{\mathscr{C}}$ in x can then be expressed as follows:

$$\forall \epsilon > 0 \quad x \notin \overline{(\mathbb{R}^n \setminus S_x^\epsilon)}. \tag{2.2}$$

Indeed, this is equivalent to the following:

$$\begin{aligned} & \forall \epsilon > 0 \; \exists \eta > 0 \; \forall y \in \mathbb{R}^n \quad y \in \mathbb{R}^n \setminus S_x^\epsilon \Rightarrow \|x - y\|_\infty \geq \eta \\ \Leftrightarrow \quad & \forall \epsilon > 0 \; \exists \eta > 0 \; \forall y \in \mathbb{R}^n \quad \|F_{\mathscr{C}}(x) - F_{\mathscr{C}}(y)\|_\infty \geq \epsilon \Rightarrow \|x - y\|_\infty \geq \eta, \end{aligned}$$

the last one being the contraposition of (2.1).

Note that, given ϵ and x, a circuit for $\mathbb{R}^n \setminus S_x^\epsilon$ can be computed in polynomial time. In addition, $\text{EADH}_{\mathsf{add}}$ is in NP_{add} by Lemma 2.2.1. Therefore, (2.2) can be decided in $\text{coNP}_{\mathsf{add}}$ and with it, the continuity of $F_{\mathscr{C}}$ at $x \in \mathbb{R}^n$.

Let us now show the hardness. Consider the reduction of Lemma 2.2.2: the function $F_{\mathscr{C}'}$ is continuous at the origin if and only if $S_{\mathscr{C}} = \emptyset$. Therefore it reduces $\text{CBS}_{\mathsf{add}}$ to the complement of $\text{LocCont}_{\mathsf{add}}$.

□

2.3.4 Continuity and Discontinuities

Lemma 2.3.5 $\text{Cont}_{\mathsf{add}}$ (Euclidean Continuity) *is the following problem: given an additive circuit $\mathscr{C}$ decide whether $F_{\mathscr{C}}$ is continuous for the Euclidean topology. The problem* $\text{Cont}_{\mathsf{add}}$ *is* $\text{coNP}_{\mathsf{add}}$*-complete under many-one reductions.*

Proof. The membership is a consequence of Lemma 2.3.4: it suffices to check the local continuity at all points. For the hardness, we reduce $\text{CSAT}_{\mathsf{add}}$ to the complement of $\text{Cont}_{\mathsf{add}}$: given an additive circuit $\mathscr{C}$ deciding a set $S_{\mathscr{C}}$, define $\mathscr{C}'$ computing:

$$F_{\mathscr{C}'}(x, y) = \begin{cases} 1 & \text{if } F_{\mathscr{C}}(x) = 1 \wedge y = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then, for all x: $F_{\mathscr{C}}(x) = 1$ if and only if $F_{\mathscr{C}'}$ is not continuous in $(x, 0)$. Therefore, $S_{\mathscr{C}}$ is not empty if and only if $F_{\mathscr{C}'}$ is not continuous.

□

Lemma 2.3.6 $\#\textsc{Disc}_{\mathsf{add}}$ (Counting Discontinuities) *is the following problem: given an additive circuit $\mathscr{C}$ count the number of points in $\mathbb{R}^n$ where $F_{\mathscr{C}}$ is not continuous for the Euclidean topology. The problem $\#\textsc{Disc}_{\mathsf{add}}$ is Turing-complete in* $\mathrm{FP}_{\mathsf{add}}^{\#\mathrm{P}_{\mathsf{add}}}$.

Proof. The reduction used in Lemma 2.3.5 turns out to be also a parsimonious reduction from $\#\mathrm{CSAT}_{\mathsf{add}}$ to $\#\textsc{Disc}_{\mathsf{add}}$, thus showing $\#\mathrm{P}_{\mathsf{add}}$-hardness of $\#\textsc{Disc}_{\mathsf{add}}$ under parsimonious reductions. This clearly implies $\mathrm{FP}_{\mathsf{add}}^{\#\mathrm{P}_{\mathsf{add}}}$ hardness under Turing reductions.

Let us now focus on the membership. Consider an additive circuit $\mathscr{C}$ computing a function $F_{\mathscr{C}}: \quad \mathbb{R}^n \to \mathbb{R}^m$. Denote by $\mathrm{Disc}(F_{\mathscr{C}})$ the set of discontinuities of $F_{\mathscr{C}}$:

$$\mathrm{Disc}(F_{\mathscr{C}}) = \{x \in \mathbb{R}^n : \quad F_{\mathscr{C}} \text{ is not continuous in } x\}.$$

It can be checked in Σ_{add}^2 whether $\mathrm{Disc}(F_{\mathscr{C}})$ has infinite cardinality. Indeed, on input $x \in \mathbb{R}^n$ and $\mathscr{C}$, by Lemma 2.3.4, it is checkable in $\mathrm{coNP}_{\mathsf{add}}$ whether x is in $\mathrm{Disc}(F_{\mathscr{C}})$. Therefore, when $\mathscr{C}$ is fixed, there exists an additive decision circuit $\mathscr{C}'_n$ with $n+k$ input gates deciding membership in $\mathrm{Disc}(F_{\mathscr{C}})$: $x \in \mathrm{Disc}(F_{\mathscr{C}}) \Leftrightarrow \forall y \in \mathbb{R}^k, \quad \mathscr{C}'_n$ accepts on input (x, y). Since $S_{\mathscr{C}'_n}$ is semilinear, the set $\mathrm{Disc}(F_{\mathscr{C}})$ has infinite cardinality if and only if it has dimension at least 1, if and only if it contains at least a line segment:

$$\mathrm{card}\,(\mathrm{Disc}(F_{\mathscr{C}})) = \infty \Leftrightarrow \exists x_1, x_2 \in \mathbb{R}^n \,:\, x \in [x_1, x_2] \Rightarrow \forall y \in \mathbb{R}^k\, F_{\mathscr{C}'_n}(x, y) = 1.$$

By Proposition 1.3.4, x_1 and x_2 can be chosen as linear functions in the constants of $\mathscr{C}'_n$ with rational coefficients of bit-size polynomial in n, from which it follows that the predicate $x \in [x_1, x_2]$ can be checked in $\mathrm{P}_{\mathsf{add}}$. The whole right hand side of the equivalence above is then checkable in Σ_{add}^2.

Assume now that $\mathrm{Disc}(F_{\mathscr{C}})$ is finite: when x is in $\mathrm{Disc}(F_{\mathscr{C}})$, by Proposition 1.3.4, x can be succintly described as a linear function in the constants of $\mathscr{C}'_n$ with rational coefficients of bit-size polynomial in n. In that case, $\#\textsc{Disc}_{\mathsf{add}}$ is computable in $D\#\Pi_{\mathsf{add}}^1$. So, an algorithm for computing $\#\textsc{Disc}_{\mathsf{add}}$ works as follows.

input $\mathscr{C}$
compute $\mathscr{C}'$ the circuit deciding $\mathrm{Disc}(F_{\mathscr{C}})$
if $\mathrm{card}(\mathrm{Disc}(F_{\mathscr{C}})) = \infty$ then output ∞ and HALT, else
compute $\#\textsc{Disc}_{\mathsf{add}}$ in $D\#\Pi_{\mathsf{add}}^1$

Since $\Sigma_{\mathsf{add}}^2 = \mathrm{D}\Sigma_{\mathsf{add}}^2$ by Theorem 1.3.1 once again, it is clear that this algorithm works in $\mathrm{D}\#\mathrm{PH}_{\mathsf{add}} \subseteq \mathrm{FP}_{\mathsf{add}}^{\#\mathrm{P}_{\mathsf{add}}}$,the last by Theorem 1.4.3 (2).

□

2.3.5 Surjectivity

Lemma 2.3.7 $\text{SURJ}_{\mathsf{add}}$ (Surjectivity) *is the following problem: given an additive circuit* $\mathscr{C}$ *decide whether* $F_{\mathscr{C}}$ *is surjective. The problem* $\text{SURJ}_{\mathsf{add}}$ *is* Π^2_{add}*-complete under many-one reductions.*

PROOF. The membership follows from the definition of surjectivity. Given an additive circuit $\mathscr{C}$, $F_{\mathscr{C}}$ is surjective if and only if:

$$\forall y \in \mathbb{R}^m \, \exists x \in \mathbb{R}^n \, F_{\mathscr{C}}(x) = y.$$

For the hardness, we reduce the Π^2_{add}-complete problem $\Pi^2\text{CSAT}_{\mathsf{add}}$ to $\text{SURJ}_{\mathsf{add}}$: $\mathscr{C} \in \Pi^2\text{CSAT}_{\mathsf{add}}$ if and only if

$$\forall y \in \mathbb{R}^m \, \exists x \in \mathbb{R}^n \, \mathscr{C}(x, y) = 1.$$

Given $\mathscr{C}$, define $\mathscr{C}'$ such that:

$$\mathscr{C}'(x,y) = \begin{cases} y & \text{if } \mathscr{C}(x,y) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

It is clear then that $F_{\mathscr{C}'}$ is surjective if and only if

$$\forall y \in \mathbb{R}^m \, \exists x \in \mathbb{R}^n \, \mathscr{C}(x, y) = 1.$$

□

2.3.6 Compactness

Lemma 2.3.8 $\text{COMPACT}_{\mathsf{add}}$ (Compactness) *is the following problem: given an additive decision circuit* $\mathscr{C}$ *decide whether* $S_{\mathscr{C}}$ *is compact. The problem* $\text{COMPACT}_{\mathsf{add}}$ *is* $\text{coNP}_{\mathsf{add}}$*-complete under many-one reductions*

PROOF. The membership follows from the membership of $\text{UNBOUNDED}_{\mathsf{add}}$ to NP_{add} and of $\text{ECLOSED}_{\mathsf{add}}$ to $\text{coNP}_{\mathsf{add}}$. The hardness follows from the reduction of Lemma 2.2.2.

□

2.3.7 Reachability and Connectedness

We recall from [BC04] the following.

Lemma 2.3.9 *[BC04]* $\text{REACH}_{\mathsf{add}}$ (Reachability) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates, and two points s and t in $\mathbb{R}^n$, decide whether s and t belong to the same connected component of $S_{\mathscr{C}}$. The problem* $\text{REACH}_{\mathsf{add}}$ *is* $\text{PAR}_{\mathsf{add}}$*-complete under Turing reductions. The same holds when restricted to problems in $\mathbb{R}^3$.*

Remark 2.3.1 While this is already proven for semilinear sets of arbitrary dimension in [BC04], we need an alternative proof that we can use later on for proving the next Theorem 2.3.2. Our graph-theoretic arguments are largely inspired by a similar result stated for graphs in [CSV84].

PROOF. By Theorem 1.4.3 (3), $\text{PAR}_{\mathsf{add}} = \text{P}_{\mathsf{add}}^{\text{PSPACE}}$, where any language in PSPACE is considered as a subset of $\{\mathbf{0},\mathbf{1}\}^* \subseteq \mathbb{K}^*$. Since we want to prove the hardness under Turing reductions, it is therefore enough to prove that $\text{REACH}_{\mathsf{add}}$ is PSPACE-hard. Let $L \subseteq \{0,1\}^*$ be any language in PSPACE. Then L is decided by a single tape deterministic Turing machine M with polynomial space bound function $p(n)$. Denote by Σ the (finite) alphabet of M, and by Q its set of nodes. For a fixed input length n, a *valid* configuration of M is an element c in the set $C_n = Q \times \{1,\dots,p(n)\} \times \Sigma^{p(n)}$. Let $T = \text{card}(C_n)$. Assuming that we have an enumeration of the nodes of M, we can identify C_n in time $\mathcal{O}(p(n))$ with the set $\{1,2,\dots,T\}$, by interpreting a valid configuration c as a natural number written in base $|\Sigma|$. Since M is deterministic, if it accepts on input x, it does so in less than T computation steps. So, without loss of generality, we can also assume that there is a unique accepting configuration $c_A \in C_n$, and that, when entering c_A, the machine M loops forever. Then, on any input of size n, M reaches either c_A after T computation steps or does never reach c_A. To an input $x \in \{0,1\}^n$, we also assign an initial configuration $c(x) \in C_n$. Finally, denote by c_R one arbitrarily chosen configuration in $C_n \setminus c_A$, which will be denoted as "rejecting".

Consider now the undirected graph $G_n = (V_n, E_n)$, where

$$V_n = C_n \times \{0,1,\dots,T\} \cup \{(c_A, T+1), (c_R, T+1)\}$$

and, for $t < T$,

$$\begin{aligned}
\{(c,t),(c',t+1)\} \in E_n \quad &\text{iff} \quad c' \text{ is the next configuration of } M \text{ from } c \\
\{(c,T),(c_A,T+1)\} \in E_n \quad &\text{iff} \quad c = c_A \\
\{(c,T),(c_R,T+1)\} \in E_n \quad &\text{iff} \quad c \neq c_A
\end{aligned}$$

Clearly, $x \in L$ if and only if there exists a path from $(c(x), 0)$ to $(c_A, T+1)$ in G_n. We claim that the graph G_n satisfies the following properties:

(i) G_n can be succintly described by a circuit of size polynomial in n.

(ii) G_n is a forest with two trees, which can be rooted at the vertices $(c_A, T+1)$ and $(c_R, T+1)$.

(iii) G_n can be embedded in $\mathbb{R}^3$ as a semilinear set decided by an additive circuit of size polynomial in n.

Indeed, (i) follows from the following: given (c,t) and (c',t') in V_n, one can check in time polynomial in n whether there is an edge between these two nodes in G_n. Also, $c(x)$, c_A and c_R can be described by a circuit of size polynomial in n.

We now prove (ii). Since M is deterministic, each configuration $c_i \in C_n$ has a unique next configuration $c_j \in C_n$. Therefore, each vertex (c_i, t), $t < T$ is connected to a unique vertex $(c_j, t+1)$. This implies that every connected component of G_n is a tree. Since for $t < T$ each configuration c_i leads either to c_A or to some $c_j \neq c_A$ in t steps, each vertex $(c_i, T-t)$, is connected either to (c_A, T) or to (c_j, T), hence either to $(c_A, T+1)$ or to $(c_R, T+1)$. Since $(c_A, T+1)$ and $(c_R, T+1)$ are not connected, this implies that there are two connected components in G_n. See Figure 2.1 for an illustration. For the rest of this proof, the two vertices $(c_A, T+1)$ and $(c_R, T+1)$ will be denoted as the roots of these two connected components.

We finally prove (iii). Since G_n is a forest, it is planar. However, the size of G_n is exponential in n, which makes it difficult to describe succintly a planar embedding of G_n. Instead, we will use the following property: as above, denote $(c_A, T+1)$ and $(c_R, T+1)$ as the roots of the two connected components of G_n. Define the layer t of G_n to be the set of vertices at a distance $T+1-t$ to one of the two roots. Clearly, the layer t is the set $\{(c,t) : c \in C_n\}$, thus one can decide in polynomial time whether a given vertex belongs to a given layer. It follows from (ii) that its edges link only vertices of two consecutive layers. let us focus first on the geometrical representation in $\mathbb{R}^3$ of two consecutive layers, corresponding to the vertices (c,t) and $(c', t+1)$ for any $c, c' \in C_n$.

Consider the set $\mathcal{S}_n(t) \subseteq \mathbb{R}^3$ of Figure 2.2, representing the layers of G_n with vertices (c,t) and $(c', t+1)$, and defined as follows. For t even, the vertices (c,t) are represented by the points $(c, t, 0) \in \mathbb{R}^3$, modulo the polynomial time identification between C_n and

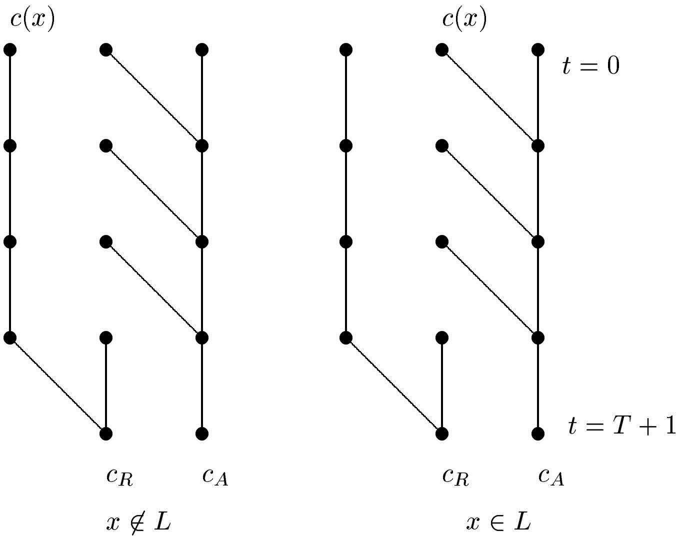

Figure 2.1: A Graph G_n, with $T = 3$, when $x \notin L$ and when $x \in L$.

$\{1, 2, \ldots, T\}$. The vertices $(c', t+1)$ are represented by the points $(0, t+1, c') \in \mathbb{R}^3$. When (c, t) and $(c', t+1)$ are linked by an edge in G_n, their corresponding representations in $\mathcal{S}_n$ are linked by a line segment. The same idea applies for t odd, with vertices (c, t) represented by the points $(0, t, c) \in \mathbb{R}^3$ and vertices $(c, t+1)$ represented by the points $(c, t+1, 0) \in \mathbb{R}^3$.

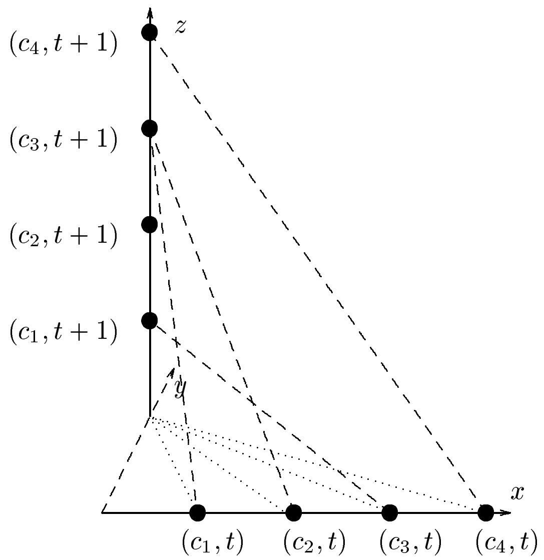

Figure 2.2: Tridimensional Representation of two Layers of G_n, with $T = 4$ and t even.

It is easy to check that the tridimensional representations of two different edges do not intersect. Indeed, by (ii), two different edges have at least two different incident vertices

(c,t) and (c',t), and, therefore, for t even their orthogonal projections on the plane $z=0$ do not intersect in the open interval $]t,t+1[$, so don't they. The same argument applies for t odd with the orthogonal projections on the plane $x=0$. Define now the set $\mathcal{S}_n \subseteq \mathbb{R}^3$ as the union of the $\mathcal{S}_n(t)$ for $t \in \{0,1,\ldots,T\}$. Clearly, $\mathcal{S}_n$ is a tridimensional representation of G_n.

Let us finally describe an algorithm deciding $\mathcal{S}_n$.

```
input (x, y, z) ∈ ℝ³
compute t = min{t' ∈ {0, 1, ..., T + 1} :   t' > y}
if t is even then
# case t is even
  if y = t + 1 = T + 1 ∧ z ∈ {c_A, c_R} ∧ x = 0 then HALT and ACCEPT
  else
  if y = t ∧ x ∈ {0, 1, ..., T} ∧ z = 0 then HALT and ACCEPT
    # (x, y, z) represents a vertex of G_n
  else
    compute c_t = min{c ∈ {0, 1, ..., T} :   c(y − t − 1) + x ≥ 0}
    compute c_{t+1} = min{c ∈ {0, 1, ..., T} :   c(y − t) − z ≥ 0}
    if {(c_t, t), (c_{t+1}, t + 1)} ∈ E_n ∧ c(y − t − 1) + x = 0 ∧ c(y − t) − z = 0
    then HALT and ACCEPT
      # (x, y, z) lies on the representation of an edge of G_n
    else HALT and REJECT
else
# case t is odd
  if y = t + 1 = T + 1 ∧ x ∈ {c_A, c_R} ∧ z = 0 then HALT and ACCEPT
  else
  if y = t ∧ z ∈ {0, 1, ..., T} ∧ x = 0 then HALT and ACCEPT
    # (x, y, z) represents a vertex of G_n
  else
    compute c_t = min{c ∈ {0, 1, ..., T} :   c(y − t − 1) + z ≥ 0}
    compute c_{t+1} = min{c ∈ {0, 1, ..., T} :   c(y − t) − x ≥ 0}
    if {(c_t, t), (c_{t+1}, t + 1)} ∈ E_n ∧ c(y − t − 1) + z = 0 ∧ c(y − t) − x = 0
    then HALT and ACCEPT,
      # (x, y, z) lies on the representation of an edge of G_n
    else HALT and REJECT.
```

It is clear from its definition that T is exponentially bounded, and thus is a natural number of bit-size polynomial in n. Therefore, the computation of t in the second line of the algorithm above can be performed by binary search in polynomial time. Indeed, assume $k \in \{0,1,\ldots,T+1\}$ is given in binary: $k \in \{0,1\}^{p(n)}$. The numerical value of k in $\mathbb{R}$, that we may denote as $\mathsf{val}(k) \in \mathbb{R}$, can be computed in time polynomial in n using a fast exponentiation technique, with only additions. Thus, a test like $x = \mathsf{val}(k)$, or $x < \mathsf{val}(k)$, can be performed in polynomial time. It is therefore enough to perform a binary search on the binary encoding of the elements of $\{0,1,\ldots,T+1\}$ to compute t.

The tests $x \in \{0, 1, \ldots, T\}$ and $z \in \{0, 1, \ldots, T\}$ can also be performed in polynomial time with binary search, as well as all equalities and inequalities, such as $c(y-t-1)+x \geq 0$. Thus, the computations of c_t and c_{t+1} can also be performed by binary search in polynomial time. Finally, since G_n is succintly described, the test $\{(c_t, t), (c_{t+1}, t+1)\} \in E_n$ can also be performed in polynomial time. So, it turns out that the whole algorithm works in polynomial time, and can therefore be carried out by an additive circuit $\mathscr{C}_n$ of polynomial size.

As above, the membership of x to L is equivalent to the reachability of the representations of $(c(x), 0)$ and of (c_A, T) in $\mathcal{S}_n$. The mapping $x \mapsto (\mathscr{C}_n, (c(x), 0), (c_A, T))$ therefore reduces L to $\text{REACH}_{\mathsf{add}}$.

□

Theorem 2.3.2 $\text{CONNECTED}_{\mathsf{add}}$ (Connectedness) *is the following problem: given an additive decision circuit $\mathscr{C}$ decide whether $S_{\mathscr{C}}$ is connected. The problem* $\text{CONNECTED}_{\mathsf{add}}$ *is* $\text{PAR}_{\mathsf{add}}$*-complete under Turing reductions. The same holds when restricted to problems in* $\mathbb{R}^3$.

PROOF. The membership follows from [BC04, Theorem 5.19] where it is proved that the computation of the 0th Betti number $b_0(S)$ is $\text{FPAR}_{\mathsf{add}}$-complete and the fact that S is connected if and only if $b_0(S) = 1$.

Let us focus on the hardness. The proof relies heavily on the proof of Lemma 2.3.9. Recall the graph G_n of Lemma 2.3.9. Consider $H_n = (V_n, E_n \cup \{(c_R, T+1), (c(x), 0)\})$ derived from G_n by adding one edge between $(c(x), 0)$ and $(c_R, T+1)$, as in figure 2.3. Then, H_n is connected if and only if x belongs to L. Indeed, x belongs to L if and only if $(c(x), 0)$ and $(c_A, T+1)$ are connected in G_n, if and only if they are connected in H_n. Since the other connected component of G_n is connected to $(c(x), 0)$ in H_n, this holds if and only if H_n is connected.

Now, from the semilinear set $\mathcal{S}_n \subseteq \mathbb{R}^3$ of Lemma 2.3.9 representing G_n, it is easy to build a semilinear set representing H_n: Consider $\mathcal{S}'_n$ containing $\mathcal{S}_n$ and the lines

$$\begin{array}{lll} x = 0, & y = 0 & \\ z = 0, & y = -1 & \\ x = 0, & y = T+2 & \text{if } T \text{ is even} \\ z = 0, & y = T+2 & \text{if } T \text{ is odd,} \end{array}$$

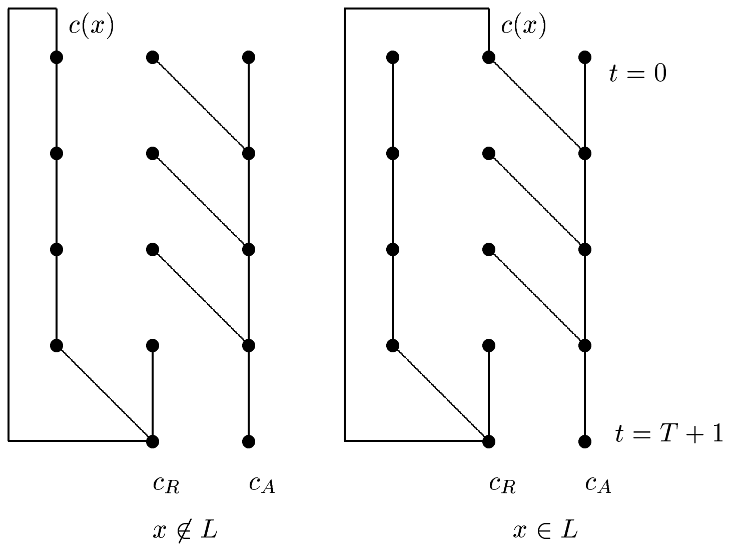

Figure 2.3: A Graph H_n, with $T = 3$, when $x \notin L$ and when $x \in L$.

as well as the segments

$$\begin{array}{llll} z = c_R, & y \in [T+1, T+2], & x = 0 & \text{if } T \text{ is even} \\ x = c_R, & y \in [T+1, T+2], & z = 0 & \text{if } T \text{ is odd} \\ x = c(x), & y \in [-1, 0], & z = 0 & \end{array}$$

Clearly, $\mathcal{S}'_n$ is decided by an additive circuit of size polynomial in n, and represents H_n.

□

2.4 Properties for the Zariski Topology

In this section we study the complexity of several topological properties of semilinear sets, when the topology at hand is the Zariski topology introduced above.

2.4.1 Zariski Adherence

Lemma 2.4.1 ZADH$_{\mathsf{add}}$ (Zariski Adherence) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates and a point $x \in \mathbb{R}^n$ decide whether x belongs to the Zariski closure of $S_{\mathscr{C}}$.The problem* ZADH$_{\mathsf{add}}$ *is* NP$_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Let us prove the membership. Denote by $\mathcal{A}$ the set of accepting paths of $\mathscr{C}$.

$$\overline{S_{\mathscr{C}}}^Z = \overline{\bigcup_{\gamma \in \mathcal{A}} D_\gamma}^Z = \bigcup_{\gamma \in \mathcal{A}} \overline{D_\gamma}^Z.$$

It follows that x is in $\overline{S_{\mathscr{C}}}^{Z}$ if and only if there exists an accepting path γ of S such that x is in $\overline{D_\gamma}^{Z}$.

Consider now $\gamma \in \mathcal{A}$ such that $D_\gamma \neq \emptyset$. Note that, given x, $\mathscr{C}$, and γ the values $F_i(x)$ and $F_j(x)$, for $i \in I_\gamma$ and $j \in E_\gamma$, can be computed in polynomial time. Also, we can test in non-deterministic polynomial time whether $D_\gamma = \emptyset$ by guessing a $y \in \mathbb{R}^n$ and checking whether $y \in D_\gamma$.

The following $\mathrm{NP}_{\mathsf{add}}$ algorithm therefore solves $\mathrm{ZADH}_{\mathsf{add}}$,

input $(\mathscr{C}, x)$
guess a path $\gamma \in \{-1, 0, 1\}^s$
check that $\gamma \in \mathcal{A}$
guess $y \in \mathbb{R}^n$
if $y \notin D_\gamma$ then REJECT
if $F_j(x) = 0$ for all $j \in E_\gamma$ then ACCEPT
else REJECT

For the hardness, the many-one reduction used in Lemma 2.2.1 carries on without modification. It reduces $\mathrm{CBS}_{\mathsf{add}}$ to $\mathrm{ZADH}_{\mathsf{add}}$.

□

2.4.2 Zariski Denseness

Lemma 2.4.2 $\mathrm{ZDENSE}_{\mathsf{add}}$ (Zariski Denseness) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates decide whether $S_{\mathscr{C}}$ is Zariski dense in $\mathbb{R}^n$. The problem* $\mathrm{ZDENSE}_{\mathsf{add}}$ *is* $\mathrm{NP}_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. For the membership, as in the proof of Lemma 2.4.1,

$$\overline{S}^{Z}_{\mathscr{C}} = \overline{\bigcup_{\gamma\in\mathcal{A}} D_\gamma}^{Z} = \bigcup_{\gamma\in\mathcal{A}} \overline{D_\gamma}^{Z}.$$

Therefore, $\overline{S}^{Z}_{\mathscr{C}} = \mathbb{R}^n$ if and only if there exists $\gamma \in \mathcal{A}$ such that D_γ is dense in $\mathbb{R}^n$. This leaf set is convex, and its Zariski closure is affine. It follows that it is dense if and only if its dimension is n. Hence, S is dense if and only if its dimension is n. The membership to $\mathrm{NP}_{\mathsf{add}}$ follows from the fact that $\mathrm{DIM}_{\mathsf{add}}$ is in $\mathrm{NP}_{\mathsf{add}}$ [BC04, Theorem 5.1].

The hardness follows, again, from the reduction used in Lemma 2.2.1 which reduces $\mathrm{CBS}_{\mathsf{add}}$ to $\mathrm{ZDENSE}_{\mathsf{add}}$.

□

2.4.3 Zariski Closedness

Lemma 2.4.3 ZCLOSED$_{\mathsf{add}}$ (Zariski Closed) *is the following problem: given an additive decision circuit $\mathscr{C}$ decide whether $S_{\mathscr{C}}$ is closed under the Zariski topology. The problem* ZCLOSED$_{\mathsf{add}}$ *is* coNP$_{\mathsf{add}}$*-complete under many-one reductions.*

PROOF. Given an additive circuit $\mathscr{C}$ with n input gates, denote by $\mathcal{A}$ its accepting paths. Then, as in the proof of Lemma 2.2.2, $S_{\mathscr{C}}$ is closed if and only if

$$\begin{aligned} & \forall x \in \mathbb{R}^n \quad x \in \overline{S_{\mathscr{C}}}^Z \Rightarrow x \in S_{\mathscr{C}} \\ \Leftrightarrow \quad & \forall x \in \mathbb{R}^n \quad \left(\exists \gamma \in \mathcal{A}\; x \in \overline{D_\gamma}^Z\right) \Rightarrow x \in S_{\mathscr{C}} \\ \Leftrightarrow \quad & \forall x \in \mathbb{R}^n \quad \left(\forall \gamma \in \mathcal{A}\; x \notin \overline{D_\gamma}^Z\right) \vee x \in S_{\mathscr{C}} \end{aligned}$$

The predicate $x \notin \overline{D_\gamma}^Z$ can be checked in coNP$_{\mathsf{add}}$: denote by E_γ the set of equalities of γ, then

$$x \notin \overline{D_\gamma}^Z \Leftrightarrow (D_\gamma = \emptyset \vee x \notin S(E_\gamma))\,.$$

$D_\gamma = \emptyset$ is clearly in coNP$_{\mathsf{add}}$, $x \notin \mathcal{S}(E_\gamma)$ in P$_{\mathsf{add}}$. This shows the membership.

The hardness follows, yet once more, from the reduction used in Lemma 2.2.1.

□

2.5 Irreducibility

Definition 2.5.1 $\Delta^{2,\|}_{\mathsf{add}}$ is the class of problems decidable by a polynomial time additive machine which asks *non adaptively* a polynomial number of queries in NP$_{\mathsf{add}}$. This means that the input to any query does not depend on the oracle answer to previous queries, but only on the input of the machine.

Some completeness results in the discrete version $\Delta^{2,\|}$ of this class exist. See [Pap83, Kre86] for examples. To the best of our knowledge, no such result over $\mathbb{R}_{ovs}$ has been proved so far.

2.5.1 Deciding Irreducibility

Lemma 2.5.1 IRR$_{\mathsf{add}}$ (Irreducibility) *is the following problem: given an additive decision circuit $\mathscr{C}$ with n input gates decide whether $S_{\mathscr{C}}$ is irreducible. The problem* IRR$_{\mathsf{add}}$ *is in* $\Delta^{2,\|}_{\mathsf{add}}$.

PROOF. Consider the following algorithm:

input $\mathscr{C}$ with n input gates
compute $d = \dim S_{\mathscr{C}}$ querying the oracle $\text{DIM}_{\mathsf{add}}$
check that for all accepting paths γ, γ' and all $x, x' \in \mathbb{R}^n$
$(x \in D_\gamma \wedge x' \in D_{\gamma'} \wedge \dim D_\gamma = \dim D_{\gamma'} = d) \Rightarrow \overline{D_\gamma}^Z = \overline{D_{\gamma'}}^Z$.

$\text{DIM}_{\mathsf{add}}$ is known to be in NP_{add} [BC04]. Therefore, $d = \dim S_{\mathscr{C}}$ can be computed with $n+1$ independent queries to NP_{add}: it suffices to perform independently the oracles queries $\dim D_{\gamma'} \geq k$ for $k = 0 \dots n$, and to choose the maximal answer. It is clear that $\text{NP}_{\mathsf{add}} \subseteq \Delta^{2,\|}_{\mathsf{add}}$.

The last line can be checked in deterministic polynomial time by Lemma 2.1.1, thus the two last lines belong to $\text{coNP}_{\mathsf{add}} \subseteq \Delta^{2,\|}_{\mathsf{add}}$.

All queries can be performed independently: it suffices to check in parallel $\dim D_{\gamma'} \geq k$ for $k = 0 \dots n$, to check in parallel the two last lines for $d = 0, \dots, n$, and to combine the answers with d the dimension of $S_{\mathscr{C}}$.

□

Lemma 2.5.2 *Let $S_1 \subseteq \mathbb{R}^n$ and $S_2 \subseteq \mathbb{R}^m$ be two non-empty semilinear sets. Then, $S_1 \times S_2 \subseteq \mathbb{R}^{n+m}$ is irreducible if and only if both S_1 and S_2 are irreducible.*

□

Lemma 2.5.3 *The problem $\text{IRR}_{\mathsf{add}}$ is $\Delta^{2,\|}_{\mathsf{add}}$-hard under many-one reductions.*

PROOF. Assume L is a problem in $\Delta^{2,\|}_{\mathsf{add}}$, and denote by L_n the set $L \cap \mathbb{R}^n$. Then, there exist a polynomial size circuit $\mathscr{C}^n$ with $n + p(n)$ input gates, and a family of polynomial size circuits $\mathscr{C}^n_1, \dots, \mathscr{C}^n_{p(n)}$ with n input gates and $q(n)$ output gates, where p and q are polynomials, such that, for $x \in \mathbb{R}^n$, x is in L_n if and only if

$$\text{for } i = 1, \dots, p(n), \quad s_i = \begin{cases} 1 & \text{if } \mathscr{C}^n_i(x) \in \text{ZDENSE}_{\mathsf{add}} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathscr{C}^n(x, s_1, \dots, s_{p(n)}) = 1.$$

Basically, the circuits $\mathscr{C}^n_i$ compute the inputs to the oracle queries. Since $\text{ZDENSE}_{\mathsf{add}}$ is NP_{add}-complete under many-one reductions, we can assume that these oracle queries are queries to this $\text{ZDENSE}_{\mathsf{add}}$ problem. The sequence $s_1, \dots, s_{p(n)}$ denotes the oracle answers, and $\mathscr{C}^n$ performs the final computation deciding the membership of x in L_n, given these oracle answers.

To be fully formal, we should add a P-uniformity condition on these circuits: there exists a Turing machine which, on input $(n, 0, j) \in \mathbb{N}^3$, computes a description of the j^{th} gate of

$\mathscr{C}^n$, and on input $(n,i,j) \in \mathbb{N}^3$, computes a description of the j^{th} gate of $\mathscr{C}_i^n$. This machine works in polynomial time.

We shall see now how to reduce this to the $\text{IRR}_{\mathsf{add}}$ problem. First, note that the output of the $\mathscr{C}_i^n$ circuits on x are inputs to the $\text{ZDENSE}_{\mathsf{add}}$ problem. As such, they are descriptions of additive circuits defining semilinear sets. We can assume that these $\mathscr{C}_i^n(x)$ have all the same number of input gates, say $r(n)$, where r is a polynomial. Denote by S_i^x the semilinear set defined by $\mathscr{C}_i^n(x)$. A second assumption that we can make without loss of generality is that, for all i, S_i^x is either empty or Zariski dense in $\mathbb{R}^{r(n)}$. Indeed, the reduction used in Lemma 2.2.1 produces semilinear sets with this property. Also, by hypothesis, the number of selection gates of the $\mathscr{C}_i^n(x)$ is at most $q(n)$.

Denote by $\mathcal{A}_i$ the set of accepting paths of $\mathscr{C}_i^n(x)$, and, if $\gamma \in \mathcal{A}_i$, denote by $D_{\gamma,i} \subseteq \mathcal{S}_i^x$ the corresponding leaf set.

Given i, γ, denote by $\partial(D_{\gamma,i})$ the Euclidean boundary of $D_{\gamma,i}$. Without loss of generality, we can assume that, for all i such that S_i^x is non-empty,

$$\bigcup_{\gamma \in \mathcal{A}_i} \partial(D_{\gamma,i}) \text{ is reducible.}$$

Indeed, the accepting leaf sets of the circuits produced by the reduction of Lemma 2.2.1 are quadrants of $\mathbb{R}^{r(n)}$, and therefore they satisfy this property.

We finally note that if $D_{\gamma,i}$ is not empty, for $y \in \mathbb{R}^{r(n)}$, the predicate $y \in \partial(D_{\gamma,i})$ is clearly in $\mathrm{P}_{\mathsf{add}}$.

Let z be any point in $\mathbb{R}^{r(n)}$, which we fix in what follows. Consider the semilinear set Ω defined by the following algorithm:

input $u \in \mathbb{R}^{q(n)}, (y_0, \ldots, y_{p(n)}) \in \mathbb{R}^{r(n)(p(n)+1)}, (a_1, \ldots, a_{p(n)}) \in \mathbb{R}^{p(n)}$
$\gamma := (\mathsf{sign}(u_1), \ldots, \mathsf{sign}(u_{q(n)}))$
$s := (\mathsf{pos}(a_1), \ldots, \mathsf{pos}(a_{p(n)}))$
(I) case $(\forall i \in \{1, \ldots, p(n)\}\ y_i \in S_i^x \cup \{z\}$ and $\exists i \in \{1, \ldots, p(n)\}\ a_i = 0)$
HALT and ACCEPT
(II) case $(\mathscr{C}^n(x,s) = 1$ and $\forall i \in \{1, \ldots, p(n)\}\ [y_i \in S_i^x \cup \{z\}]$ and
$\exists j \in \{1, \ldots, p(n)\}\ [s_j = 0 \wedge \gamma \in \mathcal{A}_j \wedge y_0 \in D_{\gamma,j} \wedge y_j \in \partial(D_{\gamma,j})])$
HALT and ACCEPT
(III) case $(\mathscr{C}^n(x,s) = 1$ and $\forall i \in \{1, \ldots, p(n)\}\ [(y_i \in S_i^x \wedge s_i = 1) \vee (y_i = z \wedge s_i = 0)])$
HALT and ACCEPT
else REJECT.

Clearly, by the P-uniformity hypothesis, this algorithm works in polynomial time. We claim that Ω is irreducible if and only if x is in L_n. To prove it we introduce some notation. We define

$$\Omega_{\mathrm{I}} = \{(u, y, a) \text{ satisfying case (I)}\}$$

and similarly $\Omega_{\rm II}$ and $\Omega_{\rm III}$ so that $\Omega = \Omega_{\rm I} \cup \Omega_{\rm II} \cup \Omega_{\rm III}$ (but this union is not necessarily disjoint).

Assume x is in L_n. Then there exists a sequence α of oracle answers $\alpha_1, \ldots, \alpha_{p(n)} \in \{0,1\}^{p(n)}$ such that $\mathscr{C}^n(x,\alpha) = 1$ and, for all $i \le p(n)$, S_i^x is Zariski dense in $\mathbb{R}^{r(n)}$ if $\alpha_i = 1$ and empty otherwise. Let $W = \{(u,y,a) \in \Omega \mid \mathsf{pos}(a) = \alpha\}$. We next prove several claims.

(a) $\overline{W}^Z \subseteq \overline{\Omega_{\rm III}}^Z$. First note that $W \cap \Omega_{\rm II} = \emptyset$. Indeed, if $y_0 \in D_{\gamma,j}$ for some j then $D_{\gamma,j} \neq \emptyset$ and $\alpha_j = 1$, in contradiction with $s_j = 0$. Therefore,

$$(W \cap \{(u,y,a) : \ a_i \neq 0 \text{ for all } i\}) \subseteq \Omega_{\rm III}.$$

Define $W_1 = W \cap \{(u,y,a) : \ a_i \neq 0 \text{ for all } i\}$, and $W_2 = W \cap \{(u,y,a) : \exists j,\ a_j = 0\}$. Clearly, $W = W_1 \cup W_2$. Consider now $x \in W_2$, and denote by J the subset of $\{1, \ldots, p(n)\}$ satisfying:

$$x = (u,y,a), \text{ with } j \in J \Leftrightarrow a_j = 0.$$

Assume $\epsilon < 0$, and define $a_j^\epsilon = \epsilon$ for all $j \in J$, and $a_j^\epsilon = a_j$ for all $j \notin J$. Then, $x^\epsilon = (u, y, a_1^\epsilon, \ldots, a_{p(n)}^\epsilon) \in W_1$. This shows that x belongs to the Euclidean closure of W_1, and therefore to its Zariski closure. Thus, $W_2 \subseteq \overline{W_1}^Z$. It follows,

$$W_1 \subseteq \Omega_{\rm III} \Rightarrow \overline{W_1}^Z \subseteq \overline{\Omega_{\rm III}}^Z \Rightarrow \overline{W}^Z \subseteq \overline{\Omega_{\rm III}}^Z.$$

(b) $\Omega_{\rm III} \subseteq \overline{W}^Z$. If $(u,y,a) \in \Omega_{\rm III} \setminus W$ then there exists j such that $(y_j = z \wedge a_j < 0 \wedge \alpha_j = 1)$. Since for such j, S_j^x is dense in $\mathbb{R}^{r(n)}$ we have $(u,y,a) \in \overline{W}^Z$.

(c) *W is irreducible.* Define

$$Y_i^x = \begin{cases} S_i^x & \text{if } \alpha_i = 1 \\ z & \text{if } \alpha_i = 0. \end{cases}$$

Then, $\overline{W}^Z = \mathbb{R}^{q(n)} \times \mathbb{R}^{r(n)} \times \overline{Y_1^x}^Z \times \ldots \times \overline{Y_{p(n)}^x}^Z \times \overline{\mathsf{pos}^{-1}(\alpha)}^Z$. Since every S_i^x is irreducible, so are the Y_i^x. Also, $\overline{\mathsf{pos}^{-1}(\alpha)}^Z = \mathbb{R}^{p(n)}$ and therefore, $\mathsf{pos}^{-1}(\alpha)$ is irreducible. Therefore, by Lemma 2.5.2, so is W.

(d) $\overline{\Omega_{\rm I}}^Z \subseteq \overline{\Omega_{\rm III}}^Z$. It is easy to check that $\overline{\Omega_{\rm I}}^Z \subseteq \overline{W}^Z$ and therefore $\overline{\Omega_{\rm I}}^Z \subseteq \overline{\Omega_{\rm III}}^Z$.

(e) $\overline{\Omega_{\rm II}}^Z \subseteq \overline{\Omega_{\rm III}}^Z$. The inclusion $\Omega_{\rm II} \subseteq \overline{\Omega_{\rm III}}^Z$ is immediate.

It now follows that Ω is irreducible since

$$\overline{\Omega}^Z = \overline{\Omega_{\rm I}}^Z \cup \overline{\Omega_{\rm II}}^Z \cup \overline{\Omega_{\rm III}}^Z = \overline{\Omega_{\rm III}}^Z = \overline{W}^Z.$$

Assume now that x is not in L_n. We will look at the sets $\Omega_{\rm II}$ and $\Omega_{\rm III}$. Two cases arise:

- There is no sequence $s = (s_1, \ldots, s_{p(n)})$ such that $\mathscr{C}^n(x,s) = 1$. In that case $\Omega_{\mathrm{II}} = \Omega_{\mathrm{III}} = \emptyset$.

- There exist sequences $s = (s_1, \ldots, s_{p(n)})$ such that $\mathscr{C}(x,s) = 1$. Since $\mathscr{C}(x,\alpha) = 0$, we have $s \neq \alpha$ for all of these sequences. For every such s we define

$$\Omega_{\mathrm{II}}(s) = \Omega_{\mathrm{II}} \cap (\mathbb{R}^{q(n)} \times \mathbb{R}^{r(n)(p(n)+1)} \times \mathsf{pos}^{-1}(s))$$

and similarly $\Omega_{\mathrm{III}}(s)$.

Again, two cases are possible:

(i) $\exists i\,(\alpha_i = 0 \wedge s_i = 1)$. If $(u,y,a) \in \Omega_{\mathrm{II}}(s)$ then $y_i \in \partial(D_{\gamma,j}) \subseteq S_i^x$. But, since $\alpha_i = 0$, $S_i^x = \emptyset$. It then follows that $\Omega_{\mathrm{II}}(s) = \emptyset$. Similarly, we prove that $\Omega_{\mathrm{III}}(s) = \emptyset$ since, if $(u,y,a) \in \Omega_{\mathrm{III}}(s)$, then $y_i \in S_i^x$.

(ii) $\exists j\,(\alpha_j = 1 \wedge s_j = 0)$. We may assume that (i) does not hold for any i. There may be several such j's, each of them corresponding to a subset of $\Omega_{\mathrm{II}}(s)$ along the second line of (II). For such a j, define

$$\begin{aligned}\Omega_{\mathrm{II}}^j(s) &= \{(u, y_0, \ldots, y_{p(n)}, a) \in \Omega_{\mathrm{II}}(s) : \\ &\quad [s_j = 0 \wedge \gamma \in \mathcal{A}_j \wedge y_0 \in D_{\gamma,j} \wedge y_j \in \partial(D_{\gamma,j})]\}.\end{aligned}$$

Denote by $J(s)$ the set of j's such that $(\alpha_j = 1 \wedge s_j = 0)$: clearly,

$$\Omega_{\mathrm{II}}(s) = \bigcup_{j \in J(s)} \Omega_{\mathrm{II}}^j(s).$$

For $(u, y_0, \ldots, y_{p(n)}, a) \in \Omega_{\mathrm{II}}^j(s)$ we have $y_j \in \bigcup_{\gamma \in \mathcal{A}_j} \partial(D_{\gamma,j})$. We also have $(u, y_0, \ldots, y_{j-1}, z, y_{j+1}, \ldots, y_{p(n)}, a) \in \Omega_{\mathrm{III}}(s)$. For a given affine subspace $\mathcal{E}$ of $\mathbb{R}^{q(n)+r(n)(p(n)+1)+p(n)}$ and a subset A of $\mathbb{R}^{q(n)+r(n)(p(n)+1)+p(n)}$, denote by $\Pi_{\mathcal{E}}(A)$ the orthogonal projection of A onto $\mathcal{E}$. It is easy to check that

$$\begin{aligned}&\Pi_{0\times\ldots\times 0\times\mathbb{R}^{r(n)}\times 0\times\ldots\times 0}\left(\left\{\Omega_{\mathrm{II}}^j(s) \cup \Omega_{\mathrm{III}}(s)\right\}\right) \\ = \;& 0 \times \ldots \times 0 \times \left(\{z\} \cup \bigcup_{\gamma \in \mathcal{A}_i} \partial(D_{\gamma,i})\right) \times 0 \times \ldots \times 0.\end{aligned}$$

Therefore, since linear maps preserve irreducibility and, by hypothesis, $\bigcup_{\gamma \in \mathcal{A}_i} \partial(D_{\gamma,i})$ is reducible, so is $\Omega_{\mathrm{II}}^j(s) \cup \Omega_{\mathrm{III}}(s)$. The same holds for

$$\Omega_{\mathrm{II}}(s) \cup \Omega_{\mathrm{III}}(s) = \bigcup_{j \in J(s)} \left\{\Omega_{\mathrm{II}}^j(s) \cup \Omega_{\mathrm{III}}(s)\right\}.$$

Since Ω_{I} is clearly non empty and reducible,

$$\Omega = \Omega_{\mathrm{I}} \cup \bigcup_{s \mid \mathscr{C}(x,s)=1} (\Omega_{\mathrm{II}}(s) \cup \Omega_{\mathrm{III}}(s)),$$

and all the sets in this union are either empty or reducible we conclude that Ω is reducible.

$\square$

Theorem 2.5.1 *The problem* $\mathrm{IRR}_{\mathsf{add}}$ *is* $\Delta^{2,\|}_{\mathsf{add}}$*-complete under many-one reductions.*

$\square$

Remark 2.5.1 One can define the class $\Delta^{2,[\log n]}_{\mathsf{add}}$ containing all problems decidable by a polynomial time additive machine which asks at most $\mathcal{O}(\log(n))$ queries in $\mathrm{NP}_{\mathsf{add}}$. Then, arguing as in [Pap94, Theorem 17.7] and [BH88], one can prove that $\Delta^{2,\|}_{\mathsf{add}} = \Delta^{2,[\log n]}_{\mathsf{add}}$.

2.5.2 Counting Irreducible Components

Theorem 2.5.2 $\#\mathrm{IRR}_{\mathsf{add}}$ (Counting Irreducible Components) *is the following problem: given an additive decision circuit* $\mathscr{C}$ *count the number of irreducible components of* $S_{\mathscr{C}}$*. The problem* $\#\mathrm{IRR}_{\mathsf{add}}$ *is* $\mathrm{FP}^{\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}}$*-complete under Turing reductions.*

PROOF. Let $S \subseteq \mathbb{R}^n$ be the set accepted by $\mathscr{C}$. Then,

$$\overline{S} = \overline{\bigcup_{\gamma \in \mathcal{A}} D_\gamma}^Z = \bigcup_{\gamma \in \mathcal{A}} \overline{D_\gamma}^Z.$$

It follows that the irreducible components of $\overline{S}$ (which are the closures of those of S and are in one-to-one correspondence with them) are some of the sets $\overline{D_\gamma}^Z$. Note, however, that some nonempty $\overline{D_\gamma}^Z$ may not be irreducible components since they are embedded in some higher dimensional components and, conversely, that some components may correspond to $\overline{D_\gamma}^Z$ for several γ's. A way to deal with these features is to associate to an irreducible component V of $\overline{S}$ the largest $\gamma \in \mathcal{A}$ such that $V = \overline{D_\gamma}^Z$. Let $\mathcal{I}_S$ be the set of paths $\gamma \in \mathcal{A}$ such that $\overline{D_\gamma}^Z$ is an irreducible component of S (i.e., it is nonempty and it is not embedded in a larger component of S) and γ is maximal in $\{\gamma' \in \mathcal{A} \mid \overline{D_{\gamma'}}^Z = \overline{D_\gamma}^Z\}$. Then, the association above bijects the irreducible components of S with the set $\mathcal{I}_S$. The following algorithm decides membership of γ to $\mathcal{I}_S$

input $(\mathscr{C}, \gamma)$
check that $\gamma \in \mathcal{A}$
check that $D_\gamma \neq \emptyset$
check that for all $\gamma' \in \mathcal{A}$ $(\overline{D_\gamma}^Z \subseteq \overline{D_{\gamma'}}^Z \Rightarrow \overline{D_\gamma}^Z = \overline{D_{\gamma'}}^Z)$
check that for all $\gamma' \in \mathcal{A}$ $(\overline{D_\gamma}^Z = \overline{D_{\gamma'}}^Z \Rightarrow \gamma' \leq \gamma))$

Note that the first line is decided in $\mathrm{P}_{\mathsf{add}}$, the second in $\mathrm{NP}_{\mathsf{add}}$, and the last two in $\mathrm{coNP}_{\mathsf{add}}$. Indeed, for any $\gamma \in \mathcal{A}$ with non-empty leaf set D_γ, $\overline{D_\gamma}^Z$ is the solution set $\mathcal{S}(E_\gamma)$ of the set E_γ of equalities corresponding to γ. For instance, the third line checks that

$$\forall \gamma' \in \mathcal{A}, \forall y \in D_{\gamma'} \ \dim(\mathcal{S}(E_\gamma \cup E_{\gamma'})) = \dim(\mathcal{S}(E_{\gamma'})) \Rightarrow \dim(\mathcal{S}(E_\gamma)) = \dim(\mathcal{S}(E_{\gamma'})),$$

and, again, this can be performed in polynomial time by Lemma 2.1.1.

Therefore, the whole algorithm is in Σ^2_{add} and it follows that $\#\mathrm{IRR}_{\mathsf{add}} \in \mathrm{D}\# \cdot \Sigma^2_{\mathsf{add}}$. Now use the additive version of Toda-Watanabe's Theorem 1.4.3 (2) to obtain $\#\mathrm{IRR}_{\mathsf{add}} \in \mathrm{FP}^{\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}}$.

We now prove the hardness. Note that the problem $\mathrm{CSAT}_{\mathsf{add}}$ trivially belongs to $\mathrm{P}^{\#\mathrm{IRR}_{\mathsf{add}}}_{\mathsf{add}}$. Indeed, for an additive decision circuit $\mathscr{C}$, $S_{\mathscr{C}} = \emptyset$ if and only if the number of irreducible components of $S_{\mathscr{C}}$ is zero. since $\mathrm{CSAT}_{\mathsf{add}}$ is $\mathrm{NP}_{\mathsf{add}}$-complete, $\mathrm{NP}_{\mathsf{add}} \subseteq \mathrm{P}^{\#\mathrm{IRR}_{\mathsf{add}}}_{\mathsf{add}}$. Therefore, by [BC04, Theorem 5.1], we have $\mathrm{DIM}_{\mathsf{add}} \in \mathrm{P}^{\#\mathrm{IRR}_{\mathsf{add}}}_{\mathsf{add}}$.

It is now easy to design a Turing reduction from $\#\mathrm{CSAT}_{\mathsf{add}}$ to $\#\mathrm{IRR}_{\mathsf{add}}$. On input an additive circuit $\mathscr{C}$, first decide whether $S_{\mathscr{C}}$ is finite using oracle calls to $\mathrm{DIM}_{\mathsf{add}}$, and hence to $\#\mathrm{IRR}_{\mathsf{add}}$. If no, return ∞, otherwise return the number of irreducible components of $S_{\mathscr{C}}$. Since $\#\mathrm{CSAT}_{\mathsf{add}}$ is $\#\mathrm{P}_{\mathsf{add}}$-complete [BC04, Theorem 3.6], this proves the hardness.

□

Lemma 2.5.4 $\#\mathrm{IRR}^{(d)}_{\mathsf{add}}$ (Counting Irreducible Components of Fixed Dimension) *is the following problem: given an additive decision circuit $\mathscr{C}$ count the number of irreducible components of $S_{\mathscr{C}}$ of dimension d.*
$\#\mathrm{IRR}^{[d]}_{\mathsf{add}}$ (Counting Irreducible Components of Fixed Codimension) *is the following problem: given an additive decision circuit $\mathscr{C}$ count the number of irreducible components of $S_{\mathscr{C}}$ of codimension d. For all $d \in \mathbb{N}$, the problems $\#\mathrm{IRR}^{(d)}_{\mathsf{add}}$ and $\#\mathrm{IRR}^{[d]}_{\mathsf{add}}$ are $\mathrm{FP}^{\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}}$-complete under Turing reductions.*

PROOF. The membership of both problems follows from the fact that deciding irreducibility is in Σ^2_{add}, and checking the dimension of a non-empty leaf set is in $\mathrm{P}_{\mathsf{add}}$.

The hardness of $\#\mathrm{IRR}^{(d)}_{\mathsf{add}}$ is given by the following reduction from $\#\mathsf{SAT}$ to $\#\mathrm{IRR}^{(d)}_{\mathsf{add}}$.

For $\ell = 1, \ldots, n-d$ and $s \in \{-1,1\}^n$ consider the linear function $L_{\ell,s}$

$$x \mapsto \sum_{j=\ell}^{n} s_j x_j - (n - \ell + 1)$$

and let H_s be the affine subspace of $\mathbb{R}^n$ defined by the equalities $\{L_{1,s} = 0, \ldots, L_{n-d,s} = 0\}$. By construction $s \in H_s$ and $\dim H_s = d$.

Assume ϕ is a first-order formula with n variables. Consider the circuit $\mathscr{C}$ doing the following, whose solution set is shown by Figure 2.4.

input $(x_1, \ldots, x_n)$
check that, $\forall i \leq n,\ x_i \neq 0$.
compute $s(x) = (\mathsf{sign}(x_1), \ldots, \mathsf{sign}(x_n))$.
check that, $\forall i \leq n, |x_i - \mathsf{sign}(x_i)| \leq \frac{1}{2}$.
check that $x \in H_{s(x)}$
check that $(\mathsf{pos}(x_1), \ldots, \mathsf{pos}(x_n)) \in \{0,1\}^n$ satisfies ϕ

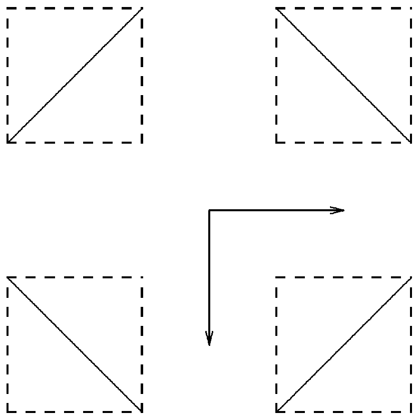

Figure 2.4: The Leaf Sets of $\mathscr{C}$ with $n = 2, d = 1$.

It is clear that each non-empty accepting leaf set is an irreducible component of dimension d. It is also clear that these leaf sets are in one-to-one correspondence with the truth assignments of ϕ. The reduction is clearly polynomial.

The hardness of $\#\mathrm{IRR}^{[d]}_{\mathsf{add}}$ is proved similarly.

$\square$

Lemma 2.5.5 $\#\mathrm{IRR}^{\{N\}}_{\mathsf{add}}$ (Counting Irreducible Components in Fixed Dimensional Space) *is the following problem: given an additive decision circuit with N input gates $\mathscr{C}$ count the number of irreducible components of $S_{\mathscr{C}}$.*
For all $N \in \mathbb{N}$, the problem $\#\mathrm{IRR}^{\{N\}}_{\mathsf{add}}$ is $\mathrm{FP}^{\#\mathrm{P}_{\mathsf{add}}}_{\mathsf{add}}$-complete under Turing reductions.

PROOF. The membership is immediate. Let us proof hardness for $N = 1$. The hardness is given by the following reduction from #SAT to $\#\mathrm{IRR}^{\{N\}}_{\mathsf{add}}$. Assume ϕ is a first-order formula with n variables. Consider the circuit $\mathscr{C}$ doing the following:

input $x \in \mathbb{R}$
check that $0 \leq x < 2^n$
check that $x \in \mathbb{N}$
compute the sequence $s(x) = (s_{n-1}, \ldots, s_0)$ of digits of x in binary
check that $s(x)$ is a satisfying assignment for ϕ

Note that the third line in the algorithm can be achieved in time polynomial in n by dichotomy search. The fourth line can also be computed in time polynomial in n. Therefore, an $\mathscr{C}$ as desired may be found with polynomial size.

It is clear that the non-empty leaf sets are irreducible, since they have dimension 0. It is also clear that they are in one-to-one correspondence with the satisfying assignments of ϕ. The reduction is clearly polynomial in n.

□

2.6 Our Completeness Results at a Glance

Problem	Class	Reduction	Reference Problem	Proof
$\mathrm{EADH}_{\mathsf{add}}$	$\mathrm{NP}_{\mathsf{add}}$	many-one	$\mathrm{CBS}_{\mathsf{add}}$	Lemma 2.2.1
$\mathrm{ZADH}_{\mathsf{add}}$	$\mathrm{NP}_{\mathsf{add}}$	many-one	$\mathrm{CBS}_{\mathsf{add}}$	Lemma 2.4.1
$\mathrm{ECLOSED}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{CBS}_{\mathsf{add}}$	Lemma 2.2.2
$\mathrm{ZCLOSED}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{CBS}_{\mathsf{add}}$	Lemma 2.4.3
$\mathrm{EDENSE}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{CBS}_{\mathsf{add}}$	Lemma 2.2.3
$\mathrm{ZDENSE}_{\mathsf{add}}$	$\mathrm{NP}_{\mathsf{add}}$	many-one	$\mathrm{CBS}_{\mathsf{add}}$	Lemma 2.4.2
$\mathrm{UNBOUNDED}_{\mathsf{add}}$	$\mathrm{NP}_{\mathsf{add}}$	many-one	$\mathrm{CSAT}_{\mathsf{add}}$	Lemma 2.3.1
$\mathrm{ISOLATED}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{EADH}_{\mathsf{add}}$	Lemma 2.3.2
$\mathrm{EXISTISO}_{\mathsf{add}}$	Σ^2_{add}	many-one	$\mathrm{ISOLATED}_{\mathsf{add}}$	Corollary 2.3.1
$\#\mathrm{ISO}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{SAT}$	Theorem 2.3.1
$\mathrm{LOCDIM}_{\mathsf{add}}$	$\mathrm{NP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{ISOLATED}_{\mathsf{add}}$	Lemma 2.3.3
$\mathrm{LOCCONT}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathbb{R}^* \setminus \mathrm{CBS}_{\mathsf{add}}$	Lemma 2.3.4
$\mathrm{CONT}_{\mathsf{add}}$	coNP	many-one	$\mathrm{LOCCONT}_{\mathsf{add}}$	Lemma 2.3.5
$\#\mathrm{DISC}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{CSAT}_{\mathsf{add}}$	Lemma 2.3.6
$\mathrm{SURJ}_{\mathsf{add}}$	Π^2_{add}	many-one	$\Pi^2\mathrm{CSAT}_{\mathsf{add}}$	Lemma 2.3.7
$\mathrm{COMPACT}_{\mathsf{add}}$	$\mathrm{coNP}_{\mathsf{add}}$	many-one	$\mathrm{ECLOSED}_{\mathsf{add}}$	Lemma 2.3.8
$\mathrm{REACH}_{\mathsf{add}}$	$\mathrm{PAR}_{\mathsf{add}}$	Turing		Lemma 2.3.9
$\mathrm{CONNECTED}_{\mathsf{add}}$	$\mathrm{PAR}_{\mathsf{add}}$	Turing	$\mathrm{REACH}_{\mathsf{add}}$	Theorem 2.3.2
$\mathrm{IRR}_{\mathsf{add}}$	Δ^2_{add}	Turing		Theorem 2.5.1
$\mathrm{IRR}_{\mathsf{add}}$	$\Delta^{2,\Vert}_{\mathsf{add}}$	many-one		Theorem 2.5.1
$\#\mathrm{IRR}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{CSAT}_{\mathsf{add}}$	Theorem 2.5.2
$\#\mathrm{IRR}^{(d)}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{SAT}$	Lemma 2.5.4
$\#\mathrm{IRR}^{[d]}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{SAT}$	Lemma 2.5.4
$\#\mathrm{IRR}^{\{N\}}_{\mathsf{add}}$	$\mathrm{FP}^{\#\mathrm{P}}_{\mathsf{add}}$	Turing	$\#\mathrm{SAT}$	Lemma 2.5.5

Figure 2.5: Completeness Results in the Additive Setting, with the Reference Problems used in the Proofs

Chapter 3

Classical Characterizations of P

In this chapter we recall briefly some classical machine independent characterization of deterministic polynomial time. We do not intend to provide a thorough landscape of all previous characterizations; we selected a few of them which we believe are representative of the major approaches, and help understand our results.

3.1 Logics on Finite Structures

We briefly recall some logical characterizations of complexity classes. They are based on a result by Fagin [Fag74], who characterized NP in terms of existential second-order logic over finite structures. Later on, Immerman [Imm83, Imm86] and Vardi [Var82] proposed some characterizations of P in terms of extensions of first-order logic, or restrictions of second-order logic, over finite structures. In Chapter 4 we propose an extension of these results to the setting of computations over arbitrary structures, based on the works by Grädel and Meer [GM95] and Cucker and Meer [CM99] over the real numbers, and of Grädel and Gurevich over arithmetical structures [GG98].

3.1.1 Definitions

Definition 3.1.1 (Logical Structures)

Let L be a vocabulary containing relation and function symbols, together with their (finite) arities, which is at least 1 for the relation symbols. A *logical structure* of signature L is a pair $\mathcal{U} = (\mathcal{A}, \mathcal{F})$, consisting of:

A a set, called the universe of $\mathcal{U}$.

$\mathcal{F}$ a set of functions $f : A^{k_i} \to A$ interpreting the function symbols of L, with the corresponding arities, and of relations $r \subseteq A^{k_i}$ interpreting the relation symbols of L, with the corresponding arities. That is, with each function symbol f_i of arity k_i in L there is an associated function $f_i^{\mathcal{U}} : A^{k_i} \to A$, and with each relation symbol r_i of arity k_i in L there is an associated relation $r_i^{\mathcal{U}} \subseteq A^{k_i}$. Note that interpretations of function symbols of arity 0 are elements in A.

A logical structure is finite when its universe A is finite. In what follows, we will only consider finite structures with finite vocabularies, containing a relation symbol $=$ interpreted as an equality over A.

Remark 3.1.1 Recall the notion of an arbitrary structure $\mathcal{K}$ given in Definition 1.1.1. Computational structures as described in Chapter 1 are instances of logical structures as above, with a possibly infinite vocabulary and a possibly infinite universe. We make however a distinction between computational and logical structures in the sense that we use computational structures as defined in Chapter 1 in order to describe the domain of possible values for the inputs of algorithms and machines, together with the operations and relations that the machines can perform in one step over these inputs, while finite logical structures as above are syntactical objects that can be used for encoding inputs to one particular Turing machine. Examples of computational structures are the binary structure $\mathbb{Z}_2$, for describing the domain of possible values for inputs to classical Turing machines, or $\mathbb{R}$ for inputs to numerical analysis algorithms. An example of finite logical structure for describing inputs to one particular algorithm is the following: consider a finite vocabulary containing only one binary relation symbol e. The class of finite logical structures of signature L is exactly the class of directed graphs. For any finite logical structure $\mathcal{U} = (\mathcal{A}, \mathcal{F})$ of signature L, the universe A is the set of vertices and $e^{\mathcal{U}} \in \mathcal{F}$ is the relation characterizing the edges of the graph. These structures provide a natural way for encoding graphs, considered as inputs to algorithms of graph-theoretical decision problems.

3.1.2 Results

It is clear that finite logical structures can be encoded in a natural way as elements in $(\mathbb{Z}_2)^*$, of size polynomial in the size of the alphabet, and vice versa. Our main concern is that it allows us to consider finite logical structures of a given signature as natural inputs for each algorithm or Turing machine. The example above gives a natural signature for the inputs to graph decision problems.

We assume that the reader is familiar with first-order and second-order logics on finite structures, see e.g. [dR03b, dR03a]. For an idea, see Chapter 4 where such logics are defined over a more general concept of $\mathcal{K}$-structures.

Modulo the natural encoding above, some characterizations of complexity classes in terms of logics on finite logical structures exist. In the following results, we assume that all finite structures considered are equipped with a binary successor relation.

A characterization in fixed point first-order logic:

Theorem 3.1.1 *[Var82, Imm86]*

Let S be a decision problem of finite structures. Then the following statements are equivalent.

(i) $S \in \mathrm{P}$.

(ii) *There exists a sentence ψ in fixed point first-order logic such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

Also, a characterization in Horn existential second-order logic:

Theorem 3.1.2 *[Var82, Imm86]*

Let S be a decision problem of finite structures. Then the following statements are equivalent.

(i) $S \in \mathrm{P}$.

(ii) *There exists a sentence ψ in Horn existential second-order logic such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

Fagin's characterization of NP is the following:

Theorem 3.1.3 *[Fag74]*

Let S be a decision problem of finite structures. Then the following statements are equivalent.

(i) $S \in \mathrm{NP}$.

(ii) *There exists an existential second-order sentence ψ such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

3.2 Interpreting Primitive Recursion over Finite Structures

Based on the notion of finite structures of Definition 3.1.1 above, the following characterization was given independently by Sazonov [Saz80] and Gurevich [Gur83]. The idea is to interpret primitive recursion over finite structures as above, equipped with a successor function.

Definition 3.2.1 (Global Relations and Functions)

A global l-ary relation R, $l \geq 0$, of vocabulary L, assigns to any finite structure $\mathcal{U}$ of signature L and universe A a relation:

$$\begin{array}{rll} R^{\mathcal{U}}: & A^l & \to \{\mathtt{false}, \mathtt{true}\} \\ & x & \to R^{\mathcal{U}}(x). \end{array}$$

Similarly, a global function F of arity $l \geq 0$ and co-arity $r > 0$, of vocabulary L, assigns to any finite structure $\mathcal{U}$ of signature L and universe A a function:

$$\begin{array}{rll} F^{\mathcal{U}}: & A^l & \to A^r \\ & x & \to F^{\mathcal{U}}(x). \end{array}$$

Clearly, global relations define decision problems of finite structures, as in the previous section. Similarly, global functions define computation problems of finite structures.

In the following, it is assumed that all finite structure are equipped with a successor function. For any $l > 1$, a l-ary successor function, obtained by the lexicographic order induced by the successor function, is also assumed. Assume also that the symbol 0 (respectively END) is interpreted as the minimal (resp. maximal) element of the universe of a structure. The following sets of global functions are then defined:

Definition 3.2.2 (Primitive Recursive and Recursive Global Functions)

A global function of vocabulary L is *primitive recursive* if it belongs to the closure of the constants, successor and projection functions under composition and primitive recursion.

Fix a signature L. Consider a system E of equations $t_1 = s_1, \ldots, t_k = s_k$ where the terms t_i, s_i are composed from the following symbols: 0, END, successor functions symbols, individual variables and function symbols. Such a system *recursively defines* a global function F in a finite structure $\mathcal{U}$ of signature L and universe A if for every $a \in A^l$, there is at most one $b \in A^r$ derivable from E and from valid equalities $g(c) = d$ in $\mathcal{U}$ by means of

(i) substitution of elements of A for variables, and,

(ii) replacement of a term $t(c)$ without variables by a vector d of elements of A provided the equation $t(c) = d$ has already been proved.

The system E recursively defines F if it recursively defines F in every finite structure $\mathcal{U}$ of signature L. A global function F is *recursive* if it is recursively defined by a system E.

Remark 3.2.1 Recursive definitions of global functions as above are actually very similar to the fixed point rule in first-order logic. It is interesting to note that they yield the same characterization.

Primitive recursive functions and recursive functions yield the two following characterizations:

Theorem 3.2.1 *[Saz80, Gur83] A global function is primitive recursive if and only if it is computable in logarithmic space.*

Theorem 3.2.2 *[Saz80, Gur83] A global function is recursive if and only if it is computable in polynomial time*

3.3 Cobham's Bounded Recursion on Notation

In the following, the calculus of recursive functions is interpreted over non-finite algebras of words or trees. The typical example is $\mathbb{N}$, considered as an algebra of binary words. In this setting, it is well known that primitive recursion allows an exponential number of computation steps. Consider the following example:

Example 3.3.1

$$\begin{array}{rcl} \mathsf{add}(0,y) & = & y \\ \mathsf{add}(\mathsf{suc}(x),y) & = & \mathsf{suc}(\mathsf{add}(x,y)) \\ \mathsf{exp}(0) & = & \mathsf{suc}(0) \\ \mathsf{exp}(\mathsf{suc}(n)) & = & \mathsf{add}(\mathsf{exp}(n),\mathsf{exp}(n)) \end{array}$$

Cobham's notion of bounded recursion on notation provides a way to rule out superpolynomial growth in recursion schemes.

Definition 3.3.1 (Bounded Recursion on Notation)
Consider the following set of *basic functions*:

1. $x\#y = 2^{|x|\cdot|y|}$.
2. $S_0(x) = 2x$, $S_1(x) = 2x + 1$
3. the constant 0

4. parity

5. projections

6. $x \to \lfloor x/2 \rfloor$

The class of Cobham's functions is the closure of the basic functions under composition and the following *bounded recursion on notation* scheme:

$$\begin{aligned} f(0, y_1, \ldots, y_k) &= g(y_1, \ldots, y_k) \\ f(x, y_1, \ldots, y_k) &= h(x, y_1, \ldots, y_k, f(\lfloor x/2 \rfloor, y_1, \ldots, y_k)), \\ & \quad \text{when } x \geq 0 \text{ and } f(x, y_1, \ldots, y_k) \leq k(x, y_1, \ldots, y_k), \end{aligned}$$

where g, h and k are already defined.

The main feature in this definition is the bound on the growth of f, by another function k of the system. It is the key that prevents superpolynomial growth.

Theorem 3.3.1 *[Cob62] The class of Cobham's functions is precisely* FP.

3.4 Data Tiering

Consider again the example above in Example 3.3.1. Another way to rule out superpolynomial growth in recursion schemes is the use of ramified data, or data tiering, introduced independently by Simmons [Sim88], Leivant [Lei90b, Lei94b] and Bellantoni and Cook [BC92]. It allows to avoid the use of such explicit bounds. One underlying idea is that the data objects are used computationally in different guises. By explicitly separating the uses, and requiring that the recursion schemes respect that separation, classes of recursive functions are obtained, which correspond closely to major complexity classes. We given below an idea of the approach used by Leivant [Lei90b, Lei94b].

Let $\mathbb{A}$ be the set of basic functions containing:

1. $\mathsf{suc}(x) = x + 1$

2. the constant 0

3. projections

Define $\mathcal{S}(\mathbb{A}) = \bigsqcup_{i=1}^{\infty} \mathbb{A}_i$, where the $\mathbb{A}_i$ are copies of $\mathbb{A}$. We denote these copies as *tiers* of the data. The set of ramified recursive functions is:

Definition 3.4.1 (Ramified Recursion)

The set of *ramified recursive functions* is the closure of $\mathcal{S}(\mathbb{A})$ under the application of the following ramified recursion scheme:

$$\begin{aligned} f(0, y_1, \ldots, y_k) &= g(y_1, \ldots, y_k) \\ f(\mathsf{suc}(x), y_1, \ldots, y_k) &= h(x, y_1, \ldots, y_k, f(x, y_1, \ldots, y_k)), \end{aligned}$$

provided x is in $\mathbb{A}_{k_x}$, the y_i are in some $\mathbb{A}_{k_{y_i}}$, $f(\mathsf{suc}(x), y_1, \ldots, y_k)$ is in $\mathbb{A}_{k_f}$, and $k_f < k_x, k_f \leq k_{y_1}, \ldots, k_f \leq k_{y_k}$, and the following ramified composition scheme:

$$f(x) = g(h_1(x), \ldots, h_k(x)),$$

provided the $h_i(x)$ are in some $\mathbb{A}_{k_i}$, and $g(h_1(x), \ldots, h_k(x))$ is in $\mathbb{A}_{k_g}$ with $k_g < k_i, \ldots, k_g < k_k$.

The main feature in this definition is that, in the recurrence scheme and in the composition scheme, the tier of the result is strictly smaller than the tier of the critical argument and of the parameters. This mechanism prevents a superpolynomial growth in recursion schemes:

Theorem 3.4.1 *[Lei94a] The set of ramified recursive functions is precisely the set of functions computed in polynomial time.*

Another approach due to Bellantoni and Cook, quite similar, can be found in [BC92]. We do not provide details here, since our Chapter 6 extends this other characterization to the setting of computation over an arbitrary structure $\mathcal{K}$. When $\mathcal{K}$ is specialized to $\mathbb{Z}_2$, our definitions coincide with the ones in [BC92].

Chapter 4

Metafinite Model Theory

This chapter is devoted to the exposition of some logical machine-independent characterizations of complexity classes over an arbitrary computational structure. It is based on the classical results by Fagin [Fag74], Immerman [Imm83, Imm86] and Vardi [Var82] over finite alphabets. In [GM95], the notion of $\mathbb{R}$-structure has been introduced, and characterization of deterministic polynomial time $P_{\mathbb{R}}$ and non-deterministic polynomial time $NP_{\mathbb{R}}$ in terms of logics over these $\mathbb{R}$-structures were provided. These results have been later on extended in [CM99], in order to capture other complexity classes, and in [GG98] over structures other than $\mathbb{R}$. Note however that, in the characterizations of [GM95, CM99], some logic is hidden in basic computations over the real numbers. In particular, these characterizations assume that the set of real numbers is equipped with a sign function and a multiplication: therefore they cannot be trivially extended, say, to the additive setting or to the field of complex numbers. Similar restrictions apply to the more general characterizations of [GG98], where it is required that the underlying computational structure contains an expansion of $(\mathbb{N}, 0, 1+, -, <, \max, \min, \Sigma, \Pi)$. Here again, this does not apply to the field of complex number, or to finite structures. In this chapter, we extend the notions of $\mathbb{R}$-structures to $\mathcal{K}$-structures over an arbitrary computational structure $\mathcal{K}$, extend the notion of first order logic and second order logic over $\mathbb{R}$-structures of [GM95] to first and second-order logic over $\mathcal{K}$-structures. We also define a proper notion of fixed-point rule, different from [GM95], allowing to capture $P_{\mathcal{K}}$, and propose a characterization of $NP_{\mathcal{K}}$. Our characterizations, while only slightly different from the ones of [GG98], apply to any computational structure as defined in Chapter 1. In particular, when $\mathcal{K}$ is $\mathbb{Z}_2$, they coincide with the classical one exposed in the previous chapter. The plan of our exposition follows closely the one in [BCSS98] for $\mathbb{R}$-structures.

4.1 First-Order Logic on $\mathcal{K}$-structures

4.1.1 Definitions

We assume that the reader is familiar with first-order mathematical logic. In order to capture computation over arbitrary structure, we extend the notion of $\mathbb{R}$-Structures from [GM95, BCSS98] to the notion of $\mathcal{K}$-Structures. Our $\mathcal{K}$-Structures are particular instances of the metafinite structures of Grädel and Gurevich [GG98]. They are based on the notion of logical structure of Definition 3.1.1. Recall Remark 3.1.1 for a distinction between an arbitrary computational structure $\mathcal{K}$ and finite logical structures.

In what follows, $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1 \ldots, rel_l, \mathbf{0}, \mathbf{1})$ is the computational structure at hand, as defined in Chapter 1. It is fixed for the rest of this chapter.

Definition 4.1.1 **($\mathcal{K}$-Structures)**
Let L_s, L_f be finite vocabularies, where L_s may contain relation and function symbols with their arities, and L_f contains function symbols only, with their arities. An $\mathcal{K}$-structure of signature $\sigma = (L_s, L_f)$ is a pair $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ consisting of

(i) a finite logical structure $\mathcal{U}$ of vocabulary L_s, called the *skeleton* of $\mathcal{D}$, whose (finite) universe A is also called the *universe* of $\mathcal{D}$, and

(ii) a finite set $\mathcal{F}$ of functions $f_i^{\mathcal{D}} : A^{k_i} \to \mathbb{K}$ interpreting the function symbols f_i in L_f, with the corresponding arities k_i.

$\mathcal{K}$-Structures as above are a generalization of the concept of logical structure, over a fixed computational structure $\mathcal{K}$, in order to denote inputs to algorithms performed by BSS machines over $\mathcal{K}$. The skeleton of a $\mathcal{K}$-structure is used for describing the finite, discrete part of a structure, while the set $\mathcal{F}$ is used for describing its computational part.

Example 4.1.1 As for the finite logical structures above for classical complexity, $\mathcal{K}$-structures provide a natural way of encoding inputs to BSS machines over $\mathcal{K}$. Consider for instance the real case $\mathbb{R}$, and the $\mathrm{NP}_{\mathbb{R}}$-complete problem 4FEAS: decide whether a given polynomial of degree 4 has a real zero. Inputs to this decision problem are polynomials of degree 4, which can be encoded by $\mathbb{R}$-structures of signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ as follows: The universe A contains one variable for each variable of the polynomial. The interpretation of the function $f_4 : A^4 \to \mathbb{R}$ of L_f gives the (real) coefficient for each monomial of degree 4, the interpretation of the function $f_3 : A^3 \to \mathbb{R}$ of L_f gives the coefficient for each monomial of degree 3, an so on for degrees 2, 1 and zero (the constant of the polynomial). We will see later on how to state the 4FEAS problem in a logical way over such structures.

A more formal description of the relations between $\mathcal{K}$-structures and inputs to BSS machines over $\mathcal{K}$ is given below in Definition 4.1.7. We present now the notion of first-order logic over $\mathcal{K}$-structures. A key feature of this logic is that the quantified variable range only over the skeleton of the structures, and not over $\mathcal{K}$.

Assume now that V is a fixed, countable set of variables.

Definition 4.1.2 (First-Order Logic for $\mathcal{K}$-Structures)

The language of *first-order logic* for $\mathcal{K}$-structures, $\mathrm{FO}_{\mathcal{K}}$, contains for each signature $\sigma = (L_s, L_f)$ a set of terms and formulas. We first define terms, of which there are two kinds. When interpreted in a $\mathcal{K}$-structure $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ with universe A, each term t takes values either in A, in which case we call it an *index term*, or in $\mathbb{K}$, in which case we call it a *number term*. Terms are defined inductively as follows.

(i) The set of index terms is the closure of the set V of variables under the application of functions symbols of L_s.

(ii) If $h_1, \ldots, h_k$ are index terms, and X is a k-ary function symbol of L_f, then $X(h_1, \ldots, h_k)$ is a number term.

(iii) If $t_1, \ldots, t_{k_i}$ are number terms, and op_i is an operation of the computational structure $\mathcal{K}$ of arity k_i, $op_i(t_1, \ldots, t_{k_i})$ is also a number term. In particular, operations of arity 0 of $\mathcal{K}$ yield constant number terms.

Atomic formulas are defined as follows.

(i) equalities $h_1 = h_2$ of index terms are atomic formulas.

(ii) If $t_1, \ldots, t_{k_i}$ are number terms, and rel_i is a relation of the computational structure $\mathcal{K}$ of arity k_i, $rel_i(t_1, \ldots, t_{k_i})$ is an atomic formula. In particular we may consider equalities $t_1 = t_2$ of number terms.

(iii) If $h_1, \ldots, h_k$ are index terms, and P is a k-ary relation symbol in L_s, $P(h_1, \ldots, h_k)$ is also an atomic formula.

The set of *formulas* of $\mathrm{FO}_{\mathcal{K}}$ is the smallest set containing all atomic formulas and which is closed under Boolean connectives, $\vee, \wedge, \neg$, and quantification $(\exists v)\psi$ and $(\forall v)\psi$.

Remark 4.1.1 It is important to note that we do *not* consider formulas $(\exists x)\psi$ where x ranges over $\mathcal{K}$. The range of the quantifiers is the universe of the structure. The interpretation of quantifiers is more formally precised below.

Definition 4.1.3 (Interpretation of First-Order Logic for $\mathcal{K}$-Structures)
Let $\sigma = (L_s, L_f)$ be a signature, and $\mathcal{D}$ be a $\mathcal{K}$-structure of signature σ. For any index term $h(x_1, \ldots, x_n)$ containing $n \geq 0$ variables, and any point $a \in A^n$, the interpretation of the function symbols of h in $\mathcal{D}$ extend to a natural interpretation $h^{\mathcal{D}}(a) \in A$ of h in $\mathcal{D}$ at the point a, where x_i is interpreted as a_i for $i = 1, \ldots, n$. The same holds with number terms $t(x_1, \ldots, x_n)$, the interpretation $t^{\mathcal{D}}(a)$ at a lying in $\mathbb{K}$. The interpretation of the relation symbols of L_s in $\mathcal{D}$ and the relations of $\mathcal{K}$ enable us to associate *truth values* with atomic formulas evaluated at point in A^n. Thus, if $h_1, \ldots, h_k$ are index terms and $t_1, \ldots, t_l$ are number terms containing the variables $x_1, \ldots, x_n$, and $a \in A^n$, we say that $r(h_1, \ldots, h_k)$ is `true` in $\mathcal{D}$ if $(h_1^{\mathcal{D}}, \ldots, h_k^{\mathcal{D}}) \in r^{\mathcal{D}}$ and `false` otherwise, and, for a l-ary relation rel of $\mathcal{K}$, $rel(t_1, \ldots, t_l)$ is `true` if $(t_1^{\mathcal{D}}, \ldots, t_k^{\mathcal{D}}) \in rel$ in $\mathcal{K}$, and `false` otherwise. The interpretation of the logical connectives $\vee$, $\wedge$ and $\neg$ is the usual. A formula $(\exists x)\psi$ where ψ contains variables $x, y_1, \ldots, y_n$, $n \geq 0$ is `true` at $b \in A^n$ if there exists $a \in A$ such that $\psi^{\mathcal{D}}$ is `true`, where x is interpreted as a and y_i is interpreted as b_i, for $i = 1, \ldots, n$. Similarly for a formula $(\forall x)\psi$ being interpreted as $\neg(\exists x)\neg\psi$. Proceeding recursively, we can associate with any formula ϕ with free variables $x_1, \ldots, x_n$ over σ and any $a \in A^n$ a truth value. If this value is `true` we say that $\mathcal{D}$ satisfies $\phi(a)$, and we denote this by $\mathcal{D} \models \phi(a)$. A formula with no variable is called a *sentence*. If T is a set of sentences over σ, $\mathcal{D}$ satisfies T if and only if $\mathcal{D}$ satisfies all the sentences of T. This is denoted as $\mathcal{D} \models T$. The $\mathcal{K}$-structures $\mathcal{D}$ such that $\mathcal{D} \models T$ are called the *models* of T. The sentences of T are called *axioms* and T is a *theory*.

Definition 4.1.4 (Order)
Let $\sigma = (L_s, L_f)$ be a signature. A $\mathcal{K}$-structure $\mathcal{D} = (\mathcal{U}, \mathcal{F})$ of signature σ with universe A is *ordered* if there is a binary relation symbol $\leq$ in L_s whose interpretation $\leq^{\mathcal{D}} \in \mathcal{U}$ is a total order on A.

Remark 4.1.2 Our order relation replaces the ranking function of [GM95, CM99, BCSS98, GG98]. In these papers, the ranking function is a function in L_f, defining a bijection between A and $\{0, \ldots, |A| - 1\}$. This uses the property that $\mathbb{R}$ is ordered and contains at least the natural numbers. It is not applicable in our setting of arbitrary structure, where no such requirement can be made on $\mathcal{K}$. We show that this requirement is not compulsory, and give general characterizations of complexity classes over $\mathcal{K}$, where no condition on $\mathcal{K}$ is assumed other than the one given in Chapter 1: $\mathcal{K}$ contains at least two constants and an equality relation.

Remark 4.1.3 It is checkable in first-order logic over $\mathcal{K}$-structures that a structure is ordered. Indeed, for a given signature $\sigma = (L_s, L_f)$ and a binary relation symbol $\leq \in L_s$, the ordered $\mathcal{K}$-structures of signature σ are the models of the theory defined as follows:

1. transitivity: $\forall x, y, z \quad (x \leq y) \wedge (y \leq z) \Rightarrow (x \leq z)$
2. symmetry: $\forall x \quad x \leq x$
3. anticommutativity: $\forall x, y \quad (x \leq y) \wedge (y \leq x) \Rightarrow (x = y)$
4. totality: $\forall x, y \quad (x \leq y) \vee (y \leq x)$

Based on this order relation, one can also define in first-order logic a lexicographic order over A^k for any $k \in \mathbb{N}$. A definition is the following:

$$\begin{aligned}
(x_1, \ldots, x_k) \quad &\leq_k \quad (y_1, \ldots, y_k) \\
&\Leftrightarrow \quad (x_1 \leq y_1) \vee \\
&\qquad (x_1 = y_1 \wedge x_2 \leq y_2) \vee \\
&\vdots \\
&\qquad (x_1 = y_1 \wedge \ldots \wedge x_{k-1} = y_{k-1} \wedge x_k \leq y_k)
\end{aligned}$$

Therefore, we will freely use the symbol $\leq_k$ for a total order over A^k. In what follows, we will only consider ordered $\mathcal{K}$-structures. Note also that the minimal element of A and the maximal one with respect to this order are definable in first-order logic. We will denote them 0 and $n^k - 1$ respectively.

Definition 4.1.5 (Fixed Point Rule)

Fix an signature $\sigma = (L_s, L_f)$, D a relation symbol of arity $r \geq 0$ and Z a function symbol of arity r, both not contained in this signature. Let $H(D, t_1, \ldots, t_r), I_1(D, t_1, \ldots, t_r), \ldots, I_{k-1}(D, t_1, \ldots, t_r)$, $k \geq 0$ be first-order formulas over the signature $(L_s \cup \{D\}, L_f \cup \{Z\})$ with free variables $t_1, \ldots, t_r$. Let $F_1(Z, t_1, \ldots, t_r), \ldots, F_k(Z, t_1, \ldots, t_r)$ be number terms over the signature $(L_s \cup \{D\}, L_f \cup$

$\{Z\}$) with free variables $t_1, \ldots, t_r$. We allow D to appear several times in H and the I_i's, but we do not require that its arguments are $(t_1, \ldots, t_r)$. The only restriction is that the number of free variables in H and the I_i's coincide with the arity of D. A similar remark holds for the F_i's and Z. For any $\mathcal{K}$-structure $\mathcal{D}$ of signature σ and any interpretation $\zeta : A^r \to \mathcal{K}$ of Z and $\Delta \subseteq A^r$ of D, respectively, the number terms $F_1(Z, t_1, \ldots, t_r), \ldots, F_k(Z, t_1, \ldots, t_r)$ define functions

$$\begin{array}{rcl} F^{\mathcal{D}}_{1,\zeta}, \ldots, F^{\mathcal{D}}_{k,\zeta} & : & A^r \to \mathcal{K} \\ u_1, \ldots, u_r & \to & \left\{ \begin{array}{l} [F_1(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ \vdots \\ [F_k(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}, \end{array} \right. \end{array}$$

where $[F_i(Z \leftarrow \zeta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}$ is obtained from $F_k(Z, t_1, \ldots, t_r)$ by replacing any occurrence of Z by ζ, any occurrence of t_j by u_j, and interpreting the whole number term in $\mathcal{D}$. Also, the formulas $H(D, t_1, \ldots, t_r)$ and $I_i(D, t_1, \ldots, t_r)$, $1 \le i \le k-1$ define relations

$$\begin{array}{rcl} H^{\mathcal{D}}_{\Delta}, I^{\mathcal{D}}_{i,\Delta} & \subseteq & A^r \\ u_1, \ldots, u_r & \to & \left\{ \begin{array}{l} [H(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ [I_1(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}} \\ \vdots \\ [I_{k-1}(D \leftarrow \Delta, t_1, \ldots, t_r \leftarrow u_1, \ldots u_r)]^{\mathcal{D}}. \end{array} \right. \end{array}$$

Let us now consider the sequence of pairs $\{\Delta^i, \zeta^i\}_{i \ge 0}$ with $\Delta^i \subseteq A^r$ and $\zeta^i : A^r \to \mathcal{K}$ inductively defined by

$$\begin{array}{rcl} \Delta^0(x) = \mathtt{false} & & \text{for all } x \in A^r \\ \zeta^0(x) = \mathbf{0} & & \text{for all } x \in A^r \\ \Delta^{i+1}(x) & = & \left\{ \begin{array}{ll} H^{\mathcal{D}}_{\Delta^i}(x) & \text{if } \Delta^i(x) = \mathtt{false} \\ \mathtt{true} & \text{otherwise} \end{array} \right. \\ \zeta^{i+1}(x) & = & \left\{ \begin{array}{ll} F^{\mathcal{D}}_{1,\zeta^i}(x) & \text{if } \neg\Delta^i(x) \wedge I^{\mathcal{D}}_{1,\Delta^i}(x), \text{else} \\ \vdots & \\ F^{\mathcal{D}}_{k-1,\zeta^i}(x) & \text{if } \neg\Delta^i(x) \wedge I^{\mathcal{D}}_{k-1,\Delta^i}(x), \text{else} \\ F^{\mathcal{D}}_{k,\zeta^i}(x) & \text{if } \neg\Delta^i(x), \text{else} \\ \zeta^i(x). & \end{array} \right. \end{array}$$

Since $\Delta^{i+1}(x)$ only differs from $\Delta^i(x)$ in case the latter is `false`, one has that $\Delta^j = \Delta^{j+1}$ for some $j < |A|^r$. In this case, we also have that $\zeta^j = \zeta^{j+1}$. We denote these fixed points by D^∞ and Z^∞ respectively and call them the *fixed points of* $H(D, t_1, \ldots, t_r)$, *the* $F_i(Z, t_1, \ldots, t_r)$, *and the* $I_i(D, t_1, \ldots, t_r)$ *on* $\mathcal{D}$. The *fixed point rule* is now stated as follows. If $F_i(Z, t_1, \ldots, t_r)$, $1 \le i \le k$ are number terms as previously, and $H(D, t_1, \ldots, t_r)$,

$I_i(D, t_1, \ldots, t_r)$, $1 \le i \le k$ are first-order formulas as previously, then

$$\mathbf{fp}[D(t_1, \ldots, t_r) \leftarrow H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$$

is a first-order formula of signature (L_s, L_f), and

$$\mathbf{fp}[Z(t_1, \ldots, t_r) \leftarrow F_i(Z, t_1, \ldots, t_r), I_i(D, t_1, \ldots, t_r), H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$$

is a number term of signature (L_s, L_f). Their interpretations on a given $\mathcal{K}$-structure $\mathcal{D}$ are D^∞ and Z^∞, respectively.

Remark 4.1.4 In [GM95, CM99, BCSS98], the fixed point rule allows to define only number terms. Thus, the authors need to introduce another rule, the maximization rule, which allows them to compute characteristic functions of relations of L_s as number terms, and perform logical operations such as AND, NOT, OR, by the appropriate use of multiplication, addition and a sign function. This approach, however, is not appropriate to our setting of an arbitrary structure, where no specific requirement is made on the functions and operations of the computational structure. We introduce therefore the possibility to define relations as fixed point, such as D^∞ in the definition above, which enables us to perform legally all kinds of logical operations in first-order logic. While our definition of the fixed point rule is more complicated, it makes the need for a maximization rule obsolete and enables us to prove our results over arbitrary structures, which can be specialized to be the real numbers with addition and order $\mathbb{R}_{ovs}$, or the complex numbers. Moreover, as stated in Remark 1.1.1, it allows us to capture also classes decided by machines having no constant node, by an appropriate specialization of the computational structure.

The fixed point rule allows us to defined an extension of the first-order logic, as follows.

Definition 4.1.6 (Fixed Point First-Order Logic for $\mathcal{K}$-Structures) Fixed point first-order logic for $\mathcal{K}$-structures, denoted by $\mathrm{FP}_\mathcal{K}$, is obtained by augmenting first-order logic $\mathrm{FO}_\mathcal{K}$ with the fixed point rule.

4.1.2 Characterizing $\mathrm{P}_\mathcal{K}$

It is clear that, for any signature $\sigma = (L_s, L_f)$, one can encode finite ordered $\mathcal{K}$-structures of signature σ in $\mathbb{K}^*$. A way of doing so is the following:

Definition 4.1.7 (Encoding of an Ordered $\mathcal{K}$-Structure in $\mathbb{K}^*$)

Let $\sigma = (L_s, L_f)$ be a signature containing a binary relation symbol $\leq$, and let $\mathcal{D}$ be an ordered $\mathcal{K}$-structure of signature σ. Next, replace all relations in the skeleton by the appropriate characteristic functions $\chi : A^k \to \{\mathbf{0}, \mathbf{1}\} \subseteq \mathbb{K}$. Thus, we get a structure with skeleton a set A and functions $X_1, \ldots, X_t$ of the form $X_i : A^{k_i} \to \mathbb{K}$. Each of these functions X_i can be represented by a tuple $\xi = (x_0, \ldots, x_{m_i-1}) \in \mathbb{K}^{m_i}$ with $m_i = |A|^{k_i}$ and $x_j = X_i(\overline{a}(j))$ where $\overline{a}(j)$ is the jth tuple in A^{k_i} with respect to the lexicographic order on A^{k_i} induced by $\leq$. The concatenation

$$e(\mathcal{D}) = \xi_1.\xi_2.\ldots.\xi_t$$

of these tuples gives the encoding $e(\mathcal{D}) \in \mathbb{K}^{m_1+\ldots+m_t}$.

Clearly, for a fixed finite signature and for any finite $\mathcal{K}$-structure $\mathcal{D}$ of size n, the size of $e(\mathcal{D})$ is bounded by some polynomial n^l where l depends only on the signature. Thus, appending zeros to $e(\mathcal{D})$ if necessary, we can also view $e(\mathcal{D}) = (x_0, \ldots, x_{l-1})$ as a single function $X_{\mathcal{D}} : A^l \to \mathbb{K}$. This means that one can encode an ordered $\mathcal{K}$-structure by a single function from the ordered set $\{0, \ldots, n^l - 1\}$ into $\mathbb{K}$. Moreover, this encoding can be performed in polynomial time.

Example 4.1.2 Let $\sigma = (L_s, L_f)$, where L_s contains only a binary relation symbol $\leq$ and L_f contains only a unary function symbol X. A simple class of ordered $\mathcal{K}$-structures of signature σ is obtained by letting A be a finite set, $\leq^{\mathcal{D}}$ be a total order on A, and $X^{\mathcal{D}}$ any unary function $X^{\mathcal{D}} : A \to \mathbb{K}$. Since $\leq^{\mathcal{D}}$ induces a bijection between A and $\{0, \ldots, n-1\}$, where $n = |A|$, this $\mathcal{K}$-structure is a point in $\mathbb{K}^n \subset \mathbb{K}^*$. Conversely, for any point $\overline{x} \in \mathbb{K}^*$, there is a $\mathcal{K}$-structure of signature σ denoting $\overline{x}$. Thus, there is a bijection between $\mathcal{K}$-structures in this class and $\mathbb{K}^*$.

Definition 4.1.8 (Decision Problem of Ordered $\mathcal{K}$-structures)

Let $\sigma = (L_s, L_f)$ be a signature containing a binary relation symbol $\leq$. We denote by $Struct(\sigma)$ the set of ordered $\mathcal{K}$-structures of signature σ. A decision problem of ordered $\mathcal{K}$-structures of signature σ is a subset of $Struct(\sigma)$. Modulo the polynomial time encoding of ordered $\mathcal{K}$-structures in $\mathbb{K}^*$ described in Definition 4.1.7 above, any decision problem of ordered $\mathcal{K}$-structures can be seen as a decision problem over $\mathcal{K}$ in the BSS model. Conversely, as shown by Example 4.1.2 above, any decision problem over $\mathcal{K}$ in the BSS model can be seen as a decision problem of ordered $\mathcal{K}$-structures. Since the encoding of Definition 4.1.7 can be performed in polynomial time, for $\mathrm{P}_{\mathcal{K}}$ and complexity classes above, the two notions coincide.

This yields the following characterization of $\mathrm{P}_{\mathcal{K}}$.

Theorem 4.1.1 *Let S be a decision problem of ordered $\mathcal{K}$-structures. Then the following statements are equivalent.*

(i) $S \in \mathrm{P}_{\mathcal{K}}$.

(ii) *There exists a sentence ψ in fixed point first-order logic such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

PROOF. Let us first prove the **(i)** $\Rightarrow$ **(ii)** direction. Assume that the signature is $\sigma = (L_s, L_f)$, and that it contains a binary relation symbol $\leq$. Let $S \in \mathrm{P}_{\mathcal{K}}$, and let M be a polynomial time BSS machine over $\mathcal{K}$ deciding S. Assume that, for any ordered $\mathcal{K}$-structure $\mathcal{D}$ of size n, encoded in $\mathbb{K}^*$ by $e(\mathcal{D})$, the computation time of M on $e(\mathcal{D})$ is bounded by n^m, for some $m \in \mathbb{N}$. Assume also that the size of $e(\mathcal{D})$ is n^m. This is without loss of generality since it suffices to add some padding $\mathbf{0}$'s to the encoding $e(\mathcal{D})$ of $\mathcal{D}$. Assume also without loss of generality that the state space of M is bounded by n^m. A point in this space has coordinates $(x_0, x_1, \ldots, x_{n^m-1})$. In the following, we will use the formalism of a Turing machine, with a scanning head moving in $\mathbb{K}^m$ instead of a state space $\mathbb{K}_*$.

Consider the lexicographic order $\leq_m$ on A^m induced by the order $\leq$ on A. By Remark 4.1.3, $\leq_m$ is defined in first-order logic, and induces a bijection between A^m and the set $\{0, \ldots, n^m - 1\}$. Therefore, for any $t \in A^m$ we will define:

$$\begin{aligned} t-1 &= \max_{s \in A^m} \{s <_m t\} \\ t+1 &= \min_{s \in A^m} \{t <_m s\}. \end{aligned}$$

It is clear that these elements are definable in first-order logic over ordered $\mathcal{K}$-structures, thus we will freely use these notations hereafter. Note however that, when t is minimal in A^m, we have $t - 1 = t$. A similar remark holds for t maximal with $t + 1 = t$. Following this notation, we will identify A^m with $\{0, \ldots, n^m - 1\}$.

Next, we assume that the number of nodes of M is bounded by 2^k, $k \in \mathbb{N}$. Note that k is fixed, independent of $\mathcal{D}$ or n. Thus, the nodes of M will be denoted as elements in A^k, with the proviso that $|A| \geq 2$. We also assume that they correspond to the first elements of A^m with respect to the lexicographic order $\leq_m$. Since their number is constant, they are all definable in first-order logic. We will therefore denote them with constants $v_{type} \in A^m$, where *type* denotes the type of the node as in Definition 1.1.2. Recall also the constant $0 \in A^m$, corresponding the the minimal element of A^m as in Remark 4.1.3.

Consider now the following relation symbol, not contained in L_s, of arity $4m$:

$$\begin{aligned}\mathtt{Node}(v,t,pos,c) \quad = \quad & \mathtt{true} \text{ iff} \\ & \text{the current node of } M \text{ at step } t \text{ is } v \text{ and} \\ & \begin{cases} pos = 0 & \text{and the head of } M \text{ is on cell } c \\ pos = 1 & \text{and the head of } M \text{ is not on cell } c. \end{cases}\end{aligned}$$

Consider also the following function symbol, not contained in L_f, of arity $2m$:

$$\mathtt{Cell}(t,c) = \text{content of cell } c \text{ at step } t.$$

If Cell can be defined in $\mathrm{FP}_{\mathcal{K}}$ by a number term Z, then the implication (**i**) $\Rightarrow$ (**ii**) holds. The $\mathcal{K}$-structure $\mathcal{D}$ is accepted by M if and only if after n^m steps, the content of the first cell is $\mathbf{1}$. That is, $\mathtt{Cell}(n^m-1,0) = \mathbf{1}$, were $n^m - 1$ is the maximal element in A^m, as defined in Remark 4.1.3. Node and Cell can be defined inductively as follows:

$$\begin{aligned}
\mathtt{Node}(input,0,pos,c) \quad &\leftarrow \quad [(pos = 0) \vee (c = 0)] \wedge [(pos = 1) \vee (c \neq 0)] \\
\mathtt{Node}(v,0,pos,c) \quad &\leftarrow \quad \mathtt{false} \text{ if } v \neq input \\
\mathtt{Node}(v,t+1,pos,c) \quad &\leftarrow \quad \mathtt{Node}(v_{op_i},t,pos,c) \text{ for } v_{op_i} \text{ s.t. } v = \beta(v_{op_i}) \\
&\vee \quad \mathtt{Node}(v_{shift_l},t,pos,c+1) \text{ for } v_{sh_l} \text{ s.t. } v = \beta(v_{shift_l}) \\
&\vee \quad \mathtt{Node}(v_{shift_r},t,pos,c-1) \text{ for } v_{sh_r} \text{ s.t. } v = \beta(v_{shift_r}) \\
&\vee \quad \mathtt{Node}(v_{copy_l},t,pos,c) \text{ for } v_{copy_l} \text{ s.t. } v = \beta(v_{copy_l}) \\
&\vee \quad \mathtt{Node}(v_{copy_r},t,pos,c) \text{ for } v_{copy_r} \text{ s.t. } v = \beta(v_{copy_r}) \\
&\vee \quad \mathtt{Node}(v_{switch},t,pos,c) \text{ for } v_{switch} \text{ s.t. } v = \beta(v_{switch}) \\
&\vee \quad \mathtt{Node}(v_{output},t,pos,c) \text{ for } v_{output} \text{ s.t. } v = \beta(v_{output}) \\
&\vee \quad \mathtt{Node}(v_{rel_i},t,0,c) \wedge rel_i(\mathtt{Cell}(t,c),\ldots,\mathtt{Cell}(t,c+k_i)) \\
&\quad\;\; \wedge pos = 0 \quad \text{for } v_{rel_i} \text{ s.t. } v = \beta^+(v_{rel_i}) \\
&\vee \quad \mathtt{Node}(v_{rel_i},t,0,c) \wedge \neg rel_i(\mathtt{Cell}(t,c),\ldots,\mathtt{Cell}(t,c+k_i)) \\
&\quad\;\; \wedge pos = 0 \quad \text{for } v_{rel_i} \text{ s.t. } v = \beta^-(v_{rel_i})
\end{aligned}$$

and:

$$\begin{array}{rcl} \mathtt{Cell}(0,c) & \leftarrow & X^{\mathcal{D}}(c) \\ \mathtt{Cell}(t+1,c) & \leftarrow & \begin{cases} op_i(\mathtt{Cell}(t,c),\ldots,\mathtt{Cell}(t,c+k_i)) & \text{if } \mathtt{Node}(v_{op_i},t,0,c) \\ \mathtt{Cell}(t,c+1) & \text{if } \mathtt{Node}(v_{copy_l},t,0,c) \\ \mathtt{Cell}(t,c-1) & \text{if } \mathtt{Node}(v_{copy_r},t,0,c) \\ \mathtt{Cell}(t,c+1) & \text{if } \mathtt{Node}(v_{switch},t,0,c) \\ \mathtt{Cell}(t,c-1) & \text{if } \mathtt{Node}(v_{switch},t,0,c-1) \\ \mathtt{Cell}(t,c) & \text{otherwise} \end{cases} \end{array}$$

This proves that `Cell` can be defined in $\mathrm{FP}_{\mathcal{K}}$ as the fixed point of the inductive definition above, from which (**i**) $\Rightarrow$ (**ii**) follows.

To prove (**ii**) $\Rightarrow$ (**i**), we only need to prove that formulas and number terms defined with the fixed point rule can be evaluated in polynomial time. Since the number of updates is polynomially bounded, one only needs to apply the inductive definition a polynomial number of times, which ends the proof.

□

Example 4.1.3 Recall the signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ given in Example 4.1.1 for encoding inputs to the 4FEAS problem. Consider $X : A \to \mathcal{K}$ whose interpretation in a $\mathcal{K}$-structure $\mathcal{D}$ of signature $(L_s, L_f \cup X)$ instanciates the variables of the polynomial. The problem 4EVAL of checking that a given polynomial of degree 4 evaluates to 0 at a given point is in $\mathrm{P}_{\mathcal{K}}$. It can be formulated as follows:

There exists a sentence ϕ in fixed point first-order logic over the signature $(L_s, L_f \cup X)$ such that

$$4\mathrm{EVAL} = \{\mathcal{D} | \mathcal{D} \models \phi\}.$$

4.2 Second-Order Logic on $\mathcal{K}$-structures

4.2.1 Definitions

Recall that $\mathcal{K}$-structures can be naturally seen as points in $\mathcal{K}^*$. A decision problem of ranked $\mathcal{K}$-structures is then a decision problem, where the (positive) inputs are encoded as $\mathcal{K}$-structures, and satisfy the natural property of being ranked.

Definition 4.2.1 (Existential Second-Order Logic For $\mathcal{K}$-Structures)
We say that ψ is an *existential second-order sentence* (of signature $\sigma = (L_s, L_f)$) if $\psi = \exists Y_1, \ldots, \exists Y_r \phi$, where ϕ is a first-order sentence in $\mathrm{FO}_{\mathcal{K}}$ of signature $(L_s, L_f \cup \{Y_1, \ldots, Y_r\})$.

The function symbols $Y_1, \ldots, Y_r$ are called *function variables.* Existential second-order sentences are interpreted in $\mathcal{K}$-structures $\mathcal{D}$ of signature σ in the following way: a sentence $\mathcal{D} \models (\psi = \exists Y_1, \ldots, \exists Y_r \phi)$ if there exist functions $X_1, \ldots, X_r : A^{r_i} \to \mathbb{K}$ with r_i the arity of Y_i, such that the interpretation of ψ taking $Y_i^{\mathcal{D}}$ to be X_i yields `true`. The set of second-order sentences together with this interpretation constitutes *existential second-order logic* and is denoted by $\exists \mathrm{SO}_{\mathcal{K}}$.

Existential second-order logic has at least the expressive power of fixed point first-order logic. Indeed:

Proposition 4.2.1 *For every sentence ψ of signature σ in $\mathrm{FP}_{\mathcal{K}}$ there exists a sentence $\widetilde{\psi}$ of signature σ in $\exists \mathrm{SO}_{\mathcal{K}}$ such that, for every $\mathcal{K}$-structure $\mathcal{D}$,*

$$\mathcal{D} \models \psi \textit{ if and only if } \mathcal{D} \models \widetilde{\psi}.$$

PROOF. It suffices to prove that the fixed point rule of Definition 4.1.5 can be expressed within existential second-order logic over $\mathcal{K}$-structures. Assume F_i, I_i, H, $1 \leq i \leq k$, Z and D are as in Definition 4.1.5. Every occurrence of $\mathbf{fp}[Z(t_1, \ldots, t_r) \leftarrow F_i(Z, t_1, \ldots, t_r), I_i(D, t_1, \ldots, t_r), H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$ and of $\mathbf{fp}[D(t_1, \ldots, t_r) \leftarrow H(D, t_1, \ldots, t_r)](u_1, \ldots, u_r)$ in ψ will be replaced by $Z(u_1, \ldots, u_r)$ and $D(u_1, \ldots, u_r)$, respectively. Denote by ϕ the resulting formula. For any r-ary function $F : A^r \to \mathbb{K}$, denote by $\widetilde{F}$ the $r{-}ary$ formula with variables $(u_1, \ldots, u_r)$ defined by $\widetilde{F}(u_1, \ldots, u_r) = (F(u_1, \ldots, u_r) = \mathbf{1})$.

Then, $\widetilde{\psi}$ is

$$\begin{array}{ll}
 & \exists Z \exists D \forall (u_1, \ldots, u_r) \\
 & \widetilde{D}(u_1, \ldots, u_r) \Leftrightarrow H(\widetilde{D}, u_1, \ldots, u_r) \\
\wedge & F_1(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow I_1(\widetilde{D}, u_1, \ldots, u_r) \\
 & \vee F_2(Z, (u_1, \ldots, u_r)) = Z(u_1, \ldots, u_r) \Leftrightarrow \left\{ \begin{array}{ll} & \neg(I_1(\widetilde{D}, u_1, \ldots, u_r)) \\ \wedge & I_2(\widetilde{D}, u_1, \ldots, u_r) \end{array} \right. \\
 & \vdots
\end{array}$$

$$\vee F_{k-1}(Z,(u_1,\ldots,u_r)) = Z(u_1,\ldots,u_r) \Leftrightarrow \begin{cases} & \neg(I_1(\widetilde{D},u_1,\ldots,u_r)) \\ \vdots & \\ \wedge & \neg(I_{k-2}(\widetilde{D},u_1,\ldots,u_r)) \\ \wedge & I_{k-1}(\widetilde{D},u_1,\ldots,u_r) \end{cases}$$

$$\vee F_{k}(Z,(u_1,\ldots,u_r)) = Z(u_1,\ldots,u_r) \Leftrightarrow \begin{cases} & \neg(I_1(\widetilde{D},u_1,\ldots,u_r)) \\ \vdots & \\ \wedge & \neg(I_{k-2}(\widetilde{D},u_1,\ldots,u_r)) \\ \wedge & \neg(I_{k-1}(\widetilde{D},u_1,\ldots,u_r)) \end{cases}$$

$\wedge \quad \phi.$

□

4.2.2 Characterizing $\mathrm{NP}_{\mathcal{K}}$

Theorem 4.2.1 *Let S be a decision problem of ordered $\mathcal{K}$-structures. Then the following statements are equivalent.*

(i) $S \in \mathrm{NP}_{\mathcal{K}}$.

(ii) *There exists an existential second-order sentence ψ such that $S = \{\mathcal{D} | \mathcal{D} \models \psi\}$.*

Proof. Let $S \in \mathrm{NP}_{\mathcal{K}}$ be a problem of ordered $\mathcal{K}$-structures of signature $\sigma = (L_s, L_f)$. By definition of $\mathrm{NP}_{\mathcal{K}}$, there exists $r \in \mathbb{N}$, a function symbol Y of arity r not in L_f, and a decision problem H of ordered $\mathcal{K}$-structures of signature $(L_s, L_f \cup Y)$, such that H belongs to $\mathrm{P}_{\mathcal{K}}$ and

$$S = \{\mathcal{D} \in \mathit{Struct}(\sigma) | \exists Y\ (\mathcal{D}, Y) \in H\}.$$

By Theorem 4.1.1, there exists a fixed point first-order formula ϕ that describes H, and thus,

$$\mathcal{D} \in S \text{ if and only if } \mathcal{D} \models \exists Y \phi.$$

By Proposition 4.2.1, ϕ can be replaced by an equivalent second-order formula $\exists Z \varphi$ with φ a first-order formula. Then,

$$\mathcal{D} \in S \text{ if and only if } \mathcal{D} \models \exists Y \exists Z \varphi,$$

which shows **(i)** $\Rightarrow$ **(ii)**. For the other direction, it suffices to guess an interpretation for the second-order quantified functions, and to check in $\mathrm{P}_{\mathcal{K}}$ that the first-order formula induced is satisfied.

□

Example 4.2.1 Recall the signature $(L_s, L_f) = (\emptyset, \{f_0, f_1, f_2, f_3, f_4\})$ given in Example 4.1.1 for encoding inputs to the 4FEAS problem. Consider $X : A \to \mathcal{K}$ whose interpretation in a $\mathcal{K}$-structure $\mathcal{D}$ instanciates the variables of the polynomial. Consider a formula ϕ in fixed point first-order logic over the signature $(L_s, L_f \cup X)$, whose interpretation for a $\mathcal{K}$-structure $\mathcal{D}$ checks whether the evaluation of the polynomial at the point $X^{\mathcal{D}} \in \mathcal{K}^{|A|}$ is 0. A instance of the result above is

$$4\text{FEAS} = \{\mathcal{D} | \mathcal{D} \models \exists X \, \phi\}.$$

Note that by Proposition 4.2.1 ϕ can be expressed in existential second-order logic.

Chapter 5

Partial Recursion and Computability

In this chapter, we recall the notion of partial recursive and primitive recursion over the real numbers introduced in [BSS89]. These classes of functions could be easily translated to any structure other than $\mathbb{R}$, yet, some of their features are unsatisfactory, and improve the difficulty of refining the mechanisms of recursion in order to capture interesting complexity classes. Thus, we suggest our own definition of partial recursive and primitive recursive functions over an arbitrary structure, and prove that our partial recursive functions over a structure $\mathcal{K}$ are the functions computable by a BSS machine over $\mathcal{K}$. These results are published in [BCdNM02, BCdNM03a, BCdNM04a].

5.1 Primitive Recursion à la BSS

We recall the notions of partial recursive and primitive recursive functions over a ring defined by Blum, Shub and Smale in their seminal paper [BSS89]. These definitions apply only when the operations of the structure form an ordered ring.

Definition 5.1.1 (BSS partial recursive functions over a ring R) The class $P_R^{<\infty}$ of finite dimensional *BSS partial recursive functions over* R is the smallest class of partial functions (maps) $f : R^l \to R^m$ $(l, m < \infty)$, containing the basic functions:

(i) the polynomial (rational if R is a field) functions $f : R^l \to R^m$ (and hence e.g. the successor, the constants and coordinate projection functions),

(ii) the characteristic function $\chi : R \to \{\mathbf{0}, \mathbf{1}\}$ where

$$\chi(x) = \begin{cases} \mathbf{-1} & \text{if } x < \mathbf{0} \\ \mathbf{0} & \text{if } x = \mathbf{0} \\ \mathbf{1} & \text{if } x > \mathbf{0}, \end{cases}$$

and closed under the operations of

(a) composition,

(b) juxtaposition,

(c) primitive recursion and

(d) minimization

For (a), assume $g : R^m \to R^n$ and $h : R^l \to R^m$ are given partial functions. Then the composition f is defined by

$$f(x) = g(h(x)).$$

For (b), assume $f_i : R^l \to R^{m_i}$, $(i = 1, \ldots, k)$. Then the juxtaposition $F = (f_1, \ldots, f_k) : R^l \to R^{m_1 + \ldots + m_k}$ is given by

$$F(x) = (f_1(x), \ldots, f_k(x)).$$

For (c), assume $g : R^l \to R^l$ is given. *Primitive recursion* then defines a partial map $G : \mathbb{N} \times R^l \to R^l$ by

$$\begin{aligned} G(0, x) &= x \\ G(t+1, x) &= g(G(t, x)). \end{aligned}$$

Finally, for (d), assume that $F : \mathbb{N} \times R^l \to R$ is given. *Minimization* defines a partial function $L : R^l \to \mathbb{N}$ by

$$L(x) = \begin{cases} \min_{t \in \mathbb{N}} \{F(t, x) = \mathbf{0}\} & \text{if } \exists t \in \mathbb{N} : \ F(t, x) = \mathbf{0} \\ \perp & \text{otherwise.} \end{cases}$$

Then, the following result holds

Theorem 5.1.1 *[BSS89] $P_R^{<\infty}$ is exactly the class of finite dimensional partial functions computable by a BSS machine over R.*

Remark 5.1.1 It is noticeable that $P_R^{<\infty}$ is a class of finite dimensional functions: they have type $R^l \to R^m$, i.e. their arguments are tuples of real numbers, as well as their outputs. Therefore, $P_R^{<\infty}$ captures only finite dimensional functions, i.e. functions that can be computed in constant space by a BSS machine. It is clear that there exist BSS computable functions that do not belong to this class, for instance all functions that output a binary encoding in $\{0, 1\}^*$ of an integer result.

This class of functions would seem to be a good candidate for adapting the tailoring techniques mentioned above in order to characterize complexity classes. However, the fact that these functions are finite dimensional is a strong handicap, and needs to be corrected. Moreover, the use of a special kind of argument, namely a natural number, for controlling primitive recursion as well as minimization, makes it also more complicated for a clear understanding of the processes at hand. Finally, the requirements that the structure be a ring, present in the definition of the basic functions, is not very general. For these reasons, we propose below another definition of partial recursive and primitive recursive functions in the BSS model of computation.

5.2 Partial Recursive and Primitive Recursive Functions

Our notion of recursion applies not on tuples of elements of $\mathbb{K}$ as above, but on *words* of elements of $\mathbb{K}$, of variable length. Also, instead of using a particular type of argument, a natural number, to control recursions, we use the length of our arguments to do so.

5.2.1 Definitions

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.

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 5.2.1 (Basic Functions)

Basic functions are the following four kinds of functions:

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

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

(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

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

Definition 5.2.2 (Partial Recursive Functions)

The set of *partial recursive functions* over $\mathcal{K}$, denoted by $\mathrm{PR}_{\mathcal{K}}$, 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 $f : (\mathbb{K}^*)^2 \to \mathbb{K}^*$

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

(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:

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

Definition 5.2.3 (Primitive Recursive Functions) The set of partial recursive functions defined without using the minimization operator, denoted as $\mathrm{PrR}_{\mathcal{K}}$, is called the set of *primitive recursive functions.*

Remark 5.2.1 (i) The formal definition of function tl is actually a primitive recursive definition with no recurrence argument. However, when we introduce the notion of safe recursion in Section 6.1, this function 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 simplification 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 we do not lose in expressivity since g may still depend on the value of the first element of $\overline{y}$.

(vi) Our choice here to consider arguments as words of elements in $\mathbb{K}$, and to use the length of the arguments to control recursion and minimization in a different fashion from Definition 5.1.1 allows us to capture non-finite dimensional functions over arbitrary structures. Also, we do not need anymore that the structure be a ring.

The following result is immediate.

Proposition 5.2.1 *The set of partial recursive (resp. primitive recursive) functions over* $\mathbb{Z}_2 = \{\{0,1\}, =, \mathbf{0}, \mathbf{1}\}$ *coincides with the classical partial recursive (resp. primitive recursive) functions.*

□

5.2.2 Partial Recursive Functions and BSS Computable Functions

Theorem 5.2.1 *Over any structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, *a function is BSS computable if and only if it can be defined in* $\mathrm{PR}_{\mathcal{K}}$.

PROOF.

Let M be a BSS machine on structure $\mathcal{K}$.

In what follows, we represent the contents of the tape of M by a pair of variables $(\overline{x}, \overline{y})$ in $(\mathbb{K}^*)^2$ such that the written part of the tape is given by $\overline{x}^R.\overline{y}$ (here $\overline{x}^R$ is the reversed word of $\overline{x}$) and the head of the machine is on the first letter of $\overline{y}$.

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. In the following definitions, node number q will be coded by the variable (word) $\mathbf{1}^q$ of length q.

We assume without loss of generality that, when M halts on any input $\overline{y}$, the left part of the tape is empty.

Let q ($q \in \mathbb{N}$) be a move node as in Figure 5.1.

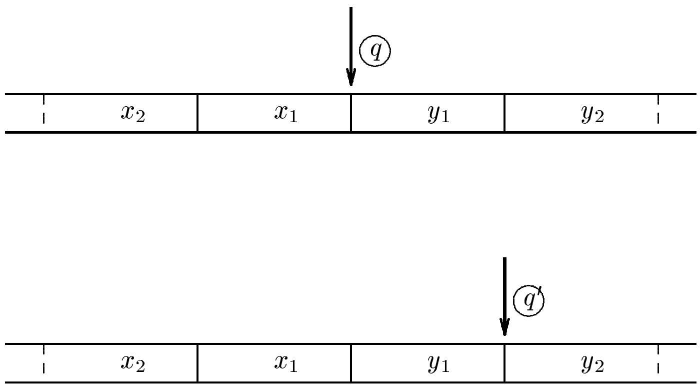

Figure 5.1: A "move right" Node

Three functions are associated with this node:

$$\begin{array}{lll} \mathcal{G}_i(\overline{x}, \overline{y}) & = \mathbf{1}^{q'} & \\ \mathcal{H}_i(\overline{x}, \overline{y}) & = \mathsf{tl}(\overline{x}) & \text{or } \mathsf{hd}(\overline{y}).\overline{x} \\ \mathcal{I}_i(\overline{x}, \overline{y}) & = \mathsf{hd}(\overline{x}).\overline{y} & \text{or } \mathsf{tl}(\overline{y}) \end{array}$$

according if one moves left or right.

Function $\mathcal{G}_i$ returns the encoding of the following node in the computation tree of M, function $\mathcal{H}_i$ returns the encoding of the left part of the tape, and function $\mathcal{I}_i$ returns the encoding of the right part of the tape.

Let q ($q \in \mathbb{N}$) be a computation node associated to some operation *op* of arity n of the structure, as in Figure 5.2. We also write Op for the corresponding basic operation.

Functions $\mathcal{G}_i$, $\mathcal{H}_i$, $\mathcal{I}_i$ associated with this node are now defined as follows:

$$\begin{array}{ll} \mathcal{G}_i(\overline{x}, \overline{y}) & = \mathbf{1}^{q'} \\ \mathcal{H}_i(\overline{x}, \overline{y}) & = \overline{x} \\ \mathcal{I}_i(\overline{x}, \overline{y}) & = \mathsf{cons}(\mathsf{Op}(\mathsf{hd}(\overline{y}), \ldots, \mathsf{hd}(\mathrm{tl}^{(n-1)}(\overline{y}))), \mathsf{tl}(\overline{y})). \end{array}$$

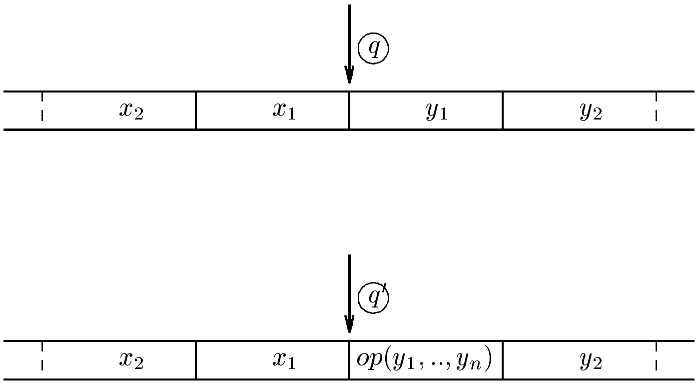

Figure 5.2: An "op" Node

Let q ($q \in \mathbb{N}$) be a copy node associated to some operation *op* of arity n of the structure, as in Figure 5.3.

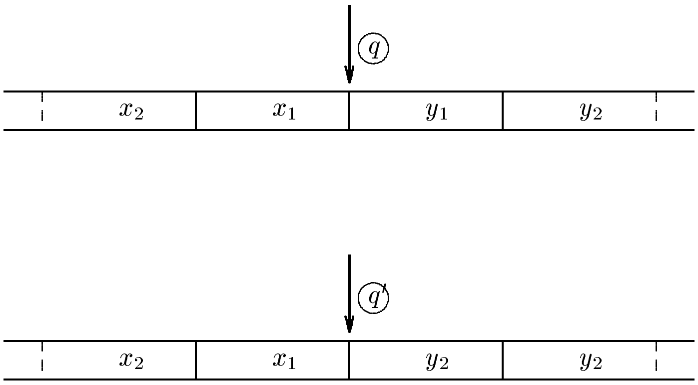

Figure 5.3: A "copy left" Node

Functions $\mathcal{G}_i$, $\mathcal{H}_i$, $\mathcal{I}_i$ associated with this node are now defined as follows:

$$
\begin{aligned}
\mathcal{G}_i(\overline{x}, \overline{y}) &= \mathbf{1}^{q'} \\
\mathcal{H}_i(\overline{x}, \overline{y}) &= \overline{x} \\
\mathcal{I}_i(\overline{x}, \overline{y}) &= \begin{cases} \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\overline{x})), \mathsf{tl}(\overline{x})) & \text{for a copy left} \\ \mathsf{cons}(\mathsf{hd}(\overline{x}), \mathsf{cons}(\mathsf{hd}(\overline{x}), \mathsf{tl}(\mathsf{tl}(\overline{x})))) & \text{for a copy right} \\ \mathsf{cons}(\mathsf{hd}(\mathsf{tl}(\overline{x})), \mathsf{cons}(\mathsf{hd}(\overline{x}), \mathsf{tl}(\mathsf{tl}(\overline{x})))) & \text{for a switch.} \end{cases}
\end{aligned}
$$

Let q ($q \in \mathbb{N}$) be a branch node corresponding to a relation *rel* of arity n of the structure, as in Figure 5.4.

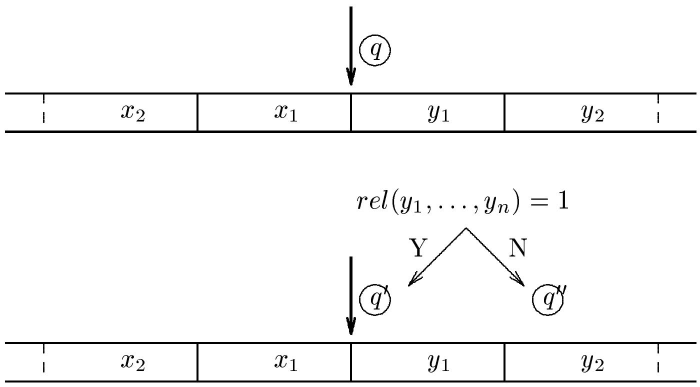

Figure 5.4: A "rel" Node

The three functions associated with this node are now:

$$
\begin{aligned}
\mathcal{G}_i(\overline{x},\overline{y}) &= \mathsf{Select}(\mathsf{Rel}(\mathsf{hd}(\overline{x}),\ldots,\mathsf{hd}(\mathsf{tl}^{(n-1)}(\overline{y}))),\mathbf{1}^{q'},\mathbf{1}^{q''})\\
\mathcal{H}_i(\overline{x},\overline{y}) &= \overline{x}\\
\mathcal{I}_i(\overline{x},\overline{y}) &= \overline{y}.
\end{aligned}
$$

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

$$
\mathsf{Equal}_k(\overline{x}) = \begin{cases} \mathbf{1} & \text{if } \overline{x} = \mathbf{1}^k \\ \epsilon & \text{otherwise.} \end{cases}
$$

We can now define the primitive recursive functions $\mathsf{next}_{\mathsf{state}}$, $\mathsf{next}_{\mathsf{left}}$ and $\mathsf{next}_{\mathsf{right}}$ 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:

$$
\begin{aligned}
\mathsf{next}_{\mathsf{state}}(\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(\mathsf{Equal}_0(\overline{s}),\mathcal{G}_0(\overline{x},\overline{y}),\mathsf{Select}\,(\mathsf{Equal}_1(\overline{s}),\mathcal{G}_1(\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(\mathsf{Equal}_m(\overline{s}),\mathcal{G}_m(\overline{x},\overline{y}),\epsilon)\ldots))\,,\\
\mathsf{next}_{\mathsf{left}}(\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(\mathsf{Equal}_0(\overline{s}),\mathcal{H}_0(\overline{x},\overline{y}),\mathsf{Select}\,(\mathsf{Equal}_1(\overline{s}),\mathcal{H}_1(\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(\mathsf{Equal}_m(\overline{s}),\mathcal{H}_m(\overline{x},\overline{y}),\epsilon)\ldots))
\end{aligned}
$$

and

$$
\begin{aligned}
\mathsf{next}_{\mathsf{right}}(\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(\mathsf{Equal}_0(\overline{s}),\mathcal{I}_0(\overline{x},\overline{y}),\mathsf{Select}\,(\mathsf{Equal}_1(\overline{s}),\mathcal{I}_1(\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(\mathsf{Equal}_m(\overline{s}),\mathcal{I}_m(\overline{s},\overline{x},\overline{y}),\epsilon)\ldots))\,.
\end{aligned}
$$

Note that the classical techniques for coding simultaneous primitive recursion with primitive recursion, as in [Ros84], apply without noticeable change to our setting. For a formal proof of this statement, see the proof of Proposition 6.2.3 in the next chapter. The following functions $\mathsf{comp}_{\mathsf{state}}$,$\mathsf{comp}_{\mathsf{left}}$ and $\mathsf{comp}_{\mathsf{right}}$, defined below with an *ad hoc* simultaneous primitive recursion scheme, are primitive recursive. They give, on input $(\overline{t}, \overline{x}, \overline{y})$, the encoding of the state of the machine reached after k computation nodes, where k is the length of the word $\overline{t} \in \mathbb{K}^*$, and also the encoding of the left part and the right part of the tape.

$$\mathsf{comp}_{\mathsf{state}}(\epsilon, \overline{x}, \overline{y}), \mathsf{comp}_{\mathsf{left}}(\epsilon, \overline{x}, \overline{y}), \mathsf{comp}_{\mathsf{right}}(\epsilon, \overline{x}, \overline{y}) = \epsilon, \overline{x}, \overline{y}$$

and

$$\begin{aligned}
\mathsf{comp}_{\mathsf{state}}(a.\overline{t}, \overline{x}, \overline{y}) &= \mathsf{next}_{\mathsf{state}}\big(\mathsf{comp}_{\mathsf{state}}(\overline{t}, \overline{x}, \overline{y}), \mathsf{comp}_{\mathsf{left}}(\overline{t}, \overline{x}, \overline{y}),\\
&\qquad \mathsf{comp}_{\mathsf{right}}(\overline{t}, \overline{x}, \overline{y})\big)\\
\mathsf{comp}_{\mathsf{left}}(a.\overline{t}, \overline{x}, \overline{y}) &= \mathsf{next}_{\mathsf{left}}\big(\mathsf{comp}_{\mathsf{state}}(\overline{t}, \overline{x}, \overline{y}), \mathsf{comp}_{\mathsf{left}}(\overline{t}, \overline{x}, \overline{y}),\\
&\qquad \mathsf{comp}_{\mathsf{right}}(\overline{t}, \overline{x}, \overline{y})\big)\\
\mathsf{comp}_{\mathsf{right}}(a.\overline{t}, \overline{x}, \overline{y}) &= \mathsf{next}_{\mathsf{right}}\big(\mathsf{comp}_{\mathsf{state}}(\overline{t}, \overline{x}, \overline{y}), \mathsf{comp}_{\mathsf{left}}(\overline{t}, \overline{x}, \overline{y}),\\
&\qquad \mathsf{comp}_{\mathsf{right}}(\overline{t}, \overline{x}, \overline{y})\big)\,.
\end{aligned}$$

Then, it suffices to define:

$$\begin{aligned}
f_S(\overline{t}, \overline{y}) &= \mathsf{comp}_{\mathsf{state}}(\overline{t}, \epsilon, \overline{y})\\
f_L(\overline{t}, \overline{y}) &= \mathsf{comp}_{\mathsf{left}}(\overline{t}, \epsilon, \overline{y})\\
f_R(\overline{t}, \overline{y}) &= \mathsf{comp}_{\mathsf{right}}(\overline{t}, \epsilon, \overline{y})
\end{aligned}$$

On input $\overline{y} \in \mathbb{K}^*$, the final state is then reached after k steps, where

$$\mathbf{1}^k = \mu \overline{s}_t \left(\mathsf{Equal}_1\left(f_S(\overline{s}_t, \overline{y})\right)\right).$$

The encoding of the machine tape is then given by $f_R(\mathbf{1}^k, \overline{y})$, i.e. by

$$f_R(\mu \overline{s}_t \left(\mathsf{Equal}_1\left(f_S(\overline{s}_t, \overline{y})\right)\right), \overline{y}).$$

For the other direction, it is clear that any function in $\mathrm{PR}_{\mathcal{K}}$ can be computed by a BSS machine over $\mathcal{K}$.

□

5.3 Circuit-Based Arguments for Theorem 5.2.1

This section is devoted to give a simpler proof for Theorem 5.2.1. The existence of similar results for classical Turing machines, and the notions of uniform families of circuits described by classical Turing machines, provide a strong link between this result and similar classical results over boolean circuits. We provide the following circuit-based proof to emphasize these links.

More precisely, we use Proposition 1.2.1 to reduce BSS machine computations to circuit evaluation. Recall, the output of this reduction, on an input of size n is a description of a circuit performing the same computation the machine does when restricted to inputs of size n. Since in this proposition we replaced the machine constants by variables, the resulting circuit can be encoded over $\{\mathbf{0},\mathbf{1}\}$ and, as it turns out, the whole reduction can be carried out by a Turing machine. Therefore we can invoke classical results to show that the reduction can be simulated by classical partial recursive and, a fortiori, by such kind of functions over an arbitrary structure $\mathcal{K}$. Next, we give a simulation of the circuit given by this reduction by partial recursive functions over an arbitrary structure $\mathcal{K}$.

5.3.1 Simulation of a Circuit by Partial Recursive Functions

Lemma 5.3.1 *Assume $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ is a family of circuits over $\mathcal{K}$ such that:*

- *$\mathscr{C}_n$ has $n+m$ input gates $(x_1,\ldots,x_n,y_1,\ldots,y_m)$,*
- *there exists a function $t:\mathbb{N}\to\mathbb{N}$ such that, for all $n\in\mathbb{N}$, $|\mathscr{C}_n|\le t(n)$ and $t(n)\ge n$*
- *there exists a partial recursive function T such that $T(\overline{x}) = \mathbf{1}^{t(|\overline{x}|)}$*
- *there exist partial recursive functions $\mathsf{Gate}, F_1,\ldots F_r$ (here r is the maximum arity of the symbols of $\mathcal{K}$) such that $\mathsf{Gate}(T(\overline{x}),\mathbf{1}^i,\overline{x})$ describes the i^{th} node of $\mathscr{C}_{|\overline{x}|}$ as follows*

$$\mathsf{Gate}(T(\overline{x}),\mathbf{1}^i,\overline{x}) = \begin{cases} \mathbf{0} & \text{if } i \text{ is an input gate} \\ \mathbf{0.0} & \text{if } i \text{ is an output gate} \\ \mathbf{1}^j & \text{if } i \text{ is a gate labeled with } op_j \\ \mathbf{1}^{k+j} & \text{if } i \text{ is a gate labeled with } rel_j \\ \mathbf{1}^{k+l+1} & \text{if } i \text{ is a selection node} \\ \epsilon & \text{otherwise} \end{cases}$$

and $F_j(T(\overline{x}),\mathbf{1}^i,\overline{x})$ identifies the j^{th} parent node of the i^{th} node of $\mathscr{C}_{|\overline{x}|}$ as follows

$$F_j(T(\overline{x}),\mathbf{1}^i,\overline{x}) = \mathbf{1}^k \text{ where } k\le i \text{ is the } j^{th} \text{ parent node of } i.$$

Then, given a set of constants $\overline{\alpha}=\alpha_1,\ldots,\alpha_m\in\mathbb{K}^m$, there exists a partial recursive function $C^*_{\overline{\alpha}}$ over $\mathcal{K}$ such that, for any $\overline{x}=x_1.\ldots.x_n$, $C^*_{\overline{\alpha}}(\overline{x})$ equals $\mathscr{C}_n(x_1,\ldots,x_n,\alpha_1,\ldots,\alpha_m)$.

PROOF. It is easy to define a partial recursive function Pick such that, for any $\overline{z}, \overline{t}, \overline{x} \in \mathbb{K}^*$ satisfying $|\overline{t}| \le |\overline{x}| \le |\overline{z}|$,

$$\mathsf{Pick}(\overline{z}, \overline{t}, \overline{x}) = x_{n+1-|\overline{t}|}$$

where $\overline{x} = x_1 . \dots . x_n$. The values of the x_i are irrelevant. We only require $|\overline{x}| \le |\overline{z}|$.

We next want to define a partial recursive function $V_{\overline{\alpha}}$ which, on an input $(\overline{z}, \overline{x})$, computes the evaluation of all gates numbered from 1 to $|\overline{z}|$ of $\mathscr{C}_{|\overline{x}|}$ on input $(\overline{x}, \overline{\alpha})$ and concatenates the results in one word. For the sake of readability, we denote by $\mathcal{F}_p$ the expression $\mathsf{Pick}(T(\overline{x}), F_p(T(\overline{x}), 1.\overline{z}, \overline{x}), V_{\overline{\alpha}}(\overline{z}, \overline{x}))$. This expression gives the evaluation of the pth parent gate of the current gate (which is the one numbered by $|\overline{z}| + 1$). It is obtained by "Picking" it at the right position in the recurrence argument. Denote by $k(i)$ the arity of op_i and by $l(i)$ the arity of rel_i. We may define $V_{\overline{\alpha}}$ as follows (we use definition by cases which can be easily described by combining Select operators and simple primitive recursive functions) where $G = \mathsf{Gate}(T(\overline{x}), \mathbf{1}.\overline{z}, \overline{x})$ is the type of the current gate,

$$\begin{array}{rcl}
V_{\overline{\alpha}}(\epsilon, \overline{x}) & = & \epsilon \\
V_{\overline{\alpha}}(c.\overline{z}, \overline{x}) & = & \left\{ \begin{array}{ll}
\mathsf{cons}(x_i, V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{0} \text{ and } |\overline{z}| + 1 = i \le |\overline{x}| \\
\mathsf{cons}(\alpha_{i-|\overline{x}|}, V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{0} \\
 & \text{and } |\overline{z}| + 1 = i \in [|\overline{x}| + 1, |\overline{x}| + m] \\
\mathsf{cons}(\mathrm{op}_j(\mathcal{F}_1, \dots, \mathcal{F}_{k(j)}), V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{1}^j \\
\mathsf{cons}(\mathrm{rel}_j(\mathcal{F}_1, \dots, \mathcal{F}_{l(j)}), V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{1}^{k+j} \\
\mathsf{cons}(\mathsf{Select}(\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3), V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{1}^{k+l+1} \\
\mathsf{cons}(\mathcal{F}_1, V_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } G = \mathbf{0.0} \\
V_{\overline{\alpha}}(\overline{z}, \overline{x}) & \text{otherwise.}
\end{array} \right.
\end{array}$$

Note that the six first cases in the definition above correspond to the current node being input, constant, operator, relation, selection and output, respectively. The seventh case is to deal with unexpected values of $\overline{z}$, e.g., a too large gate number. We define now a function $R_{\overline{\alpha}}$ which concatenates the values returned by the outputs gates of $\mathscr{C}_{|\overline{x}|}$,

$$\begin{array}{rcl}
R_{\overline{\alpha}}(\epsilon, \overline{x}) & = & \epsilon \\
R_{\overline{\alpha}}(c.\overline{z}, \overline{x}) & = & \left\{ \begin{array}{ll}
\mathsf{cons}(V_{\overline{\alpha}}(\overline{z}, \overline{x}), R_{\overline{\alpha}}(\overline{z}, \overline{x})) & \text{if } \mathsf{Gate}(T(\overline{x}), \overline{z}, \overline{x}) = \mathbf{0.0} \\
R_{\overline{\alpha}}(\overline{z}, \overline{x}) & \text{otherwise.}
\end{array} \right.
\end{array}$$

The result of the evaluation of $\mathscr{C}_{|\overline{x}|}$ on input $(\overline{x}, \overline{\alpha})$ is given by $R_{\overline{\alpha}}(T(\overline{x}), \overline{x})$.

□

5.3.2 Proof of Theorem 5.2.1

Assume M is a BSS machine over $\mathcal{K}$ computing a function $f_M : \mathbb{K}^* \to \mathbb{K}^*$. Denote by $\alpha_1, \dots, \alpha_m \in \mathbb{K}$ the constants used by M. A description of the configuration reached by M

on input $\overline{x}$ after t iterations can be given by the values of the following four functions:

- $\mathsf{Node}(\overline{x}, \mathbf{0}^t)$, which gives a encoding for the node of M reached after t steps. We assume without loss of generality that the encoding of all output nodes begins with $\mathbf{1}$ and that the encoding of all other nodes begins with $\mathbf{0}$.

- $\mathsf{Length}(\overline{x}, \mathbf{0}^t) = \mathbf{1}^k.\mathbf{0}^{t+|\overline{x}|-k}$, which gives the actual length k of the non-empty part of the tape of M after t steps (the $\mathbf{0}$s are for padding the output so that its length only depends on the length of the input)

- $\mathsf{Tape}(\overline{x}, \mathbf{0}^t) = \overline{y}.\mathbf{0}^{t+|\overline{x}|-k}$, which gives the content $\overline{y}$ of the tape of M after t steps (the $\mathbf{0}$s are for padding as above)

- $\mathsf{Head}(\overline{x}, \mathbf{0}^t)$ which describes the position of the head of M after t steps.

By construction, the length of the output of these four functions depends only on the size of their input. The existence of a universal BSS machine ensures that these functions are BSS computable and, by appropriately clocking the machines computing them, we may assume that the computation time of these machines depends only on the size of their input. Therefore, we can apply Proposition 1.2.1 to deduce the existence of four families $\{\mathcal{N}_{n,t}, \mathcal{L}_{n,t}, \mathcal{T}_{n,t}, \mathcal{H}_{n,t} \mid n, t \in \mathbb{N}\}$ of circuits, each of them describable in time $T(n,t)$, polynomial in $n+t$, by a deterministic Turing machine. Using a classical result [Kle36] these machines can be simulated by classical partial recursive functions, which are also partial recursive functions over $\mathcal{K}$ by Remark 1.1.1 and Proposition 5.2.1. Without loss of generality, one can assume that the descriptions of the circuits above have the form given in the hypotheses of Lemma 5.3.1. We can then apply this lemma to conclude that the functions Node, Length, Tape and Head are partial recursive over $\mathcal{K}$. From Length and Tape, it is trivial to build a partial recursive function Result such that $\mathsf{Result}(\overline{x}, \mathbf{1}^t)$ gives exactly the non-empty part of the tape of M on input $\overline{x}$ after t steps. The computation time of M on input $\overline{x}$ is then given by

$$\mathsf{Time}(\overline{x}) = \mu\overline{b}(\mathsf{Node}(\overline{x}, \overline{b}))$$

and the result of this computation by

$$f_M(\overline{x}) = \mathsf{Result}(\overline{x}, \mathsf{Time}(\overline{x})).$$

The computation of M can therefore be simulated by a partial recursive function over $\mathcal{K}$.

As before, we do not detail the simulation of a partial recursive function by a BSS machine.

□

Chapter 6

Safe Recursion and Poly-Time

In this chapter, we use our notion of partial recursive and primitive recursive functions to extend the approach of Bellantoni and Cook [BC92], in order to capture the class $\mathrm{FP}_{\mathcal{K}}$ of functions computable in polynomial time by a BSS machine over $\mathcal{K}$. These results are published in [BCdNM02, BCdNM03a, BCdNM04a].

6.1 Safe Recursive Functions

In this section we extend to an arbitrary structure $\mathcal{K}$ the notion of safe recursive function over the natural numbers defined by Bellantoni and Cook [BC92].

Example 6.1.1 Consider the following function

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

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

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

On the other hand, note that $\mathsf{exp} \notin \mathrm{FP}_{\mathcal{K}}$ 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 [BC92], 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. 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* which are the basic functions of Definition 5.2.1 with the feature that their arguments are all safe.

Definition 6.1.1 (Safe Recursive Functions)

The set of *safe recursive functions* over $\mathcal{K}$, denoted as $\mathrm{SR}_{\mathcal{K}}$, 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:

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

Remark 6.1.1 (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 6.1.1 into a safe recursion scheme. This would need the use of the recurrence argument $\mathsf{exp}(\overline{x})$ as a normal argument of function Cons, which is forbidden. This obstruction is at the basis of the equivalence between safe recursion and polynomial time computability.

The following result is immediate.

Proposition 6.1.1 *The set of safe recursive functions over* $\mathbb{Z}_2 = \{\{0, 1\}, =, \mathbf{0}, \mathbf{1}\}$ *coincides with the classical set of safe recursive functions defined by Bellantoni and Cook in [BC92].*

□

Definition 6.1.2 (Safe Recursive Functions Relative to ϕ)
Given a function $\phi : \emptyset \times \mathbb{K}^* \to \{\mathbf{0}, \mathbf{1}\}$, the set of *safe recursive functions relative to* ϕ over $\mathcal{K}$, denoted by $\mathrm{SR}_{\mathcal{K}}(\phi)$, is the smallest set of functions $f : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to \mathbb{K}^*$ containing the basic safe functions and ϕ, and closed under safe composition and safe recursion.

Remark 6.1.2 Note that we do only consider functions ϕ with values in $\{\mathbf{0}, \mathbf{1}\}$. Abusing notations, we will freely identify ϕ with $\{\overline{x} \in \mathbb{K}^* : \phi(; \overline{x}) = \mathbf{1}\}$, and consider machines with oracle to this set, which we denote as machines with oracle ϕ.

6.2 BSS PolyTime Computability and Safe Recursion

Before introducing our first machine independent characterizations of complexity classes over an arbitrary structure, we introduce a secondary class of functions by a simultaneous safe recursion scheme.

6.2.1 Simultaneous Safe Recursion

Definition 6.2.1 (Simultaneous Safe Recursive Functions Relative to ϕ)
Given a function $\phi : \emptyset \times \mathbb{K}^* \to \mathbb{K}^*$, the set of *simultaneous safe recursive functions relative to* ϕ over $\mathcal{K}$, denoted by $\mathrm{SSR}_{\mathcal{K}}(\phi)$, is the smallest set of functions $f : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to$

$\mathbb{K}^*$ containing the basic safe functions and ϕ, and closed under safe composition and the following simultaneous safe recursion scheme:

$$\begin{cases} f_1(\epsilon, \overline{x}; \overline{y}) & = & h_1(\overline{x}; \overline{y}) \\ f_2(\epsilon, \overline{x}; \overline{y}) & = & h_2(\overline{x}; \overline{y}) \\ f_1(a.\overline{z}, \overline{x}; \overline{y}) & = & g_1(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \overline{y}), f_2(\overline{z}, \overline{x}; \overline{y}), \overline{y}) \\ f_2(a.\overline{z}, \overline{x}; \overline{y}) & = & g_2(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \overline{y}), f_2(\overline{z}, \overline{x}; \overline{y}), \overline{y}) \end{cases}$$

Remark 6.2.1 Simultaneous safe recursion is defined here as a double recursion scheme. It is clear that a definition with a larger number of inter-dependent functions would yield the same class of functions.

6.2.2 Polynomial Time Evaluation of $\mathbf{SSR}_{\mathcal{K}}(\phi)$

In this section we show the polynomial time evaluation of any function in $\mathrm{SSR}_{\mathcal{K}}(\phi)$. In order to simplify notations, we give the proof for $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$. The proof is the same for higher arities (i.e., $f : (\mathbb{K}^*)^p \times (\mathbb{K}^*)^q \to \mathbb{K}^*$) and simultaneous recursion.

Given $\phi : \emptyset \times \mathbb{K}^* \to \{\mathbf{0}, \mathbf{1}\}^*$ a function over $\mathcal{K}$, and $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ a function in $\mathrm{SSR}_{\mathcal{K}}(\phi)$, we denote by $T[f(\overline{x}; \overline{y})]$ the time necessary to compute $f(\overline{x}; \overline{y})$ with a BSS machine over $\mathcal{K}$ with oracle ϕ.

Proposition 6.2.1 *Let $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ be a function in $SSR_{\mathcal{K}}(\phi)$. There exists a polynomial p_f such that, for all $(\overline{x}, \overline{y}) \in \mathbb{K}^* \times \mathbb{K}^*$,*

$$T[f(\overline{x}; \overline{y})] \leq p_f(|\overline{x}|).$$

Proof. By induction on the depth of the definition tree of f.

If f is a basic safe function, it can be evaluated in constant time. Therefore, the statement is true for basic safe functions.

If $f = \phi$, querying the oracle $\phi(; \overline{x}) = \mathbf{1}$ allows to compute $\phi(; \overline{x})$ at cost one by Remark 6.1.2, since, by hypothesis, $\phi(; \overline{x}) \in \{\mathbf{0}, \mathbf{1}\}$.

Assume now f is defined by safe composition from safe recursive functions $g, h_1, \ldots, h_{m+n}$ be as in Definition 6.1.1. Then, there exists polynomials $p_g, p_1, \ldots, p_{m+n}$ satisfying the statement for these functions respectively. Without loss of generality, we may assume these polynomials to be nondecreasing. Therefore,

$$\begin{aligned} T[f(\overline{x}; \overline{y})] &\leq T[h_1(\overline{x}; \overline{y})] + \cdots + T[h_{m+n}(\overline{x}; \overline{y})] \\ &\quad + T[g(h_1(\overline{x}; \overline{y}), \ldots, h_m(\overline{x}; \overline{y}); h_{m+1}(\overline{x}; \overline{y}), \ldots, h_{m+n}(\overline{x}; \overline{y}))] \\ &\leq p_1(|\overline{x}|) + \cdots + p_{m+n}(|\overline{x}|) + p_g(p_1(|\overline{x}|) + \cdots + p_m(|\overline{x}|)) \end{aligned}$$

which shows we may take

$$p_f = p_1 + \cdots + p_{m+n} + p_g \circ (p_1 + \cdots + p_m).$$

Finally, assume f is defined by safe recursion and let h, g be as in Definition 6.1.1. Then, there exist polynomials p_h, p_g satisfying the statement for these functions respectively. In addition, we may also assume that p_g and p_h are nondecreasing. Therefore,

$$T[f(\epsilon, \overline{x}; \overline{y})] = p_h(|\overline{x}|)$$

and, when unfolding the recurrence,

$$\begin{aligned} T[f(a.\overline{z}, \overline{x}; \overline{y})] &\leq T[f(\overline{z}, \overline{x}; \overline{y})] + T[g(\overline{z}, \overline{x}; f(\overline{z}, \overline{x}; \overline{y}), \overline{y})] \\ &\leq T[f(\overline{z}, \overline{x}; \overline{y})] + p_g(|\overline{z}, \overline{x}|) \\ &\vdots \\ &\leq |\overline{z}| p_g(|\overline{z}, \overline{x}|) + p_h(|\overline{x}|) \end{aligned} \tag{6.1}$$

which shows we may take

$$p_f = \text{Id } p_g + p_h.$$

□

Remark 6.2.2 **(i)** Note that in (6.1) above the evaluation time $T[g(\overline{z}, \overline{x}; f(\overline{z}, \overline{x}; \overline{y}), \overline{y})]$, because of the induction hypothesis, does not depend on $f(\overline{z}, \overline{x}; \overline{y})$. This independence in the inductive argument in the proof is the key that keeps the evaluation time polynomially bounded.

(ii) A similar result could be obtained by using the notion of "tier" introduced in [Lei94a].

6.2.3 Safe Recursion Captures Poly-time Computability

This section is devoted to the proof of the fact that a function computable in deterministic polynomial time can be defined by safe recursion. In a first, we show that it holds for simultaneous safe recursion, and in a second step we show the equivalence between simultaneous and non-simultaneous safe recursion.

Simultaneous Safe Recursion Captures $\mathrm{FP}^{\phi}_{\mathcal{K}}$

We prove first the following lemma:

Lemma 6.2.1 *For any $c, d \in \mathbb{N}$, one can write a safe recursive function P_{cd} such that, on any input $\overline{x}$ with $|\overline{x}| = n$, $|P_{cd}(\overline{x};)| = cn^d$.*

PROOF.

We define the following functions:

$$\begin{array}{rcl}
C_1\,(a_1.\overline{z_1},\ldots,\overline{z_{d+1}};\overline{c_0}) & = & C_2\,(\overline{z_2},\ldots,\overline{z_{d+1}};C_1\,(\overline{z_1},\ldots,\overline{z_{d+1}};\overline{c_0})) \\
C_1\,(\epsilon,\overline{z_2},\ldots,\overline{z_{d+1}};\overline{c_0}) & = & \overline{c_0} \\
C_2\,(a_2.\overline{z_2},\ldots,\overline{z_{d+1}};\overline{c_1}) & = & C_3\,(\overline{z_3},\ldots,\overline{z_{d+1}};C_2\,(\overline{z_2},\ldots,\overline{z_{d+1}};\overline{c_1})) \\
C_2\,(\epsilon,\overline{z_3},\ldots,\overline{z_{d+1}};\overline{c_1}) & = & \overline{c_1} \\
\vdots & & \vdots \\
C_d\,(a_d.\overline{z_d},\overline{z_{d+1}};\overline{c_{d-1}}) & = & C_{d+1}\,(\overline{z_{d+1}};C_d\,(\overline{z_d},\overline{z_{d+1}};\overline{c_{d-1}})) \\
C_d\,(\epsilon,\overline{z_{d+1}};\overline{c_{d-1}}) & = & \overline{c_{d-1}} \\
C_{d+1}\,(a_{d+1}.\overline{z_{d+1}};\overline{c_d}) & = & \mathsf{cons}\,(;\mathbf{1},C_{d+1}\,(\overline{z_{d+1}};\overline{c_d})) \\
C_{d+1}\,(\epsilon;\overline{c_d}) & = & \overline{c_d}
\end{array}$$

Let us show the following: for any $i \in \{1,\ldots,d+1\}$:

$$|C_i\,(\overline{z_i},\ldots,\overline{z_{d+1}};\overline{c_{i-1}})\,| = |\overline{z_i}|.\ldots.|\overline{z_{d+1}}| + |\overline{c_{i-1}}|.$$

This is obviously true for $d+1-i=0$, i.e. $i = d+1$.

If we suppose it true for $i+1$, then:

$$\begin{array}{rcl}
|C_i\,(a_i.\overline{z_i},\ldots,\overline{z_{d+1}};\overline{c_{i-1}})\,| & = & |C_{i+1}\,(\overline{z_{i+1}},\ldots,\overline{z_d};C_i\,(\overline{z_i},\ldots,\overline{z_{d+1}};\overline{c_{i-1}})) \\
 & = & |\overline{z_{i+1}}|.\ldots.|\overline{z_{d+1}}| + |C_i\,(\overline{z_i},\ldots,\overline{z_{d+1}};\overline{c_{i-1}})\,| \\
|C_i\,(\epsilon,\ldots,\overline{z_{d+1}};\overline{c_{i-1}})\,| & = & 0 + |\overline{c_{i-1}}|.
\end{array}$$

This gives the result by induction on the length of $\overline{z_i}$, therefore the property holds for i. The end of the proof is given by the following definition:

$$P_{cd}(\overline{x}) = C_1(\mathbf{1}^c,\overline{x},\ldots,\overline{x};\epsilon).$$

□

We are now able to prove the following intermediate result:

Proposition 6.2.2 *Let $\Phi \in \mathbb{K}^*$ be a decision problem over $\mathcal{K}$, and denote by $\phi : \emptyset \times \mathbb{K}^* \to \{\mathbf{0}, \mathbf{1}\}$ its characteristic function. Then, a function is in the class $\mathrm{FP}^{\Phi}_{\mathcal{K}}$ of functions computable in polynomial time with oracle Φ if and only if it can be defined in $SSR_{\mathcal{K}}(\phi)$.*

Proof. Assume M is a machine over $\mathcal{K}$ with oracle ϕ, with input $\overline{x} \in \mathbb{K}^*$. Assume also that the computation time of M is bounded by $P_{cd}(|\overline{x}|) = c|\overline{x}|^d, \quad c, d \in \mathbb{N}$. Without loss of generality, assume that, when halting, the head of the machine is on the first cell of the tape: all cells on the left part of the tape contain the blank symbol "#".

The rest of the proof is very similar to the proof of Theorem 5.2.1 in Section 5.2.2. The simulation of the machine is essentially the same; it uses safe recursive functions instead of primitive recursive functions, and oracle nodes are also dealt with. The halting criterion for getting the result of the computation also differs.

In what follows, as in the proof of Theorem 5.2.1, we represent the contents of the tape of M by a pair of variables $(\overline{x}, \overline{y})$ in $(\mathbb{K}^*)^2$ such that the written part of the tape is given by $\overline{x}^R.\overline{y}$ (here $\overline{x}^R$ is the reversed word of $\overline{x}$) and the head of the machine is on the first letter of $\overline{y}$.

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. In the following definitions, node number q will be coded by the variable (word) $\mathbf{1}^q$ of length q.

Let q ($q \in \mathbb{N}$) be a move node as in Figure 6.1.

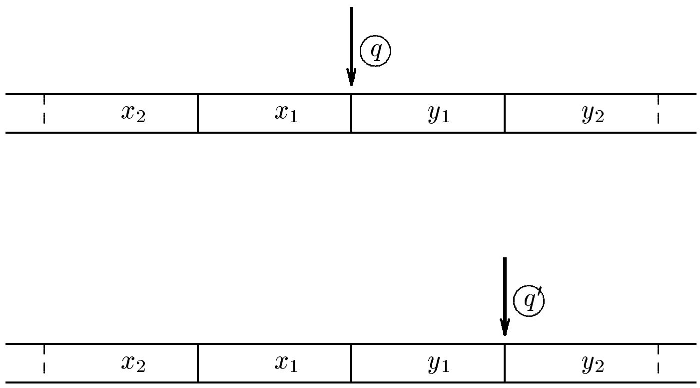

Figure 6.1: A "move right" Node

Three functions are associated with this node:

$$\begin{array}{lll} \mathcal{G}_i(;\overline{x},\overline{y}) & = \mathbf{1}^{q'} & \\ \mathcal{H}_i(;\overline{x},\overline{y}) & = \mathsf{tl}(;\overline{x}) & \text{or } \mathsf{hd}(;\overline{y}).\overline{x} \\ \mathcal{I}_i(;\overline{x},\overline{y}) & = \mathsf{hd}(;\overline{x}).\overline{y} & \text{or } \mathsf{tl}(;\overline{y}) \end{array}$$

according if one moves left or right.

Function $\mathcal{G}_i$ returns the encoding of the following node in the computation tree of M, function $\mathcal{H}_i$ returns the encoding of the left part of the tape, and function $\mathcal{I}_i$ returns the encoding of the right part of the tape.

Let q ($q \in \mathbb{N}$) be a computation node associated to some operation op of arity n of the structure, as in Figure 6.2. We also write Op for the corresponding basic operation.

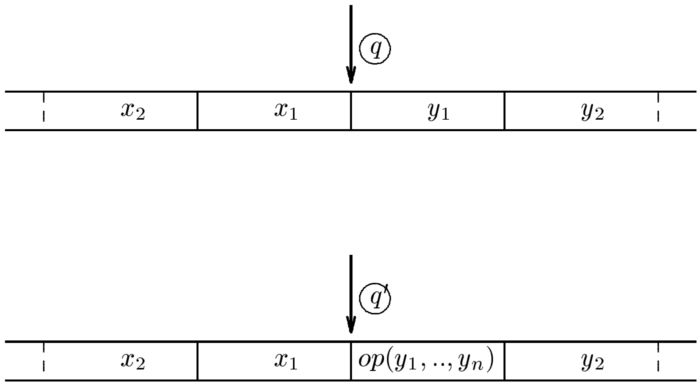

Figure 6.2: An “op” Node

Functions $\mathcal{G}_i$, $\mathcal{H}_i$, $\mathcal{I}_i$ associated with this node are now defined as follows:

$$\begin{array}{ll} \mathcal{G}_i(;\overline{x},\overline{y}) & = \mathbf{1}^{q'} \\ \mathcal{H}_i(;\overline{x},\overline{y}) & = \overline{x} \\ \mathcal{I}_i(;\overline{x},\overline{y}) & = \mathsf{cons}(;\mathsf{Op}(;\mathsf{hd}(;\overline{y}),\ldots,\mathsf{hd}(;\mathrm{tl}^{(n-1)}(;\overline{y}))),\mathsf{tl}(;\overline{y})). \end{array}$$

Let q ($q \in \mathbb{N}$) be a copy node associated to some operation op of arity n of the structure, as in Figure 6.3.

Functions $\mathcal{G}_i$, $\mathcal{H}_i$, $\mathcal{I}_i$ associated with this node are now defined as follows:

$$\begin{array}{lcl} \mathcal{G}_i(;\overline{x},\overline{y}) & = & \mathbf{1}^{q'} \\ \mathcal{H}_i(;\overline{x},\overline{y}) & = & \overline{x} \\ \mathcal{I}_i(;\overline{x},\overline{y}) & = & \left\{ \begin{array}{ll} \mathsf{cons}(;\mathsf{hd}(;\mathsf{tl}(;\overline{x})),\mathsf{tl}(;\overline{x})) & \text{for a copy left} \\ \mathsf{cons}(;\mathsf{hd}(;\overline{x}),\mathsf{cons}(;\mathsf{hd}(;\overline{x}),\mathsf{tl}(;\mathsf{tl}(;\overline{x})))) & \text{for a copy right} \\ \mathsf{cons}(;\mathsf{hd}(;\mathsf{tl}(;\overline{x})),\mathsf{cons}(;\mathsf{hd}(;\overline{x}),\mathsf{tl}(;\mathsf{tl}(;\overline{x})))) & \text{for a switch.} \end{array} \right. \end{array}$$

Let q ($q \in \mathbb{N}$) be a branch node corresponding to a relation rel of arity n of the structure, as in Figure 6.4.

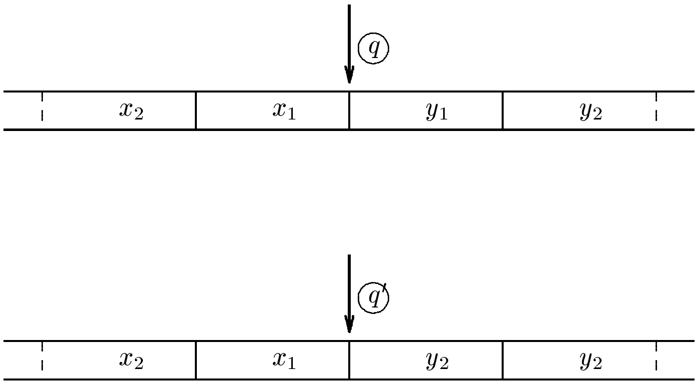

Figure 6.3: A "copy left" Node

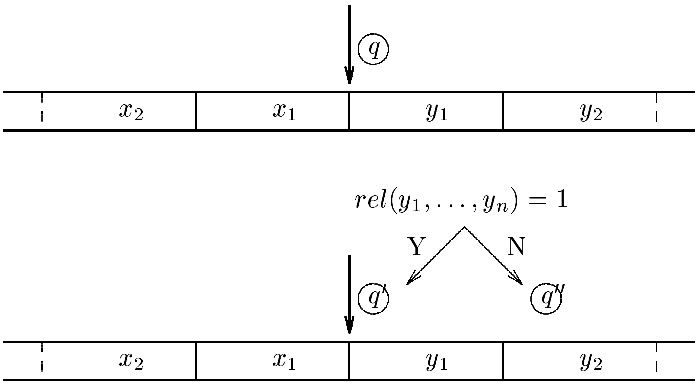

Figure 6.4: A "rel" Node

The three functions associated with this node are now:

$$
\begin{aligned}
\mathcal{G}_i(;\overline{x},\overline{y}) &= \mathsf{Select}(;\mathsf{Rel}(;\mathsf{hd}(;\overline{x}),\ldots,\mathsf{hd}(;\mathsf{tl}^{(n-1)}(;\overline{y}))),\mathbf{1}^{q'},\mathbf{1}^{q''})\\
\mathcal{H}_i(;\overline{x},\overline{y}) &= \overline{x}\\
\mathcal{I}_i(;\overline{x},\overline{y}) &= \overline{y}.
\end{aligned}
$$

Let q ($q \in \mathbb{N}$) be an oracle node calling for the oracle function ϕ, as in Figure 6.5.

The three functions associated with this node are now:

$$
\begin{aligned}
\mathcal{G}_i(;\overline{x},\overline{y}) &= \mathbf{1}^{q'}\\
\mathcal{H}_i(;\overline{x},\overline{y}) &= \overline{x}\\
\mathcal{I}_i(;\overline{x},\overline{y}) &= \mathsf{cons}(;\phi(;\overline{y}),\mathsf{tl}(;\overline{y})).
\end{aligned}
$$

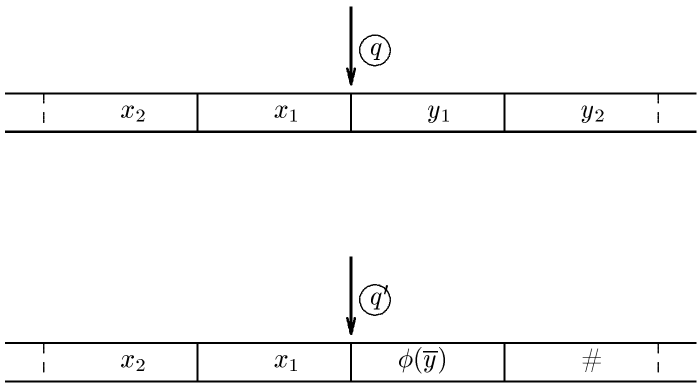

Figure 6.5: An Oracle Node

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

$$\mathsf{Equal}_k(;\overline{x}) = \begin{cases} \mathbf{1} & \text{if } \overline{x} = \mathbf{1}^k \\ \epsilon & \text{otherwise.} \end{cases}$$

We can now define the safe recursive functions $\mathsf{next}_{\mathsf{state}}$, $\mathsf{next}_{\mathsf{left}}$ and $\mathsf{next}_{\mathsf{right}}$ 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:

$$\begin{aligned}
\mathsf{next}_{\mathsf{state}}(;\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(;\mathsf{Equal}_0(;\overline{s}),\mathcal{G}_0(;\overline{x},\overline{y}),\mathsf{Select}\,(;\mathsf{Equal}_1(;\overline{s}),\mathcal{G}_1(;\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(;\mathsf{Equal}_m(;\overline{s}),\mathcal{G}_m(;\overline{x},\overline{y}),\epsilon)\ldots))\,,\\
\mathsf{next}_{\mathsf{left}}(;\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(;\mathsf{Equal}_0(;\overline{s}),\mathcal{H}_0(;\overline{x},\overline{y}),\mathsf{Select}\,(;\mathsf{Equal}_1(\overline{s}),\mathcal{H}_1(;\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(;\mathsf{Equal}_m(;\overline{s}),\mathcal{H}_m(;\overline{x},\overline{y}),\epsilon)\ldots))
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{next}_{\mathsf{right}}(;\overline{s},\overline{x},\overline{y}) &= \mathsf{Select}\,(;\mathsf{Equal}_0(;\overline{s}),\mathcal{I}_0(;\overline{x},\overline{y}),\mathsf{Select}\,(;\mathsf{Equal}_1(;\overline{s}),\mathcal{I}_1(;\overline{x},\overline{y}),\\
&\quad \ldots\mathsf{Select}\,(;\mathsf{Equal}_m(;\overline{s}),\mathcal{I}_m(;\overline{s},\overline{x},\overline{y}),\epsilon)\ldots))\,.
\end{aligned}$$

From now on, we define with simultaneous safe recursion the encoding of the state of the machine reached after k computation nodes, where k is the length of the word $\overline{t} \in \mathbb{K}^*$, and we also define the encoding of the left part and the right part of the tape. All this is done with functions $\mathsf{comp}_{\mathsf{state}}$, $\mathsf{comp}_{\mathsf{left}}$ and $\mathsf{comp}_{\mathsf{right}}$ defined with simultaneous safe recursion as follows:

$$\mathsf{comp}_{\mathsf{state}}(\epsilon;\overline{x},\overline{y}),\mathsf{comp}_{\mathsf{left}}(\epsilon;\overline{x},\overline{y}),\mathsf{comp}_{\mathsf{right}}(\epsilon;\overline{x},\overline{y}) = \epsilon,\overline{x},\overline{y}$$

and

$$\begin{aligned}
\mathsf{comp}_{\mathsf{state}}(a.\overline{t};\overline{x},\overline{y}) &= \mathsf{next}_{\mathsf{state}}\left(;\mathsf{comp}_{\mathsf{state}}(\overline{t};\overline{x},\overline{y}),\mathsf{comp}_{\mathsf{left}}(\overline{t};\overline{x},\overline{y}),\right.\\
&\qquad \left.\mathsf{comp}_{\mathsf{right}}(\overline{t};\overline{x},\overline{y})\right)\\
\mathsf{comp}_{\mathsf{left}}(a.\overline{t};\overline{x},\overline{y}) &= \mathsf{next}_{\mathsf{left}}\left(;\mathsf{comp}_{\mathsf{state}}(\overline{t};\overline{x},\overline{y}),\mathsf{comp}_{\mathsf{left}}(\overline{t};\overline{x},\overline{y}),\right.\\
&\qquad \left.\mathsf{comp}_{\mathsf{right}}(\overline{t};\overline{x},\overline{y})\right)\\
\mathsf{comp}_{\mathsf{right}}(a.\overline{t};\overline{x},\overline{y}) &= \mathsf{next}_{\mathsf{right}}\left(;\mathsf{comp}_{\mathsf{state}}(\overline{t};\overline{x},\overline{y}),\mathsf{comp}_{\mathsf{left}}(\overline{t};\overline{x},\overline{y}),\right.\\
&\qquad \left.\mathsf{comp}_{\mathsf{right}}(\overline{t};\overline{x},\overline{y})\right).
\end{aligned}$$

After k computation steps, on input $\overline{y} \in \mathbb{K}^*$, the content of the tape of M if given by $\mathsf{comp}_{\mathsf{right}}(\mathbf{1}^k;\epsilon,\overline{y})$. We apply also Lemma 6.2.1: There exists a safe recursive function P_{cd} over $\mathcal{K}$ such that $|P_{cd}(\overline{x};)| = c|\overline{x}|^d$.

The output of M on input $\overline{y} \in \mathbb{K}^*$ is then given in $\mathrm{SSR}_{\mathcal{K}}(\phi)$ by

$$\mathsf{comp}_{\mathsf{right}}(P_{cd}(\overline{x};);\epsilon,\overline{y}).$$

The other direction of the proof follows from Proposition 6.2.1

□

We now move on to the second step of the proof, showing the equivalence between simultaneous and non-simultaneous safe recursion.

Proving $\mathbf{SSR}_{\mathcal{K}}(\phi) = \mathbf{SR}_{\mathcal{K}}(\phi)$

Definition 6.2.2 (Homogeneous Length Hypothesis)

Let $g_1, g_2, h_1, h_2 \in \mathrm{SSR}_{\mathcal{K}}(\phi)$, and let $f_1, f_2 \in \mathrm{SSR}_{\mathcal{K}}(\phi)$ be defined a simultaneous safe recursion scheme as follows:

$$\begin{cases}
f_1(\epsilon,\overline{x};\overline{y}) &= h_1(\overline{x};\overline{y})\\
f_2(\epsilon,\overline{x};\overline{y}) &= h_2(\overline{x};\overline{y})\\
f_1(a.\overline{z},\overline{x};\overline{y}) &= g_1(\overline{z},\overline{x};f_1(\overline{z},\overline{x};\overline{y}),f_2(\overline{z},\overline{x};\overline{y}),\overline{y})\\
f_2(a.\overline{z},\overline{x};\overline{y}) &= g_2(\overline{z},\overline{x};f_1(\overline{z},\overline{x};\overline{y}),f_2(\overline{z},\overline{x};\overline{y}),\overline{y})
\end{cases}$$

Functions h_1, h_2, g_1, g_2 respect the *homogeneous length hypothesis* if and only if there exists a function $L : (\mathbb{K}^*)^2 \to \{\mathbf{0}\}^* \in \mathrm{SR}_{\mathcal{K}}$ such that:

$$\forall \overline{z},\overline{x},\overline{r_1},\overline{r_2}\,\overline{y} \in \mathbb{K}^*$$

$$\begin{aligned}
|g_1(\overline{z},\overline{x};\overline{r_1},\overline{r_2},\overline{y})| &= |L(\overline{z},\overline{x};)|\\
|g_2(\overline{z},\overline{x};\overline{r_1},\overline{r_2},\overline{y})| &= |L(\overline{z},\overline{x};)|\\
|h_1(\overline{x};\overline{y})| &= |L(\epsilon,\overline{x};)|\\
|h_2(\overline{x};\overline{y})| &= |L(\epsilon,\overline{x};)|
\end{aligned}$$

Lemma 6.2.2 *For any functions $f_1, f_2 : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ in $SSR_{\mathcal{K}}(\phi)$, with*

$$\begin{cases} f_1(\epsilon, \overline{x}; \overline{y}) & = & h_1(\overline{x}; \overline{y}) \\ f_2(\epsilon, \overline{x}; \overline{y}) & = & h_2(\overline{x}; \overline{y}) \\ f_1(a.\overline{z}, \overline{x}; \overline{y}) & = & g_1(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \overline{y}), f_2(\overline{z}, \overline{x}; \overline{y}), \overline{y}) \\ f_2(a.\overline{z}, \overline{x}; \overline{y}) & = & g_2(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \overline{y}), f_2(\overline{z}, \overline{x}; \overline{y}), \overline{y}), \end{cases}$$

and $g_1, g_2, h_1, h_2 \in SR_{\mathcal{K}}(\phi)$, there exist functions $f_1', f_2' : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ in $SSR_{\mathcal{K}}(\phi)$, with

$$\begin{cases} f_1'(\epsilon, \overline{x}; \overline{y}) & = & h_1'(\overline{x}; \overline{y}) \\ f_2'(\epsilon, \overline{x}; \overline{y}) & = & h_2'(\overline{x}; \overline{y}) \\ f_1'(a.\overline{z}, \overline{x}; \overline{y}) & = & g_1'(\overline{z}, \overline{x}; f_1'(\overline{z}, \overline{x}; \overline{y}), f_2'(\overline{z}, \overline{x}; \overline{y}), \overline{y}) \\ f_2'(a.\overline{z}, \overline{x}; \overline{y}) & = & g_2'(\overline{z}, \overline{x}; f_1'(\overline{z}, \overline{x}; \overline{y}), f_2'(\overline{z}, \overline{x}; \overline{y}), \overline{y}), \end{cases}$$

$g_1', g_2', h_1', h_2' \in SR_{\mathcal{K}}(\phi)$, which respect the homogeneous length hypothesis, and $F \in SR_{\mathcal{K}}$ such that, for all $\overline{z}, \overline{x}, \overline{y} \in \mathbb{K}^*$, $F(\overline{z}, \overline{x}; f_1'(\overline{z}, \overline{x}; \overline{y})) = f_1(\overline{z}, \overline{x}; \overline{y})$ and $F(\overline{z}, \overline{x}; f_2'(\overline{z}, \overline{x}; \overline{y})) = f_2(\overline{z}, \overline{x}; \overline{y})$.

PROOF.

Consider $f_1, f_2 : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ as above.

Since g_1, g_2, h_1, h_2 are in $\mathrm{SR}_{\mathcal{K}}(\phi)$, by Proposition 6.2.1, there exists a function $B : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \{0\}^*$ in $\mathrm{SR}_{\mathcal{K}}$ such that:

$$\begin{array}{rcl} \forall \overline{z}, \overline{x}, \overline{r_1}, \overline{r_2} \overline{y} \in \mathbb{K}^* & & \\ |g_1(\overline{z}, \overline{x}; \overline{r_1}, \overline{r_2}, \overline{y})| & < & |B(\overline{z}, \overline{x};)| \\ |g_2(\overline{z}, \overline{x}; \overline{r_1}, \overline{r_2}, \overline{y})| & < & |B(\overline{z}, \overline{x};)| \\ |h_1(\overline{x}; \overline{y})| & < & |B(\epsilon, \overline{x};)| \\ |h_2(\overline{x}; \overline{y})| & < & |B(\epsilon, \overline{x};)| \end{array}$$

Let us define $C : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ such that:

$$C(\overline{b}; \overline{m}) = \mathbf{1}^{|\overline{m}|}.\mathbf{0}^{2(|\overline{b}| - |\overline{m}|)}.\overline{m}$$

and $C^{-1} : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ such that:

$$C^{-1}(\overline{b}; C(\overline{b}; \overline{m})) = \overline{m}$$

Note that $|C(\overline{b}; \overline{m})| = 2|\overline{b}|$.

These functions can be formally defined as follows:

$$
\begin{array}{rcl}
C(\overline{b};\overline{m}) & = & Cons(\overline{b};q''(\overline{b};\overline{m}),Cons(\overline{b};q'''(\overline{b};\overline{m}),Cons(\overline{b};q'''(\overline{b};\overline{m}),\overline{m})))\\
q''(\epsilon;\overline{m}) & = & \epsilon\\
q''(a.\overline{z};\overline{m}) & = & \left\{\begin{array}{lr} q''(\overline{z};\overline{m}) & \text{if } \mathrm{hd}(;Tl(\overline{z};\overline{m}))=\epsilon\\ \mathrm{cons}(;\mathbf{1},q''(\overline{z};\overline{m})) & \text{otherwise}\end{array}\right.\\
q'''(\epsilon,\overline{m}) & = & \epsilon\\
q'''(a.\overline{z};\overline{m}) & = & \left\{\begin{array}{lr} \mathrm{cons}(;\mathbf{0},q'''(\overline{z};\overline{m})) & \text{if } Tl(\overline{z};\overline{m})=\epsilon\\ q'''(\overline{z};\overline{m}) & \text{otherwise}\end{array}\right.\\
C^{-1}(\overline{b},\overline{m'}) & = & q'(\overline{b};T(\overline{b},\overline{b};\overline{m'}))\\
T(\epsilon,\overline{l};\overline{m'}) & = & \epsilon\\
T(a.\overline{z},\overline{l};\overline{m'}) & = & \left\{\begin{array}{lr} \mathrm{cons}(;\mathrm{hd}(;Tl(\overline{z};\overline{m'})),T(\overline{z},\overline{l};\overline{m'})) & \\ \qquad\qquad \text{if } \mathrm{hd}(;Tl(Sub(\overline{z},\overline{l};);\overline{m'}))=\mathbf{1} & \\ T(\overline{z},\overline{l};\overline{m'}) & \text{otherwise}\end{array}\right.\\
Sub(\overline{z},\overline{l};) & = & \mathrm{tl}(;Tl(\overline{z};\overline{l}))
\end{array}
$$

It is obvious that the matching cases in this definition can be formally defined with the test function Select. These functions are therefore in $\mathrm{SR}_{\mathcal{K}}(\phi)$.

Define now by simultaneous safe recursion:

$$
\begin{array}{rcl}
f_1''(\epsilon,\overline{x};\overline{y}) & = & C\left(B(\epsilon,\overline{x};);h_1(\overline{x};\overline{y})\right)\\
f_1''(a.\overline{z},\overline{x};\overline{y}) & = & C\left(B(\overline{z},\overline{x};);g_1\left(\overline{z},\overline{x};C^{-1}(B(\mathrm{tl'}(\overline{z};),\overline{x};)f_1''(\overline{z},\overline{x};\overline{y})),\right.\right.\\
 & & \left.\left.C^{-1}(B(\mathrm{tl'}(\overline{z};),\overline{x};)f_2''(\overline{z},\overline{x};\overline{y})),\overline{y}\right)\right)\\
f_2''(\epsilon,\overline{x};\overline{y}) & = & C\left(B(\epsilon,\overline{x};);h_2(\overline{x};\overline{y})\right)\\
f_1''(a.\overline{z},\overline{x};\overline{y}) & = & C\left(B(\overline{z},\overline{x};);g_2\left(\overline{z},\overline{x};C^{-1}(B(\mathrm{tl'}(\overline{z};),\overline{x};)f_1''(\overline{z},\overline{x};\overline{y})),\right.\right.\\
 & & \left.\left.C^{-1}(B(\mathrm{tl'}(\overline{z};),\overline{x};)f_2''(\overline{z},\overline{x};\overline{y})),\overline{y}\right)\right)
\end{array}
$$

f_1' and f_2' are then given by:

$$
\begin{array}{rcl}
f_1'(\overline{z},\overline{x};\overline{y}) & = & C^{-1}(B(\overline{z},\overline{x};);f_1''(\overline{z},\overline{x}))\\
f_2'(\overline{z},\overline{x};\overline{y}) & = & C^{-1}(B(\overline{z},\overline{x};);f_2''(\overline{z},\overline{x}))
\end{array}
$$

□

Proposition 6.2.3 *For any structure $\mathcal{K}$, any function $\phi:\emptyset\times\mathbb{K}^*\to\{\mathbf{0},\mathbf{1}\}^*$, $SSR_{\mathcal{K}}(\phi)=SR_{\mathcal{K}}(\phi)$.*

PROOF.

We only need to prove that simple safe recursive functions can compute safe recursive functions. The other direction is trivial.

We proceed by induction on the definition tree. For basic safe functions and ϕ, the result is trivial. For a function defined with safe composition, the induction hypothesis gives the result.

Assume that $f_1, f_2 : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ are defined by a simultaneous safe recursion scheme as follows:

$$\left\{ \begin{array}{rcl} f_1(\epsilon,\overline{x};\overline{y}) & = & h_1(\overline{x};\overline{y}) \\ f_2(\epsilon,\overline{x};\overline{y}) & = & h_2(\overline{x};\overline{y}) \\ f_1(a.\overline{z},\overline{x};\overline{y}) & = & g_1(\overline{z},\overline{x}; f_1(\overline{z},\overline{x};\overline{y}), f_2(\overline{z},\overline{x};\overline{y}),\overline{y}) \\ f_2(a.\overline{z},\overline{x};\overline{y}) & = & g_2(\overline{z},\overline{x}; f_1(\overline{z},\overline{x};\overline{y}), f_2(\overline{z},\overline{x};\overline{y}),\overline{y}), \end{array} \right.$$

The induction hypothesis allows us to assume that functions h_1, h_2, g_1, g_2 are in $\mathrm{SR}_{\mathcal{K}}(\phi)$, i.e. they are simply safe recursive. Assume moreover that they respect the homogeneous length hypothesis of Definition 6.2.2.

This hypothesis allows us to define a function $H : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ such that:

$$H(\overline{z},\overline{x};\overline{y}) = f_1(\overline{z},\overline{x};\overline{y}).f_2(\overline{z},\overline{x};\overline{y})$$

Let us define a simple safe recursive function p such that:

$$\forall \overline{x},\overline{y},\overline{m} = \overline{m_x}.\overline{m_y}.\overline{m_z} \in \mathbb{K}^* \qquad \text{with } |\overline{m_x}| = |\overline{x}|, |\overline{m_y}| = |\overline{y}|$$
$$p(\overline{x},\overline{y};\overline{m}) \quad = \quad \overline{m_y}$$

This function realizes some kind of projection on $\overline{m}$, and can be formally defined with simple safe recursion as follows:

$$\begin{array}{rcl} p(\epsilon,\overline{y};\overline{m}) & = & q(\overline{y};\overline{m}) \\ p(a.\overline{x},\overline{y};\overline{m}) & = & \mathrm{tl}(;p(\overline{x},\overline{y};\overline{m})) \\ q(\overline{y};\overline{m}) & = & q'(\overline{y};q'(\overline{y};\overline{m})) \\ q'(\epsilon;\overline{m}) & = & \epsilon \\ q'(a.\overline{y};\overline{m}) & = & \mathrm{cons}(;\mathrm{hd}(;Tl(\overline{y};\overline{m})),q'(\overline{y};\overline{m})) \\ Tl(\epsilon;\overline{m}) & = & \overline{m} \\ Tl(a.\overline{y};\overline{m}) & = & \mathrm{tl}(;Tl(\overline{y};\overline{m})) \end{array}$$

Let us also define a generalized concatenation $Cons$ with simple safe recursion:

$$\begin{aligned}
Cons(\overline{z};\overline{x},\overline{y}) &= Cons'(\overline{z};q'(\overline{z};\overline{x}),\overline{y})\\
Cons'(\epsilon;\overline{x},\overline{y}) &= \overline{y}\\
Cons'(a.\overline{z};\overline{x},\overline{y}) &= \mathrm{cons}(;\mathrm{hd}(;Tl(\overline{z};\overline{x})),Cons'(\overline{z};\overline{x},\overline{y}))
\end{aligned}$$

$Cons(\overline{z};\overline{x},\overline{y})$ returns $\overline{x}.\overline{y}$ provided that $|\overline{z}| \geq |\overline{x}|$. Let us detail now how we can formally define H with simple safe recursion. Since h_1, h_2, g_1 and g_2 respect the homogeneous length hypothesis, we have:

$$\begin{aligned}
|h_1(\overline{x};\overline{y})| &= |L(\epsilon,\overline{x})|\\
|h_2(\overline{x};\overline{y})| &= |L(\epsilon,\overline{x})|\\
|f_1(\overline{z},\overline{x};\overline{y})| &= |L(\mathrm{tl}'(\overline{z};),\overline{x};)| \text{ with } \mathrm{tl}'(\overline{z};) = \mathrm{tl}(;\overline{z})\\
|f_2(\overline{z},\overline{x};\overline{y})| &= |L(\mathrm{tl}'(\overline{z};),\overline{x};)|
\end{aligned}$$

Then, our definition of H is:

$$\begin{aligned}
H(\epsilon,\overline{x};\overline{y}) &= Cons(L(\epsilon,\overline{x};);h_1(\overline{x};\overline{y}),h_2(\overline{x};\overline{y}))\\
H(a.\overline{z},\overline{x};\overline{y}) &= Cons(L(\overline{z},\overline{x};);\\
&\quad g_1\left(\overline{z},\overline{x};p(\epsilon,L(\mathrm{tl}'(\overline{z};),\overline{x};),H(\overline{z},\overline{x};\overline{y})),\right.\\
&\quad \left.p(L(\mathrm{tl}'(\overline{z};),\overline{x};),L(\mathrm{tl}'(\overline{z};),\overline{x}),H(\overline{z},\overline{x};\overline{y})),\overline{y}\right),\\
&\quad g_2\left(\overline{z},\overline{x};p(\epsilon,L(\mathrm{tl}'(\overline{z};),\overline{x};),H(\overline{z},\overline{x};\overline{y})),\right.\\
&\quad \left.p(L(\mathrm{tl}'(\overline{z};),\overline{x};),L(\mathrm{tl}'(\overline{z};),\overline{x}),H(\overline{z},\overline{x};\overline{y}))\,,\overline{y}\right))
\end{aligned}$$

The end of the proof follows directly from Lemma 6.2.2.

□

We are now able to state the main result of this section, which is our first characterization of a complexity class.

Theorem 6.2.1 *Let $\Phi \in \mathbb{K}^*$ be a decision problem over $\mathcal{K}$, and denote by $\phi : \emptyset \times \mathbb{K}^* \to \{\mathbf{0},\mathbf{1}\}$ its characteristic function. Then, a function is in the class $\mathrm{FP}_{\mathcal{K}}^{\Phi}$ of functions computable in polynomial time with oracle Φ if and only if it can be defined in $SR_{\mathcal{K}}(\phi)$.*

Proof. Immediate from Propositions 6.2.2 and 6.2.3.

□

The following result extends the characterization of FP by Bellantoni and Cook [BC92] to the setting of computations over arbitrary structures.

Corollary 6.2.1 *Over any structure* $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i\in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$*, a function is computed in polynomial time by a BSS machine if and only if it can be defined in* $\mathrm{SR}_{\mathcal{K}}$.

□

6.3 Circuit-Based Arguments for Corollary 6.2.1

As in the previous chapter, there is a strong similarity between Corollary 6.2.1 and previous, classical results over Turing machines and boolean circuits. A careful use of these results, with the notion of P-uniform families of circuits, described by classical Turing machines, provides a strong link with the classical characterization by Bellantoni and Cook [BC92]. It gives also a simpler proof, where no simultaneous recursion scheme is needed. However, we are not able to use these arguments for Theorem 6.2.1, since the notion of oracle call, with variable input size, is not easily embedded in circuits.

More precisely, we use again Proposition 1.2.1 to reduce BSS machine computations to circuit evaluation. Next, we give a simulation of the circuit given by this reduction by safe recursive functions over an arbitrary structure $\mathcal{K}$. The reduction as well as the simulation of the family of circuits is done by safe recursive functions. When dealing with a polynomial time BSS machine, the reduction is polynomial in the classical meaning, and we obtain a P-uniform family of circuits. The reduction and the simulation of the family of circuits is done by safe recursive functions.

Remark 6.3.1 It is also noticeable that all circuit based arguments used here are the same as those used previously for Theorem 5.2.1. The proofs of these two results could therefore be factorized into a single one, as in [BCdNM04a]. For a clearer exposition we give them separately here.

6.3.1 Evaluation of P-Uniform Circuit Families with Safe Recursion

In this section, we make the simulation of a P-uniform family of circuits by safe recursive functions explicit. The simulation is essentially the same as the one performed in

Lemma 5.3.1 of Section 5.3.1. The only difference lies in the use of safe recursion instead of primitive recursion.

Lemma 6.3.1 *Assume $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ is a family of circuits over $\mathcal{K}$ such that:*

- *$\mathscr{C}_n$ has $n+m$ input gates $(x_1, \ldots, x_n, y_1, \ldots, y_m)$,*
- *there exists a function $t : \mathbb{N} \to \mathbb{N}$ such that, for all $n \in \mathbb{N}$, $|\mathscr{C}_n| \leq t(n)$ and $t(n) \geq n$*
- *there exists a safe recursive function T such that $T(\overline{x};) = \mathbf{1}^{t(|\overline{x}|)}$*
- *there exist safe recursive functions $\mathsf{Gate}, F_1, \ldots F_r$ (here r is the maximum arity of the symbols of $\mathcal{K}$) such that $\mathsf{Gate}(T(\overline{x};), \mathbf{1}^i, \overline{x};)$ describes the i^{th} node of $\mathscr{C}_{|\overline{x}|}$ as follows*

$$\mathsf{Gate}(T(\overline{x};), \mathbf{1}^i, \overline{x};) \quad = \quad \begin{cases} \mathbf{0} & \text{if } i \text{ is an input gate} \\ \mathbf{0.0} & \text{if } i \text{ is an output gate} \\ \mathbf{1}^j & \text{if } i \text{ is a gate labeled with op}_j \\ \mathbf{1}^{k+j} & \text{if } i \text{ is a gate labeled with rel}_j \\ \mathbf{1}^{k+l+1} & \text{if } i \text{ is a selection node} \\ \epsilon & \text{otherwise} \end{cases}$$

  and $F_j(T(\overline{x};), \mathbf{1}^i, \overline{x};)$ identifies the j^{th} parent node of the i^{th} node of $\mathscr{C}_{|\overline{x}|}$ as follows

$$F_j(T(\overline{x};), \mathbf{1}^i, \overline{x};) = \mathbf{1}^k \text{ where } k \leq i \text{ is the } j^{th} \text{ parent node of } i.$$

Then, given a set of constants $\overline{\alpha} = \alpha_1, \ldots, \alpha_m \in \mathbb{K}^m$, there exists a safe recursive function $C^*_{\overline{\alpha}}$ over $\mathcal{K}$ such that, for any $\overline{x} = x_1. \ldots .x_n$, $C^*_{\overline{\alpha}}(\overline{x};)$ equals $\mathscr{C}_n(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m)$.

Proof. It is easy to define a safe recursive function Pick such that, for any $\overline{z}, \overline{t}, \overline{x} \in \mathbb{K}^*$ satisfying $|\overline{t}| \leq |\overline{x}| \leq |\overline{z}|$,

$$\mathsf{Pick}(\overline{z}; \overline{t}, \overline{x}) = x_{n+1-|\overline{t}|}$$

where $\overline{x} = x_1. \ldots .x_n$. Note that $\overline{z}$ does not occur in the right-hand side of the equality above. It is used to control the recursion and ensures that x can be placed in a safe position. The contents of its components is irrelevant. We only require $|\overline{x}| \leq |\overline{z}|$.

We next want to define a safe recursive function $V_{\overline{\alpha}}$ which, on an input $(\overline{z}, \overline{x};)$, computes the evaluation of all gates numbered from 1 to $|\overline{z}|$ of $\mathscr{C}_{|\overline{x}|}$ on input $(\overline{x}, \overline{\alpha})$ and concatenates the results in one word. For the sake of readability, we denote by $\mathcal{F}_p$ the expression $\mathsf{Pick}(T(\overline{x};); F_p(T(\overline{x};), 1.\overline{z}, \overline{x};), V_{\overline{\alpha}}(\overline{z}, \overline{x}))$. This expression gives the evaluation of the pth parent gate of the current gate (which is the one numbered by $|\overline{z}| + 1$). It is obtained by "Picking" it at the right position in the recurrence argument. Denote by $k(i)$ the arity of op_i and by $l(i)$ the arity of rel_i. We may define $V_{\overline{\alpha}}$ as follows (we use definition by cases which can be easily described by combining Select operators and simple safe recursive functions) where $G = \mathsf{Gate}(T(\overline{x};), \mathbf{1}.\overline{z}, \overline{x};)$ is the type of the current gate,

$$
\begin{array}{rcl}
V_{\overline{\alpha}}(\epsilon, \overline{x};) & = & \epsilon \\
V_{\overline{\alpha}}(c.\overline{z}, \overline{x};) & = & \left\{ \begin{array}{lr}
\mathsf{cons}(; x_i, V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{0} \text{ and } |\overline{z}| + 1 = i \leq |\overline{x}| \\
\mathsf{cons}(; \alpha_{i-|\overline{x}|}, V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{0} \\
 & \text{and } |\overline{z}| + 1 = i \in [|\overline{x}| + 1, |\overline{x}| + m] \\
\mathsf{cons}(; \mathrm{op}_j(; \mathcal{F}_1, \ldots, \mathcal{F}_{k(j)}), V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{1}^j \\
\mathsf{cons}(; \mathrm{rel}_j(; \mathcal{F}_1, \ldots, \mathcal{F}_{l(j)}), V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{1}^{k+j} \\
\mathsf{cons}(; \mathsf{Select}(; \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3), V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{1}^{k+l+1} \\
\mathsf{cons}(; \mathcal{F}_1, V_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } G = \mathbf{0}.\mathbf{0} \\
V_{\overline{\alpha}}(\overline{z}, \overline{x};) & \text{otherwise.}
\end{array} \right.
\end{array}
$$

Note that the six first cases in the definition above correspond to the current node being input, constant, operator, relation, selection and output, respectively. The seventh case is to deal with unexpected values of $\overline{z}$, e.g., a too large gate number. We define now a function $R_{\overline{\alpha}}$ which concatenates the values returned by the outputs gates of $\mathscr{C}_{|\overline{x}|}$,

$$
\begin{array}{rcl}
R_{\overline{\alpha}}(\epsilon, \overline{x};) & = & \epsilon \\
R_{\overline{\alpha}}(c.\overline{z}, \overline{x};) & = & \left\{ \begin{array}{lr}
\mathsf{cons}(; V_{\overline{\alpha}}(\overline{z}, \overline{x};), R_{\overline{\alpha}}(\overline{z}, \overline{x};)) & \text{if } \mathsf{Gate}(T(\overline{x};), \overline{z}, \overline{x};) = \mathbf{0}.\mathbf{0} \\
R_{\overline{\alpha}}(\overline{z}, \overline{x};) & \text{otherwise.}
\end{array} \right.
\end{array}
$$

The result of the evaluation of $\mathscr{C}_{|\overline{x}|}$ on input $(\overline{x}, \overline{\alpha})$ is given by $R_{\overline{\alpha}}(T(\overline{x};), \overline{x};)$.

□

6.3.2 Proof of Corollary 6.2.1

The proof relies on the previous Lemma 6.3.1, which allows to simulate a polynomial time BSS machine.

Assume M is a polynomial time BSS machine over $\mathcal{K}$ computing f_M. Denote by $\alpha_1, \ldots, \alpha_m \in \mathbb{K}$ the constants used by M. Assume that the computation time is bounded by some polynomial $P(n) = cn^d$. There exist polynomial time BSS machines M' and M'', with the same constants, such that, on input $\overline{x}$, M' returns $\mathbf{1}^{|f_M(\overline{x})|}.\mathbf{0}^{P(|\overline{x}|) - |f_M(\overline{x})|}$, and M'' returns $f_M(\overline{x}).\mathbf{0}^{P(|\overline{x}|) - |f_M(\overline{x})|}$. The strings of $\mathbf{0}$s ensure that the output length depends only on the length of the input. Denote by $f_{M'}$ and $f_{M''}$ the corresponding functions.

By Proposition 1.2.1 there exist two P-uniform families of circuits computing $f_{M'}$ and $f_{M''}$.

By Lemma 6.3.1, these two families of circuits are simulated by safe recursive functions $F_{M'}$ and $F_{M''}$ over $\mathcal{K}$ respectively. Therefore, $F_{M'}$ and $F_{M''}$ coincide with $f_{M'}$ and $f_{M''}$

respectively. We next define a safe recursive function Hd such that, for $\overline{z}$ of length n and $\overline{y}$ of length at least n, $\mathsf{Hd}(\overline{z};\overline{y})$ returns the first n characters of $\overline{y}$. In order to do so we introduce Tl such that $\mathsf{Tl}(\overline{z};\overline{y})$ iterates n times tl on $\overline{y}$, and RevHd which gives the same result as Hd in the reversed order. We define also a safe recursive function unpad for eliminating all padding $\mathbf{0}$s from the output of $F_{M'}$. These functions are formally defined by

$$
\begin{array}{rcl}
\mathsf{Tl}(\epsilon;\overline{y}) & = & \overline{y} \\
\mathsf{Tl}(a.\overline{z};\overline{y}) & = & \mathsf{tl}(;\mathsf{Tl}(\overline{z};\overline{y})) \\
\mathsf{RevHd}(\epsilon;\overline{y}) & = & \epsilon \\
\mathsf{RevHd}(a.\overline{z};\overline{y}) & = & \mathsf{cons}(;\mathsf{Tl}(\overline{z};\overline{y}),\mathsf{RevHd}(\overline{z};\overline{y})) \\
\mathsf{Hd}(\overline{z};\overline{y}) & = & \mathsf{RevHd}(\overline{z};\mathsf{RevHd}(\overline{z};\overline{y})) \\
f(\epsilon;) & = & \epsilon \\
f(a.\overline{z};) & = & \left\{ \begin{array}{lr} \mathsf{cons}(;\mathbf{1},f(\overline{z};)) & \text{if } \mathsf{hd}(;\overline{z}) = \mathbf{1} \\ \epsilon & \text{otherwise} \end{array} \right. \\
\mathsf{unpad}(\overline{z};) & = & \mathsf{cons}(;\mathbf{1},f(\overline{z};)).
\end{array}
$$

Then f_M can be defined by

$$f_M(\overline{x};) = \mathsf{Hd}(\mathsf{unpad}(F_{M'}(\overline{x};););F_{M''}(\overline{x})).$$

$\square$

Chapter 7

Parallelism and Non-Determinism

In this chapter we extend the characterization of the classical complexity class PSPACE introduce in [LM95] to capture the class $\mathrm{FPAR}_{\mathcal{K}}$ of functions computable in parallel polynomial time by a BSS machine over $\mathcal{K}$. Next, based on the classical characterization of the polynomial hierarchy [Bel94], we propose a machine independent characterization of the levels of the polynomial hierarchy over an arbitrary structure $\mathcal{K}$. Finally , we provide a characterization of the class $\mathrm{FPAT}_{\mathcal{K}}$ of functions computable in polynomial alternating time. Versions of our characterizations for capturing digital non-determinism are also provided. Our characterization of $\mathrm{FPAR}_{\mathcal{K}}$ are published in [BCdNM03b, BCdNM03a, BCdNM04a], of the polynomial hierarchy in [BCdNM03b, BCdNM04a] and of $\mathrm{FPAT}_{\mathcal{K}}$ in [BCdNM04b].

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

This section is devoted to show a characterization of the parallel class $\mathrm{FPAR}_{\mathcal{K}}$ in terms of safe recursive functions with substitutions over $\mathcal{K}$. We first introduce this new class of functions.

7.1.1 Safe Recursion with Substitutions

Definition 7.1.1 (Safe Recursive Functions with Substitutions)
The set of functions defined with *safe recursion with substitutions* over $\mathcal{K}$, denoted as $\mathrm{SRS}_{\mathcal{K}}$, 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:

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

The functions σ_j are called *substitution functions.*

Our first important result in this chapter is the following characterization of $\mathrm{FPAR}_{\mathcal{K}}$.

Theorem 7.1.1 *Over any structure $\mathcal{K} = (\mathbb{K}, \{op_i\}_{i \in I}, rel_1, \ldots, rel_l, \mathbf{0}, \mathbf{1})$, a function is computed in parallel polynomial time by if and only if it can be defined in* $\mathrm{SRS}_{\mathcal{K}}$.

□

7.1.2 Proof of Theorem 7.1.1 ("only if" part)

By hypothesis, the family of circuits we want to simulate here is P-uniform. This means that there exists a machine over $\mathcal{K}$ which, given a pair (n, i), computes the description of the ith gate of the circuit $\mathscr{C}_n$ in time polynomial in n. Let $L(n)$ be the polynomial bounding the number of output gates of $\mathscr{C}_n$ and, for $1 \leq j \leq L(n)$ let $\mathscr{C}_{nj}$ be the circuit induced by $\mathscr{C}_n$ with only one output node (corresponding to the jth output node of $\mathscr{C}_n$).

The idea for writing a function evaluating the circuit $\mathscr{C}_{nj}$ at an input $\overline{x} \in \mathbb{K}^n$ is simple. The function is defined recursively, the evaluation of every node depending on the evaluation of its parent nodes. However, in order to do this with the mechanism of safe recursion with substitutions, we need a way to describe these parents nodes in constant time (actually with a substitution function) and this is not provided by the P-uniformity hypothesis. A way to do so is by describing a parent node simply by its relative position among the possible parent nodes of the node at hand. The iteration of this procedure naturally leads to the notion of path.

A *path* from the output node o to a node w is a sequence of nodes $o = v_0, \ldots, v_\ell = w$ such that v_i is a parent of v_{i-1} for $i = 1, \ldots, \ell$. The node w is called the *top* of the path. We describe a path by an element $\overline{y} \in \mathbb{K}^*$ as follows

- The path consisting on the single output node o is represented by $\epsilon \in \mathbb{K}^*$.

- Let r be the maximal arity of a relation or a function of the structure. Then $s = \lceil \log_2(\max\{r,3\}) \rceil$ is the size necessary to write the binary encoding of any parent for a given node. Assume that $v_0, \ldots, v_{\ell-1}$ is represented by $\overline{y}$. If v_ℓ is the ith parent node of $v_{\ell-1}$ then the path $v_0, \ldots, v_\ell$ is represented by $\overline{a_i}.\overline{y}$, where $\overline{a_i} \in \{\mathbf{0}, \mathbf{1}\}^*$ represents the binary encoding of i. We add as many $\mathbf{0}$s as needed in front such that $\overline{a_i}$ have length s.

Lemma 7.1.1 *There exists a safe recursive function* Gate *such that, for an element* $\overline{x} \in \mathbb{K}^n$*, a natural* $j \leq L(n)$*, and a path* $\overline{y}$ *in* $\mathscr{C}_{nj}$*,*

$$\mathsf{Gate}(\mathbf{1}^j, \overline{x}; \overline{y}) = \begin{cases} \mathbf{0} & \text{if } \mathsf{top}(\overline{y}) \text{ is the } i\text{th input gate} \\ \mathbf{0}.\mathbf{0} & \text{if } \mathsf{top}(\overline{y}) \text{ is the output gate} \\ \mathbf{1}^i & \text{if } \mathsf{top}(\overline{y}) \text{ is a gate labeled with } op_i \\ \mathbf{1}^{k+i} & \text{if } \mathsf{top}(\overline{y}) \text{ is a gate labeled with } rel_i \\ \mathbf{1}^{k+l+1} & \text{if } \mathsf{top}(\overline{y}) \text{ is a selection node} \\ \epsilon & \text{otherwise} \end{cases}$$

where $\mathsf{top}(\overline{y})$ *denotes the top of* $\overline{y}$*.*

PROOF. Since the length of $\overline{y}$ is polynomial in n, the function Gate can be computed in polynomial time. Therefore, it is safe recursive by Theorem 6.2.1.

□

Assume $p(n) = cn^d$ is a polynomial bounding the depth of the circuit $\mathscr{C}_n$. The goal now is to describe a function Eval, definable via safe recursion with substitutions, such that $\mathsf{Eval}(\mathbf{1}^j, \overline{t}, \overline{x}; \epsilon)$ is the output of $\mathscr{C}_{nj}$ on input $\overline{x}$ whenever $|\overline{x}| = n$ and $|\overline{t}| = p(n)$.

To do so we will use the substitution functions σ_i defined by $\sigma_i(;\overline{y}) = \overline{a_i}.\overline{y}$ where $\overline{a_i} \in \{\mathbf{0}, \mathbf{1}\}^*$ is the binary encoding of i. Note that σ_i is safe recursive for $i = 1, \ldots, r$. Also, note that a path $\overline{y}$ of length ℓ is easily described by composing ℓ times some of these functions.

Denote by $k(i)$ the arity of op_i, and $l(i)$ the arity of rel_i. Also, denote by $\mathcal{F}_p$ the expression $\mathsf{Eval}(\mathbf{1}^j, \overline{t}, \overline{x}; \sigma_p(;\overline{y}))$ which gives the evaluation of the pth parent gate of the current gate and by $G = \mathsf{Gate}(\mathbf{1}^j, \overline{x}; \overline{y})$ the type of the current gate. The function Eval is then defined as follows,

$$\begin{aligned} \mathsf{Eval}(\mathbf{1}^j, \epsilon, \overline{x}; \overline{y}) &= \epsilon \\ \mathsf{Eval}(\mathbf{1}^j, a.\overline{t}, \overline{x}; \overline{y}) &= \begin{cases} x_i & \text{if } G = \mathbf{0} \\ \mathrm{op}_i(;\mathcal{F}_1, \ldots, \mathcal{F}_{k(i)}) & \text{if } G = \mathbf{1}^i \\ \mathrm{rel}_i(;\mathcal{F}_1, \ldots, \mathcal{F}_{l(i)}) & \text{if } G = \mathbf{1}^{k+i} \\ \mathsf{Select}(;\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3) & \text{if } G = \mathbf{1}^{k+l+1} \\ \mathcal{F}_1 & \text{if } G = \mathbf{0}.\mathbf{0} \end{cases} \end{aligned}$$

All equality and inequality tests are once again easily obtained by combining Select operators and simple safe recursive functions.

Note that we use above an enumeration for the gates of $\mathscr{C}_{nj}$ different to the one described in Remark 1.2.1. The expressions $\mathcal{F}_p$ evaluating the pth parent of a gate are, consequently, different here and in the proof of Lemma 6.3.1.

Theorem 6.2.1 ensures the existence of a safe recursive functions P_{cd} such that $|P_{cd}(\overline{x};)| = cn^d$. We use P_{cd} to define with safe recursion over $\mathcal{K}$ the following function

$$\begin{aligned}\mathsf{concat}(\epsilon, \overline{x};) &= \mathsf{Eval}(\mathbf{1}, P_{cd}(\overline{x};), \overline{x}.\overline{z}; \epsilon)\\ \mathsf{concat}(a.\overline{y}, \overline{x};) &= \mathsf{cons}(; \mathsf{Eval}(\overline{y}, P_{cd}(\overline{x};), \overline{x}.\overline{z}; \epsilon), \mathsf{concat}(\overline{y}, \overline{x};))\end{aligned}$$

such that $\mathsf{concat}(\overline{y}, \overline{x};))$ concatenates the outputs of $\mathscr{C}_{nj}$ for $j = 1, \ldots, |\overline{y}|$.

Theorem 6.2.1 also ensures the existence of a safe recursive functions P_L such that $P_L(\overline{x}) = \mathbf{1}^{L(n)}$. The function computed by the P-uniform family of circuits $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ is then defined with safe recursion with substitutions by $\mathsf{Circuit}(\overline{x};) = \mathsf{concat}(P_L(\overline{x};), \overline{x};)$.

□

7.1.3 Proof of Theorem 7.1.1 ("if" part)

Again, we give the proof for $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$.

Let f be a function defined with safe recursion with substitutions, and denote by f_n the restriction of f to the set of inputs of size n. We need to prove that f can be computed by a P-uniform family of circuits $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ of polynomial depth.

As in Proposition 6.2.1 of Section 6.2.2, for a function $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ defined with safe recursion with substitutions, we denote by $D[f(\overline{x};\overline{y})]$ the depth of the shortest circuit $\mathscr{C}$ computing $f(\overline{x};\overline{y})$.

Proposition 7.1.1 *Let $f : \mathbb{K}^* \times \mathbb{K}^* \to \mathbb{K}^*$ be defined by safe recursion with substitutions. There exists a polynomial p_f such that, for all $(\overline{x}, \overline{y}) \in \mathbb{K}^* \times \mathbb{K}^*$,*

$$D[f(\overline{x};\overline{y})] \leq p_f(|\overline{x}|)$$

and

$$|f(\overline{x};\overline{y})| \leq p_f(|\overline{x}|).$$

PROOF. It is done by induction on the depth of the definition tree of f, as in the proof of Proposition 6.2.1 (to which this proof is actually similar).

If f is a basic safe function then it can be evaluated in constant time. Therefore, the statement is true for basic safe functions.

Assume now f is defined by safe composition from safe recursive functions $g, h_1, \ldots, h_{m+n}$ as in Definition 6.1.1. Then, there exists polynomials $p_g, p_1, \ldots, p_{m+n}$ satisfying the statement for these functions respectively. Without loss of generality, we may assume these polynomials to be nondecreasing. Therefore,

$$\begin{aligned} D[f(\overline{x};\overline{y})] &\leq \max\{D[h_1(\overline{x};\overline{y})], \ldots, D[h_{m+n}(\overline{x};\overline{y})]\} \\ &\quad + D[g(h_1(\overline{x};\overline{y}), \ldots, h_m(\overline{x};\overline{y}); h_{m+1}(\overline{x};\overline{y}), \ldots, h_{m+n}(\overline{x};\overline{y}))] \\ &\leq \max\{p_1(|\overline{x}|), \ldots, p_{m+n}(|\overline{x}|)\} + p_g(p_1(|\overline{x}|) + \cdots + p_m(|\overline{x}|)) \end{aligned}$$

which shows we may take

$$p_f = \max\{p_1, \ldots, p_{m+n}\} + p_g \circ (p_1 + \cdots + p_m)$$

where $\max\{p_1, \ldots, p_{m+n}\}$ denotes any polynomial bounding above $p_1, \ldots, p_{m+n}$.

Finally, assume f is defined by safe recursion with substitutions and let $h_1, \ldots, h_k, g_1, \ldots, g_k$ and σ_{ij} ($i = 1, \ldots, k$ and $j = 1, \ldots, l$) as in Definition 7.1.1. Then, there exist polynomials $p_{g_1}, \ldots, p_{g_k}, p_{h_1}, \ldots, p_{h_k}$ satisfying the statement for $g_1, \ldots, g_k, h_1, \ldots, h_k$ respectively. Again, without loss of generality, we may assume $p_{g_i} = p_g$ for $i = 1, \ldots, k$ and $p_{h_i} = p_h$ for $i = 1, \ldots, k$. In addition, we also may assume that p_g and p_h are nondecreasing. Therefore, for $i = 1, \ldots, k$,

$$D[f_i(\epsilon, \overline{x}; \overline{u}, \overline{y})] = p_h(|\overline{x}|)$$

and

$$\begin{aligned} D[f_i(a.\overline{z}, \overline{x}; \overline{u}, \overline{y})] &\leq \max_{i,j}\{D[f_i(\overline{z}, \overline{x}; \sigma_{ij}(;\overline{u}), \overline{y})]\} \\ &\quad + D\big[g_i\big(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; \sigma_{11}(;\overline{u}), \overline{y}), \ldots, f_1(\overline{z}, \overline{x}; \sigma_{1l}(;\overline{u}), \overline{y}), \\ &\qquad \ldots, f_k(\overline{z}, \overline{x}; \sigma_{k1}(;\overline{u}), \overline{y}), \ldots, f_k(\overline{z}, \overline{x}; \sigma_{kl}(;\overline{u}), \overline{y}), \overline{y}\big)\big] \\ &\leq \max_{i,j}\{D[f_i(\overline{z}, \overline{x}; \sigma_{ij}(;\overline{u}), \overline{y})]\} + p_g(|\overline{z}, \overline{x}|). \end{aligned}$$

Applying the same reasoning for $f_i(\overline{z}, \overline{x}; \sigma_{ij}(; \overline{u}), \overline{y})$ and denoting $\overline{z'} = \mathsf{tl}(\overline{z})$ we obtain

$$\begin{aligned}
& D[f_i(\overline{z}, \overline{x}; \sigma_{ij}(; \overline{u}), \overline{y})] \\
\leq \quad & D[\sigma_{ij}(; \overline{u})]\} + \max_{i',j'}\{D[f_{i'}(\overline{z}, \overline{x}; \sigma_{i'j'}(; \sigma_{ij}(; \overline{u})), \overline{y})]\} \\
& + D\big[g_i\big(\overline{z}, \overline{x}; f_1(\overline{z}, \overline{x}; ; \sigma_{11}(; \sigma_{ij}(; \overline{u})), \overline{y}), \ldots, f_1(\overline{z}, \overline{x}; \sigma_{1l}(; \sigma_{ij}(; \overline{u})), \overline{y}), \\
& \qquad \ldots, f_k(\overline{z}, \overline{x}; \sigma_{k1}(; \sigma_{ij}(; \overline{u})), \overline{y}), \ldots, f_k(\overline{z}, \overline{x}; \sigma_{kl}(; \sigma_{ij}(; \overline{u})), \overline{y}), \overline{y}\big)\big] \\
\leq \quad & D[\sigma_{ij}(; \overline{u})]\} + \max_{i',j'}\{D[f_{i'}(\overline{z}, \overline{x}; \sigma_{i'j'}(; \sigma_{ij}(; \overline{u})), \overline{y})]\} + p_g(|\overline{z'}, \overline{x}|).
\end{aligned}$$

Continuing in this way and denoting by $\overline{z_\ell}$ the result of applying ℓ times tl to $\overline{z}$ we obtain

$$\begin{aligned}
D[f_i(a.\overline{z}, \overline{x}; \overline{u}, \overline{y})] \quad \leq \quad & \sum_{\ell=1}^{|a.\overline{z}|} \left(\max_{\substack{i_1,\ldots,i_\ell \leq k \\ j_1,\ldots,j_\ell \leq l}} \left\{ D[\sigma_{i_1 j_1}(; \sigma_{i_2 j_2}(; \ldots \sigma_{i_{\ell-1} j_{\ell-1}}(; u) \ldots))] \right\} + p_g(\overline{z_\ell}, \overline{x}) \right) \\
& + \max_{\substack{i_1,\ldots,i_\ell \leq k \\ j_1,\ldots,j_\ell \leq l}} \left\{ D[h_{i_1}(\overline{x}; \sigma_{i_1 j_1}(; \sigma_{i_2 j_2}(; \ldots \sigma_{i_\ell j_\ell}(; u) \ldots)), \overline{y})] \right\} \\
\leq \quad & |a.\overline{z}|\big(\mathcal{O}(|\overline{z}|) + p_g(|\overline{z}, \overline{x}|)\big) + p_h(|\overline{x}|)
\end{aligned}$$

the $\mathcal{O}(|\overline{z}|)$ since each σ_{ij} can be computed in constant time by Proposition 6.2.1, and that we compute them sequentially. This yields again a polynomial bound for the depth.

The polynomial bound for the output size is obvious and we can take p_f to be a polynomial bounding both depth and output size.

□

Proposition 7.1.1 shows that every circuit $\mathscr{C}_n$ of the family computing f has depth polynomial in n. The P-uniformity of this family is implicit in the proof of the proposition.

In the classical setting [LM95], safe recursion with substitution characterizes the class FPSPACE. 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 in the classical setting we have FPAR = FPSPACE, our result extends the classical one from [LM95].

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

7.2.1 Safe Recursion with Predicative Minimization

In the spirit of [Bel94], we now introduce the notion of predicative minimization.

Definition 7.2.1 (Predicative Minimization)

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

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

We now introduce new sets of functions.

Definition 7.2.2 (Restricted Safe Recursive Functions Relative to F)

Let F be a class of functions. The set of *restricted safe recursive functions relative to F* over $\mathcal{K}$, denoted by $\mathrm{RSR}_{\mathcal{K}}(\mathrm{F})$, is the smallest set of functions containing the basic safe functions and F, and closed under safe composition 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 $\mathrm{RSR}_{\mathcal{K}}(\mathrm{F})$ and g to $\mathrm{SR}_{\mathcal{K}}$. This implies that no function in $\mathrm{F}\backslash\mathrm{SR}_{\mathcal{K}}$ may be involved in the definition of g.

Definition 7.2.3 ($\ni\mathrm{PH}_{\mathcal{K}}$)

Assume F is a class of functions: a function f is in $\ni\mathrm{F}$ if it is defined with one predicative minimization over a function h of F.

We define by induction the following sets:

- $\mathrm{F}^0_{\mathcal{K}} = \mathrm{SR}_{\mathcal{K}}$.
- for $i \geq 0$ $\mathrm{F}^{i+1}_{\mathcal{K}} = \mathrm{RSR}_{\mathcal{K}}(\mathrm{F}^i_{\mathcal{K}} \bigcup \ni\mathrm{F}^i_{\mathcal{K}})$.

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

Remark 7.2.1 This notion of restricted safe recursion ensures that, in for any function f in $\mathrm{F}^i_{\mathcal{K}}$, there are at most i nested predicative minimizations. This bound does not depend on the arguments of f. In other words, there exists h in $\mathrm{SR}_{\mathcal{K}}$ and $f_1,\ldots,f_n$ in $\mathrm{F}^{i-1}_{\mathcal{K}}$, such that, for all $\mathbf{x} = (\overline{x_1},\ldots,\overline{x_l})$,

$$f(\mathbf{x};) = h(\mathbf{x};\ni\overline{z_1}(f_1(\mathbf{x};\overline{z_1})),\ldots,\ni\overline{z_n}(f_n(\mathbf{x};\overline{z_n}))).$$

Lemma 7.2.1 *Assume $f : (\mathbb{K}^*)^n \times \emptyset \to \mathbb{K}^*$ is a function in $\mathrm{F}\Delta_{\mathcal{K}}^i$. Then f can be defined in $F_{\mathcal{K}}^i$.*

PROOF. By induction on i. For $i = 0$, f is in $\mathrm{F}\Delta_{\mathcal{K}}^0 = \mathrm{FP}_{\mathcal{K}}$ and we may apply Theorem 6.2.1. Assume now that the result holds for $i > 0$.

Let f be a function in $\mathrm{F}\Delta_{\mathcal{K}}^i$. By definition of $\mathrm{F}\Delta_{\mathcal{K}}^i$, there exist a polynomial time BSS machine M_f and a set Φ in $\Sigma_{\mathcal{K}}^i$ such that, for all $\overline{x} \in \mathbb{K}^*$, $f(\overline{x})$ is computed by M_f with oracle Φ. We are now establishing that the oracle Φ can be denoted by a function in $\mathrm{F}_{\mathcal{K}}^i$.

Since $\Phi \in \Sigma_{\mathcal{K}}^i = \mathrm{NP}_{\mathcal{K}}^{\Sigma_{\mathcal{K}}^{i-1}}$ there exist a deterministic polynomial-time BSS machine M_g over $\mathcal{K}$ and a set $\Psi \in \Sigma_{\mathcal{K}}^{i-1}$ such that

$$\overline{x} \in \Phi \Leftrightarrow \exists \overline{y} \text{ s.t. } M_g \text{ accepts } (\overline{x}, \overline{y}) \text{ with oracle } \Psi.$$

Denote by g the characteristic function computed by M_g with oracle Ψ and let ψ be the characteristic function of Ψ. Then, apply Theorem 6.2.1: g belongs to $\mathrm{SR}(\psi)_{\mathcal{K}}$. Since the evaluation time of M_g on $(\overline{x}, \overline{y})$ is polynomial in $|\overline{x}|$, Proposition 6.2.1 gives g' in $\mathrm{SR}(\psi)_{\mathcal{K}}$ such that: $g'(\overline{x}; \overline{y}) = g(\overline{x}, \overline{y};)$. Therefore $\phi(\overline{x};) = \boldsymbol{\exists}\overline{y}(g'(\overline{x}; \overline{y}))$ decides Φ, and, since $\Sigma_{\mathcal{K}}^{i-1} \subseteq \mathrm{F}\Delta_{\mathcal{K}}^{i-1}$, we may apply the induction hypothesis on ψ to deduce that ϕ belongs to $\mathrm{F}_{\mathcal{K}}^i$ and therefore so does f.

□

Lemma 7.2.2 *Assume $f : (\mathbb{K}^*)^n \times \emptyset \to \mathbb{K}^*$ is a function in $F_{\mathcal{K}}^i$. Then it belongs to $\mathrm{F}\Delta_{\mathcal{K}}^i$.*

PROOF. By induction on i. For $i = 0$, the result is a straightforward consequence of Theorem 6.2.1. Assume now that the result holds for $i > 0$.

Assume f is a function in $\mathrm{F}_{\mathcal{K}}^i$. Then, as in Remark 7.2.1,

$$f(\mathbf{x};) = h(\mathbf{x}; \boldsymbol{\exists}\overline{z_1}(f_1(\mathbf{x}; \overline{z_1})), \ldots, \boldsymbol{\exists}\overline{z_n}(f_n(\mathbf{x}; \overline{z_n}))).$$

By induction hypothesis, the functions $f_1, \ldots, f_n$ belong to $\mathrm{F}\Delta_{\mathcal{K}}^{i-1}$. The corresponding decision problems $f_1(\mathbf{x}; \overline{z_1}) = \mathbf{0}, \ldots, f_n(\mathbf{x}; \overline{z_n}) = \mathbf{0}$ belong to $\mathrm{P}_{\mathcal{K}}^{\Sigma_{\mathcal{K}}^{i-1}} = \Sigma_{\mathcal{K}}^{i-1}$. Indeed, they use a polynomial number of queries in $\Sigma_{\mathcal{K}}^{i-1}$. If S_{i-1} denotes a complete problem in $\Sigma_{\mathcal{K}}^{i-1}$ we can replace these different oracles by S_{i-1} (by making the oracle machine compute the reductions).

Define $g_j(\mathbf{x};) = \boldsymbol{\exists}\overline{z_j}(f_j(\mathbf{x}; \overline{z_j}))$ for $1 \le j \le n$. Then, g_j is the characteristic function of a set in $\Sigma_{\mathcal{K}}^i$. Indeed, if there exists $\overline{z_j} \in \mathbb{K}^*$ such that $f_j(\mathbf{x}; \overline{z_j}) = \mathbf{0}$, since the evaluation time for $f_j(\mathbf{x};\overline{z_j})$ is bounded by $p_j(|\mathbf{x}|)$ for some polynomial p_j, only the first $p_j(|\mathbf{x}|)$ elements of $\overline{z_j}$ may possibly be taken into account. Therefore, there exists $\overline{z'_j} \in \mathbb{K}^*$ of length $p_j(|\mathbf{x}|)$ such that $f_j(\mathbf{x};\overline{z'_j}) = \mathbf{0}$, which proves the claim. Therefore, f can be computed in polynomial time using n oracles in $\Sigma^i_{\mathcal{K}}$. If S_i denotes a complete problem in $\Sigma^i_{\mathcal{K}}$, again, we can replace these n different oracles by S_i: $f \in \mathrm{FP}_{\mathcal{K}}^{\Sigma^i_{\mathcal{K}}} = \mathrm{F}\Delta^i_{\mathcal{K}}$.

□

Theorem 7.2.1 *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* $F^i_{\mathcal{K}}$.

□

Example 7.2.1 Over the real numbers, an example of $\mathrm{NP}_{\mathbb{R}}$-complete problem is $4-\mathrm{FEAS}$: does a given polynomial of degree four have a zero? Assume by Theorem 6.2.1 that the safe recursive function $p(\overline{x};\overline{y})$ evaluates a polynomial encoded in $\overline{x}$ on an input encoded in $\overline{y}$. $4-\mathrm{FEAS}$ is then decided on $\overline{x}$ by $f(\overline{x};) = \boldsymbol{\exists}\overline{y}(p(\overline{x};\overline{y}))$.

Corollary 7.2.1 *A decision problem over* $\mathcal{K}$ *belongs to* $\mathrm{PH}_{\mathcal{K}}$ *if and only if its characteristic function is defined in* $\boldsymbol{\exists}\mathrm{PH}_{\mathcal{K}}$.

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

7.3.1 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 7.3.1 (Digital Predicative Minimization)
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

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

Definition 7.3.2 ($\boldsymbol{\exists}_{\mathrm{D}}\mathrm{PH}_{\mathcal{K}}$)
Let F be a class of functions. A function f is in $\boldsymbol{\exists}_{\mathrm{D}}\mathrm{F}$ if it is defined with one predicative minimization over a function h of F.

We define by induction the following sets:

- $\mathrm{dF}^0_{\mathcal{K}} = \mathrm{SR}_{\mathcal{K}}$
- for $i \geq 0$ $\mathrm{dF}^{i+1}_{\mathcal{K}} = \mathrm{RSR}_{\mathcal{K}}(\mathrm{F}^i_{\mathcal{K}} \bigcup \boldsymbol{\ni}_{\mathrm{D}} \mathrm{F}^i_{\mathcal{K}})$.

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

The proof of Theorem 7.2.1, *mutatis mutandis*, yields the following results.

Theorem 7.3.1 *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 7.3.1 *A decision problem over* $\mathcal{K}$ *belongs to* $\mathrm{DPH}_{\mathcal{K}}$ *if and only if its characteristic function is defined in* $\boldsymbol{\ni}_{\mathrm{D}}\mathrm{DPH}_{\mathcal{K}}$.

Example 7.3.1 Over the real numbers, a problem in $\mathrm{D}\Sigma^1_{\mathbb{R}}$ is KNAPSACK: given n objects of weight $w_i \in \mathbb{R}$ and value $v_i \in \mathbb{R}$, a weight limit W and a minimal value V, can we carry a total value greater than V? Assume by Theorem 6.2.1 that the safe recursive function $v(\overline{x};\overline{y})$ decides wether, for an instance described by $\overline{x}$ in size polynomial in n, a choice among the objects described by $\overline{y} \in \{\mathbf{0},\mathbf{1}\}^n$, the requirements of weight and value are satisfied. KNAPSACK is then decided on $\overline{x}$ by $f(\overline{x};) = \boldsymbol{\ni}_{\mathrm{D}}\overline{y}(v(\overline{x};\overline{y}))$.

When considering finite structures, this yields naturally a characterization of the classical polynomial hierarchy alternative to the one found in [Bel94]:

Corollary 7.3.2 *A decision problem belongs to* PH *if and only if its characteristic function is defined in* $\boldsymbol{\ni}_{\mathrm{D}}\mathrm{DPH}_{\{\mathbf{0},\mathbf{1}\}}$.

7.4 A Characterization of $\mathrm{PAT}_{\mathcal{K}}$

Definition 7.4.1 (Predicative Substitution)

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

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

Definition 7.4.2 (Safe Recursive Functions with Predicative Substitution) 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

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

The set $\boldsymbol{\ni}^{[1]}\mathrm{PAT}_{\mathcal{K}}$ 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.

Theorem 7.4.1 *A function is computed in* $\mathrm{FPAT}_{\mathcal{K}}$ *if and only if it can be defined in* $\boldsymbol{\ni}^{[1]}\mathrm{PAT}_{\mathcal{K}}$.

Proof. Let F be a function in $\mathrm{FPAT}_{\mathcal{K}}$, and denote by G the associated oracle in $\mathrm{PAT}_{\mathcal{K}}$. There exists a polynomial time BSS machine M over $\mathcal{K}$, and a polynomial function $q : \mathbb{N} \to \mathbb{N}$ such that, for all $\overline{x} \in \mathbb{K}^*$,

$$\begin{array}{rcl} \overline{x} \in G & \Leftrightarrow & \exists a_1 \in \mathbb{K}\, \neg\exists b_1 \in \mathbb{K} \, \ldots \, \exists a_{q(|\overline{x}|)} \in \mathbb{K}\, \neg\exists b_{q(|\overline{x}|)} \in \mathbb{K} \\ & & M \text{ accepts } (\overline{x}, a_1.b_1 \ldots a_{q(|\overline{x}|)}.b_{q(|\overline{x}|)}). \end{array}$$

Theorem 6.2.1 ensures that there exists a safe recursive function f_M over $\mathcal{K}$ such that, for any $(\overline{x}, \overline{y}) \in (\mathbb{K}^*)^2$, M accepts on input $(\overline{x}, \overline{y})$ if and only if $f_M(\overline{x}; \overline{y}) = \mathbf{1}$.

Consider now the function $F_G : (\mathbb{K}^*)^2 \times \mathbb{K}^* \to \mathbb{K}^*$ deciding G. $F_G(\epsilon, \overline{x}; \overline{u})$ simulates M on input $\overline{x}, \overline{u}$. The recurrence parameter $a.\overline{z}$ in $F_G(a.\overline{z}, \overline{x}; \overline{u})$ describes the shape of the quantifier sequence. F_G is defined with quantified safe recursion as follows, where all tests can be easily done with composition and the Select function,

$$\begin{array}{rcl} F_G(\epsilon, \overline{x}; \overline{u}) & = & f_M(\overline{x}; \overline{u}) \\ F_G(a.\overline{z}, \overline{x}; \overline{u}) & = & \begin{cases} \boldsymbol{\ni}^{[1]}c(F_G(\overline{z}, \overline{x}; c.\overline{u})) & \text{if } \mathsf{hd}(; \overline{z}) = \mathbf{1} \\ \mathbf{0} & \text{if } \mathsf{hd}(; \overline{z}) = \mathbf{0} \text{ and } \boldsymbol{\ni}^{[1]}c(F_G(\overline{z}, \overline{x}; c.\overline{u})) = \mathbf{1} \\ \mathbf{1} & \text{if } \mathsf{hd}(; \overline{z}) = \mathbf{0} \text{ and } \boldsymbol{\ni}^{[1]}c(F_G(\overline{z}, \overline{x}; c.\overline{u})) = \mathbf{0} \\ f_M(\overline{x}; \overline{u}) & \text{otherwise} \end{cases} \end{array}$$

In addition, let $g_q : \mathbb{K}^* \times \emptyset \to \mathbb{K}^*$ such that $g_q(\overline{x};) = (\mathbf{1.0})^{q(|\overline{x}|)}$. Since g_q is computable in polynomial time over $\mathcal{K}$, by Theorem 6.2.1, it is safe recursive. This function g_q actually gives the type of the quantifier at every level of the quantifier alternation for any input $\overline{x}$ to the problem G.

It is easy to check by induction on $|\overline{x}|$ that $F_G(\mathsf{cons}(\mathbf{1}, g_q(\overline{x};);), \overline{x}; \mathbf{0})$ decides whether $\overline{x}$ belongs to G. Therefore, the characteristic function χ_G of G belongs to $\mathbf{\ni}^{[1]}\mathrm{PAT}_{\mathcal{K}}$.

Consider a polynomial time machine M' with oracle G computing F. We apply Theorem 6.2.1: F belongs to $\mathrm{SR}_{\mathcal{K}}(F_G)$, i.e., $F \in \mathbf{\ni}^{[1]}\mathrm{PAT}_{\mathcal{K}}$.

The other direction of the proof is by induction on the definition of f. The only critical case is when f is defined by safe recursion with predicative substitutions, as in Definition 7.4.2. In this case, $f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y})$ equals $\mathbf{1}$ if and only if

$$\begin{array}{ll} & (\exists c \in \mathbb{K}\ f(\overline{z}, \overline{x}; c.\overline{u}, \overline{y}) = \mathbf{0} \wedge g(\overline{z}, \overline{x}; \mathbf{1}, \overline{y}) = \mathbf{1}) \\ \vee & (\forall c \in \mathbb{K}\ f(\overline{z}, \overline{x}; c.\overline{u}, \overline{y}) \neq \mathbf{0} \wedge g(\overline{z}, \overline{x}; \mathbf{0}, \overline{y}) = \mathbf{1}). \end{array}$$

If $g(\overline{z}, \overline{x}; \mathbf{1}, \overline{y}) = \mathbf{1}$ and $g(\overline{z}, \overline{x}; \mathbf{0}, \overline{y}) = \mathbf{1}$, then $f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \mathbf{1}$ and there is no need for a recursive call. If $g(\overline{z}, \overline{x}; \mathbf{1}, \overline{y}) \neq \mathbf{1}$ and $g(\overline{z}, \overline{x}; \mathbf{0}, \overline{y}) \neq \mathbf{1}$, then $f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) \neq \mathbf{1}$ and there is no need for a recursive call either. If $g(\overline{z}, \overline{x}; \mathbf{1}, \overline{y}) = \mathbf{1}$ and $g(\overline{z}, \overline{x}; \mathbf{0}, \overline{y}) \neq \mathbf{1}$, then $f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \mathbf{1}$ if and only if

$$\exists c \in \mathbb{K}\ f(\overline{z}, \overline{x}; c.\overline{u}, \overline{y}) = \mathbf{0}.$$

If $g(\overline{z}, \overline{x}; \mathbf{1}, \overline{y}) \neq \mathbf{1}$ and $g(\overline{z}, \overline{x}; \mathbf{0}, \overline{y}) = \mathbf{1}$, then $f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \mathbf{1}$ if and only if

$$\forall c \in \mathbb{K}\ f(\overline{z}, \overline{x}; c.\overline{u}, \overline{y}) \neq \mathbf{0}.$$

Therefore, at every level of the recursion, the choice is determined by the function g. By induction hypothesis, this can be done in $\mathrm{FPAT}_{\mathcal{K}}$. When unfolding the recursion, we get a sequence of quantifiers $Q_1, \ldots, Q_{|\overline{z}|+1}$ and a relation symbol $r \in \{=, \neq\}$ such that

$$f(a.\overline{z}, \overline{x}; \overline{u}, \overline{y}) = \mathbf{1} \text{ iff } Q_1 c_1 \in \mathbb{K}, \ldots, Q_{|\overline{z}|+1} c_{|\overline{z}|+1} \in \mathbb{K}\ h(\overline{x}; c_1 . \ldots . c_{|\overline{z}|+1}.\overline{u}, \overline{y})\ r\ \mathbf{0}.$$

Apply the induction hypothesis on function h. Then, f belongs to $\mathrm{FPAT}_{\mathcal{K}}^{\mathrm{FPAT}_{\mathcal{K}}}$, with an oracle which computes g and gives the quantifier sequence. One just needs to note that $\mathrm{FPAT}_{\mathcal{K}}^{\mathrm{FPAT}_{\mathcal{K}}} = \mathrm{FPAT}_{\mathcal{K}}$ to conclude.

□

7.5 A Characterization of $\mathrm{DPAT}_{\mathcal{K}}$

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

Definition 7.5.1 (Digital Predicative Substitution)

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 substitution*,

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

Definition 7.5.2 (Safe Recursive Functions with Digital Predicative Substitution)

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

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

The set $\mathbf{\ni}^{[1]}_{\mathsf{D}}\mathrm{PAT}_{\mathcal{K}}$ 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.

Again, the proof of Theorem 7.4.1 yields, *mutatis mutandis*, the following result.

Theorem 7.5.1 *A function is computed in* $\mathrm{DFPAT}_{\mathcal{K}}$ *if and only if it can be defined in* $\mathbf{\ni}^{[1]}\mathrm{DPAT}_{\mathcal{K}}$.

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

Corollary 7.5.1 *A decision problem is decided in* PSPACE *if and only if its characteristic function can be defined in* $\mathbf{\ni}^{[1]}\mathrm{DPAT}_{\mathbb{Z}_2}$.

7.6 An Alternative Characterization of $\mathrm{DPAT}_{\mathcal{K}}$

7.6.1 Safe Recursive Functions with Digital Substitutions

When restricted to finite structure, $\mathrm{DPAT}_{\mathcal{K}}$ coincides with $\mathrm{PAR}_{\mathcal{K}}$ and with PSPACE. However, when $\mathcal{K}$ is arbitrary, we only have the inclusion $\mathrm{DPAT}_{\mathcal{K}} \subset \mathrm{PAR}_{\mathcal{K}}$. 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 7.6.1 (Pseudo Logical Functions)

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

A pseudo logical function allows no insight further than the first letter of its arguments: no recursion scheme is allowed, no tl function either. It is therefore computable in constant time, and its output depends only on the value of the first letter of these arguments; more precisely, on whether these letters are $\mathbf{1}$ or not, since the only branching node in the computation correspond to the Select function.

Definition 7.6.2 (Safe Recursive Functions with Digital Substitutions) 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 denote as the set of *safe recursive functions with digital substitutions* the closure of the basic safe functions under the application of safe composition, safe recursion and safe recursion with digital substitutions.

We state our last result:

Theorem 7.6.1 *A function is in* $\mathrm{DFPAT}_{\mathcal{K}}$ *if and only if it is defined as a safe recursive functions with digital substitutions over* $\mathcal{K}$.

Remark 7.6.1 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 [BCdNM03a], and captures PSPACE.

Proof. Let F be a function in $\mathrm{DFPAT}_{\mathcal{K}}$, and denote by G the associate oracle in $\mathrm{DPAT}_{\mathcal{K}}$. There exists a polynomial time BSS machine M over $\mathcal{K}$, and a polynomial function $q : \mathbb{N} \to \mathbb{N}$ such that, for any $\overline{x} \in \mathbb{K}^*$, the following propositions are equivalent:

- (i): $\overline{x} \in G$
- (ii): $\exists b_1 \in \{\mathbf{0}, \mathbf{1}\}\ \forall c_1 \in \{\mathbf{0}, \mathbf{1}\} \ \ldots\ \exists b_{q(|\overline{x}|)} \in \{\mathbf{0}, \mathbf{1}\}\ \forall c_{q(|\overline{x}|)} \in \{\mathbf{0}, \mathbf{1}\}$ M accepts $(\overline{x}, b_1.c_1 \ldots b_{q(|\overline{x}|)}.c_{q(|\overline{x}|)})$

Theorem 6.2.1 ensures that there exists a safe recursive function f_M over $\mathcal{K}$ such that, for any $(\overline{x}, \overline{y}) \in (\mathbb{K}^*)^2$, M accepts on input $(\overline{x}, \overline{y})$ if and only if $f_M(\overline{x}; \overline{y}) = \mathbf{1}$. Moreover, define $g_q : \mathbb{K}^* \times \emptyset \to \mathbb{K}^*$ such that $g_q(\overline{x};) = (\mathbf{1.0})^{q(|\overline{x}|)}$. The existence of such a g_q is once again given by Theorem 6.2.1. This function g_q actually gives the type of the quantifier at every level of the quantifier alternation for any input $\overline{x}$ to the problem G.

Define now the following function,

$$\begin{array}{rcl} F_G(\epsilon, \overline{x}; \overline{u}) & = & f_M(\overline{x}; \overline{u}) \\ F_G(a.\overline{z}, \overline{x}; \overline{u}) & = & \left\{ \begin{array}{ll} F_G(\overline{z}, \overline{x}; \mathbf{1}.\overline{u}) = \mathbf{1} \vee F_G(\overline{z}, \overline{x}; \mathbf{0}.\overline{u}) = \mathbf{1} & \text{if } \mathsf{hd}(;\overline{z}) = \mathbf{1} \\ F_G(\overline{z}, \overline{x}; \mathbf{1}.\overline{u}) = \mathbf{1} \wedge F_G(\overline{z}, \overline{x}; \mathbf{0}.\overline{u}) = \mathbf{1} & \text{otherwise} \end{array} \right. \end{array}$$

The formal definition with safe recursion with digital substitutions of F_G is as follows,

$$\begin{array}{rl} F_G(\epsilon, \overline{x}; \overline{u}) = & f_M(\overline{x}; \overline{u}) \\ F_G(a.\overline{z}, \overline{x}; \overline{u}) = & \mathsf{Select}\,(; \mathsf{hd}(;\overline{z}), \mathsf{Select}\,(; F_G(\overline{z}, \overline{x}; \mathsf{cons}(; \mathbf{1}, \overline{u})), \mathbf{1}, \\ \multicolumn{2}{l}{F_G(\overline{z}, \overline{x}; \mathsf{cons}(; \mathbf{0}, \overline{u})))\,, \mathsf{Select}(; F_A(\overline{z}, \overline{x}; \mathsf{cons}(; \mathbf{1}, \overline{u})), F_G(\overline{z}, \overline{x}; \mathsf{cons}(; \mathbf{0}, \overline{u})), \mathbf{0}))} \end{array}$$

It is clear from the definition that $F_G(\mathsf{cons}(\mathbf{1}, g_S(\overline{x};);), \overline{x}; \mathbf{0})$ decides whether $\overline{x}$ belongs to G.

It follows that F belongs to the set of safe recursive functions with digital substitutions.

The other direction of the proof is done by induction on the definition of F. The only critical case is when function F is defined with safe recursion with digital substitutions:

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

The result follows from the following lemma.

Lemma 7.6.1 *The relation* $F(a.\overline{z}, \overline{x}; \overline{u}) = \mathbf{1}$ *can be reduced in polynomial time to a decision problem in* $(\mathrm{D}\Sigma_{\mathcal{K}}^{2|\overline{z}|+2})^H$ *where* H *is an oracle deciding* $h(\overline{y}; \overline{v}) = \mathbf{1}$.

□

Proof. By induction on $|\overline{z}|$. For $\overline{z} = \epsilon$,it is a consequence of Theorem 6.2.1.

Assume $\overline{z} \neq \epsilon$, and define:

$$\begin{array}{rcl} a_1 & = & \left\{ \begin{array}{ll} \mathbf{1} & \text{if } \mathsf{hd}(; F(\overline{z}, \overline{x}; \sigma_1(;\overline{u}))) = \mathbf{1} \\ \mathbf{0} & \text{otherwise} \end{array} \right. \\ a_2 & = & \left\{ \begin{array}{ll} \mathbf{1} & \text{if } \mathsf{hd}(; F(\overline{z}, \overline{x}; \sigma_2(;\overline{u}))) = \mathbf{1} \\ \mathbf{0} & \text{otherwise} \end{array} \right. \end{array}$$

Since g is a pseudo logical function, it is computable in constant time, and the value of $F(a.\overline{z}, \overline{x}; \overline{u}) = g(\overline{z}, \overline{x}; F(\overline{z}, \overline{x}; \sigma_1(;\overline{u})), F(\overline{z}, \overline{x}; \sigma_2(;\overline{u})))$ depends only on the relations

$\mathsf{hd}(;F(\overline{z},\overline{x};\sigma_1(;\overline{u}))) = \mathbf{1}$ and $\mathsf{hd}(;F(\overline{z},\overline{x};\sigma_2(;\overline{u}))) = \mathbf{1}$. Thus,

$$g(\overline{z},\overline{x};F(\overline{z},\overline{x};\sigma_1(;\overline{u})),F(\overline{z},\overline{x};\sigma_2(;\overline{u}))) = g(\overline{z},\overline{x};a_1,a_2).$$

Define

$$F'(\overline{z},\overline{x};\overline{c},\overline{u}) = \begin{cases} F(\overline{z},\overline{x};\sigma_1(;\overline{u})) & \text{if } \mathsf{hd}(;\overline{c}) = \mathbf{1} \\ F(\overline{z},\overline{x};\sigma_2(;\overline{u})) & \text{otherwise} \end{cases}$$

Consider now the four possible values for (a_1, a_2). The relation $F(a.\overline{z},\overline{x};\overline{u}) = \mathbf{1}$ is given by:

- $(a_1 = \mathbf{1}, a_2 = \mathbf{1})$: $\forall c \in \{\mathbf{1},\mathbf{0}\}\ (g(\overline{z},\overline{x};a_1,a_2) = \mathbf{1}) \wedge (F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})$
- $(a_1 = \mathbf{1}, a_2 = \mathbf{0})$: $\forall c \in \{\mathbf{1},\mathbf{0}\}\ (g(\overline{z},\overline{x};a_1,a_2) = \mathbf{1}) \wedge (c = \mathbf{1} \Leftrightarrow F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})$
- $(a_1 = \mathbf{0}, a_2 = \mathbf{1})$: $\forall c \in \{\mathbf{1},\mathbf{0}\}\ (g(\overline{z},\overline{x};a_1,a_2) = \mathbf{1}) \wedge (c = \mathbf{0} \Leftrightarrow F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})$
- $(a_1 = \mathbf{0}, a_2 = \mathbf{0})$: $\forall c \in \{\mathbf{1},\mathbf{0}\}\ (g(\overline{z},\overline{x};a_1,a_2) = \mathbf{1}) \wedge (F'(\overline{z},\overline{x};c,\overline{u}) \neq \mathbf{1})$

Note that the logical operations "$\wedge, \vee, \Leftrightarrow$" can be easily computed with the basic function Select, projections and safe composition.

The relation $F(a.\overline{z},\overline{x};\overline{u}) = \mathbf{1}$ is therefore given by

$$\begin{aligned} \exists a_1, a_2 \in \{\mathbf{0},\mathbf{1}\}, \forall c \in \{\mathbf{0},\mathbf{1}\} \qquad & (g(\overline{z},\overline{x};a_1,a_2) = \mathbf{1}) \\ \bigwedge \quad & [((a_1 = \mathbf{1} \wedge a_2 = \mathbf{1}) \wedge (F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})) \\ & \vee((a_1 = \mathbf{1} \wedge a_2 = \mathbf{0}) \wedge (c = \mathbf{1} \Leftrightarrow F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})) \\ & \vee((a_1 = \mathbf{0} \wedge a_2 = \mathbf{1}) \wedge (c = \mathbf{0} \Leftrightarrow F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1})) \\ & \vee((a_1 = \mathbf{0} \wedge a_2 = \mathbf{0}) \wedge (F'(\overline{z},\overline{x};c,\overline{u}) \neq \mathbf{1}))] \end{aligned}$$

Apply the induction hypothesis: $F(\overline{z},\overline{x};\overline{u}) = \mathbf{1}$ can be reduced to a decision problem in $(\mathrm{D}\Sigma_{\mathcal{K}}^{2|\overline{z}|})^H$, and therefore $F'(\overline{z},\overline{x};c,\overline{u}) = \mathbf{1}$ can also be reduced to the same problem, which ends the proof.

$\square$

Conclusion

We have given several completeness results for natural geometrical and topological problems in the context of BSS computation over the real numbers with addition and order. These completeness results cover the major complexity classes in this setting. However, the complexity of some interesting problems remains unknown. As a perspective, one could study the complexity of deciding wether a semilinear set is a manifold, and the complexity of deciding wether a semilinear manifold is orientable.

We have also characterized deterministic and non-deterministic polynomial time classes $\mathrm{P}_{\mathcal{K}}$ and $\mathrm{NP}_{\mathcal{K}}$ over an arbitrary structure $\mathcal{K}$ in descriptive complexity, extending some results of [GM95, CM99, GG98]. A similar approach could be used to characterize other classes, such as $\mathrm{PAR}_{\mathcal{K}}$, $\mathrm{EXP}_{\mathcal{K}}$ and $\mathrm{NC}^k_{\mathcal{K}}$, along the lines of [CM99] where such characterizations are given for the real numbers.

We have also given some characterizations of complexity classes over an arbitrary structure $\mathcal{K}$ in implicit complexity, extending many classical results. Our characterizations cover the classes $\mathrm{FP}_{\mathcal{K}}$, $\mathrm{FPAR}_{\mathcal{K}}$, $\mathrm{FPAT}_{\mathcal{K}}$, $\mathrm{FP}_{\mathcal{K}}^{\mathrm{NP}_{\mathcal{K}}}$ and other levels of the polynomial hierarchy, as well as their digital versions. It could be intersting to search for similar results for subpolynomial complexity classes.

Bibliography

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, MA, 1994.

[ASV90] S. Abiteboul, S. Simon, and V. Vianu. Non-deterministic languages to express deterministic transformations. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1990.

[AV89] S. Abiteboul and V. Vianu. Fixed point extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 71–79, Washington, D.C., 1989. IEEE Computer Society Press.

[AV91] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

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

[BC04] P. Bürgisser and F. Cucker. Counting complexity classes for numeric computations I: Semilinear sets. SIAM J. Comp., 33:227–260, 2004.

[BCdNM] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J. Y. Marion. Implicit complexity over an arbitrary structure: Quantifier alternations. Submitted to Information and Computation.

[BCdNM02] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J. Y. Marion. Safe recursion and calculus over an arbitrary structure. In *Federated Logic Conference*, 2002.

[BCdNM03a] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J.-Y. Marion. Computability over an arbitrary structure. sequential and parallel polynomial time. In

Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference (FOSSACS'2003)*, volume 2620 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2003.

[BCdNM03b] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J. Y. Marion. Safe recursion over an arbitrary structure: PAR, PH and DPH. In *Federated Logic Conference*, 2003.

[BCdNM04a] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J. Y. Marion. Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time. *Journal of Logic and Computations*, 2004. To appear.

[BCdNM04b] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J. Y. Marion. Tailoring recursion to characterize non-deterministic complexity classes over arbitrary structures. In E. W. Mayr J-J. Levy and J. C. Mitchell, editors, *"Exploring new Frontiers of Theoretical Informatics", proceedings of the 3rd International Conference on Theoretical Computer Science*. Kluwer Academic Publishers, 2004.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[Bel92] S. Bellantoni. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.

[Bel94] S. Bellantoni. Predicative recursion and the polytime hierarchy. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II, Perspectives in Computer Science*. Birkhaüser, 1994.

[BG00] A. Blass and Y. Gurevich. The logic of choice. *Journal of Symbolic Logic*, 65(3):1264–1310, Sept. 2000.

[BH88] S. R. Buss and L. Hay. On truth-table reducibility to SAT and the difference hierarchy over NP. In *Proc. 3rd Symp. on Structure in Complexity Theory*, pages 224–233, 1988.

[Blo92] S. Bloch. Functional characterizations of uniform log-depth and polylog-depth circuit families. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 193–206. IEEE Computer Society Press, 1992.

[Bon88] A. J. Bonner. Hypothetical datalog: complexity and expressibility. In *Proceedings of the International Conference on Database Theory*, volume 326 of *LNCS*, pages 144–160, Berlin, 1988.

[BS90] R. Boppana and M. Sipser. *Handbook of theoretical computer science*, volume A, chapter The complexity of finite functions. Elsevier Science, 1990.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Soc.*, 21:1–46, 1989.

[CFGK03] A. Chistov, H. Fournier, L. Gurvits, and P. Koiran. Vandermonde matrices, NP-completeness, and transversal subspaces. *Foundations of Computational Mathematics*, 3(4):421–427, 2003.

[Chu41] A. Church. The calculi of lambda-conversion. In *Annals of Mathematical Studies*, volume 6, Princeton, N.J., 1941. Princeton Univ. Press.

[CK95] F. Cucker and P. Koiran. Computing over the reals with addition and order: Higher complexity classes. *Journal of Complexity*, 11:358–376, 1995.

[CKM97] F. Cucker, P. Koiran, and M. Matamala. Complexity and dimension. *Information Processing Letters*, 62:209–212, 1997.

[CL90] K. Compton and C. Laflamme. An algebra and a logic for NC^1. *Information and Computation*, 87:241–263, 1990.

[Clo95] P. Clote. Computational models and function algebras. In D. Leivant, editor, *LCC'94*, volume 960 of *Lect. Notes in Comp. Sci.*, pages 98–130. Springer-Verlag, 1995.

[CM99] F. Cucker and K. Meer. Logics which capture complexity classes over the reals. *J. of Symb. Logic*, (64):363–390, 1999.

[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.

[Coo71] S. Cook. The complexity of theorem proving procedures. In *ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Coo92] S. 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.

[CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. SIAM J. Comp., 13:423–439, 1984.

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

[Cuc93] F. Cucker. On the complexity of quantifier elimination: the structural approach. *The Computer Journal*, 36:400–408, 1993.

[dR93] M. de Rougemont. *Logique et fondements de l'informatique*. Number ISBN 2-86601-380-8. Hermès, 1993.

[dR95] M. de Rougemont. *Logique et Complexité*. Number ISBN 2-86601-496-0. Hermès, 1995.

[dR03a] M. de Rougemont. *Discrete Mathematics and Theoretical Computer Science*. Springer-Verlag, 2003.

[dR03b] M. de Rougemont. *Logic and Complexity*. Springer-Verlag, 2003.

[EF95] H.-D. Ebbinghaus and J. 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.

[FK98] H. Fournier and P. Koiran. Are lower bounds easier over the reals? In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 507–513, Dallas, TX, 1998.

[FK00] H. Fournier and P. Koiran. Lower bounds are not easier over the reals: Inside PH. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 832–843, Geneva, Switzerland, 2000. Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin.

[Gak96] J. S. Gakwaya. Extended Grzegorczyk hierarchy in the BSS model of computability. Technical Report NC-TR-96-049, NeuroCOLT Technical Report Series, 1996.

[Gak97] J. S. Gakwaya. A survey on the Grzegorczyk hierarchy and its extension through the BSS model of computability. Technical Report NC-TR-97-041, NeuroCOLT Technical Report Series, 1997.

[Gat86] J. V. Z. Gathen. Parallel arithmetic computations: A survey. In J. Gruska and B. Rovan, editors, *Proceedings of the 12th Annual Symposium on Mathematical Foundations of Sciences*, pages 93–112, Bratislava, Czechoslovakia, 1986. Lecture Notes in Comput. Sci., Springer Verlag, Berlin.

[GG95] Y. Gurevich and E. Grädel. Tailoring recursion for complexity. *Journal for Symbolic Logic*, 60:952–969, 1995.

[GG98] E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998.

[GJ79] M. Garey and D. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* Freeman, 1979.

[GM95] E. Grädel and K. Meer. Descriptive complexity over the real numbers. In *Proceedings of the 27th Annual ACM Symp. on the Theory of Computing*, pages 315–324, 1995.

[Goe89] A. Goerdt. Characterizing complexity classes by general recursive definitions in higher types. In *Proceedings of 2nd workshop on Computer Science logic CSL'88*, volume 385 of *Lecture Notes in Computer Science*, pages 99–117. Springer Verlag, 1989.

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

[Grz53] A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Mate. IV*, 1953. Warsaw.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

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

[Gur88] Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

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

[IKR01] R. Irwin, B. Kapron, and J. Royer. On characterizations of the basic feasible functionals. *J. of Functional Programming*, 11:117–153, 2001.

[Imm83] N. Immerman. languages which capture complexity classes. In *ACM Symposium on Theory of Computing*, pages 347–354, 1983.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[Imm91] N. Immerman. Dspace$[n^k]$ = var$[k+1]$. In *Proc. of the 6th IEEE Symp. on Structure in Complexity Theory*, pages 334–340, 1991.

[Imm99] N. Immerman. *Descriptive complexity.* Springer Verlag, 1999.

[Jon97] N. Jones. Computability and complexity, from a programming perspective. MIT Press, 1997.

[Jon99] N. Jones. Logspace and ptime characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[Jon01] N. Jones. The expressive power of higher order types. *J. of Functional Programming*, 11:55–94, 2001.

[Kar72] R. Karp. Reducibility among combinatorial problems. In J. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 1–20, 1972.

[Kle36] S. C. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.

[Koi94] P. Koiran. Computing over the reals with addition and order. *Theoret. Comput. Sci.*, 133(1):35–48, 1994.

[Koi99] P. Koiran. The real dimension problem is NP_R complete. *Journal of Complexity*, 15:227–238, 1999.

[Koi00] P. Koiran. The complexity of local dimensions for constructible sets. *Journal of Complexity*, 16:311–323, 2000.

[Koi02] P. Koiran. Transfer theorems via sign conditions. *Information Processing Letters*, 81(65-69), 2002.

[Kre86] M. W. Krentel. The complexity of optimization problems. In *Proc. 18th ACM Symp. on the Theory of Computing*, pages 79–86, 1986.

[Lei90a] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.

[Lei90b] D. Leivant. Subrecursion and lambda representation over free algebras. In S. Bluss and P. Scott, editors, *Feasible Mathematics*, Perspectives in Computer Science, pages 281–291. Birkhauser-Boston, New York, 1990.

[Lei93] D. Leivant. Stratified functional programs and computational complexity. In *Conference Record of the Twentieth Annual ACM Symposium on the Principles of Programming Languages*, New York, 1993. ACM.

[Lei94a] D. Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

[Lei94b] D. Leivant. Predicative recurrence in finite type. In A. Nerode and V. Yu, editors, *Logical Foundations of Computer Science*, volume LNCS 813, pages 227–239. Springer Verlag, Berlin, 1994. Proceedings of the Third LFCS Symposium, St. Petersburg.

[Lei95] D. Leivant. Intrinsic theories and computational complexity. In *LCC'94*, volume 960 of *Lect. Notes in Comp. Sci.*, pages 177–194. Springer-Verlag, 1995.

[Lev73] L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[LM93] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, September 1993.

[LM95] D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL'94*, volume 933 of *Lect. Notes in Comp. Sci.*, pages 369–380, Kazimierz, Poland, 1995. Springer-Verlag.

[LM00] D. Leivant and J. Y. Marion. A characterization of alternating log time by ramified recurrence. *Theoretical Computer Science*, 236(1-2):192–208, 2000.

[Lo92] L. Lo. Functions and functionals on finite systems. *Journal of Symbolic Logic*, 57:118–130, 1992.

[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.

[Mee00] K. Meer. Counting problems over the reals. *Theoret. Comput. Sci.*, 242:41–58, 2000.

[Mey84] F. Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. Assoc. Comput. Mach.*, 31:668–676, 1984.

[Mey88] F. Meyer auf der Heide. Fast algorithms for n-dimensional restrictions of hard problems. *J. Assoc. Comput. Mach.*, 35:740–747, 1988.

[Mic89] C. Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *C. R. Acad. Sci. Paris*, 309, Série I:435–437, 1989.

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

[Mos83] Y. Moschovakis. Abstract recursion as the foundation for a theory of algorithms. In *Computation and Proof Theory (Proc. of the ASL Meeting in Aachen*, Lecture notes in Mathematics. Springer Verlag, Berlin, 1983.

[Ngu93] A. P. Nguyen. A formal system for linear space reasoning. Master of science thesis, University of Toronto, 1993.

[Pap83] C. H. Papadimitriou. The complexity of unique solutions. In *Proc. 23rd IEEE Symp. on the Foundations of Computer Science*, pages 14–20, 1983.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pet66] R. Peter. *Rekursive Funktionen*. Akadémiai Kiadò, Budapest, 1966. English translation: *Recursive Functions*, Academic Press, New York, 1967.

[Poi95] B. Poizat. *Les Petits Cailloux*. Aléas, 1995.

[Rit63] R. W. Ritchie. Classes of predictably computable functions. *Trans. A.M.S.*, 106:139–173, 1963.

[Ros84] H.E. Rose. *Subrecursion*. Oxford Univ. Press, 1984.

[Sav72] J. Savage. Computational work and time on finite machines. *Journal of the Association of Computing Machinery*, 19:660–674, 1972.

[Sav76] J. Savage. *The complexity of computing*. 1976.

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

[Sha49] C. Shannon. The synthesis of two-terminal networks. *Bell System Technical Journal*, 28:59–98, 1949.

[Sim88] H. Simmons. The realms of primitive recursion. *Archive for Mathematical Logic*, 27:177–, 1988.

[Sto76] L. J. Stockmeyer. The polynomial hierarchy. *Theor. Comp. Science*, 3:1–22, 1976.

[Tar86] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Oper. Res.*, 34:250–256, 1986.

[Tod90] S. Toda. On the computational power of PP and $\oplus$P. In *Proc. 30th IEEE Symp. on the Foundations of Computer Science*, pages 26–35, 1990.

[Tur36] A. M. Turing. On computable numbers, with an application to the *entscheidungsproblem.* In *Proc. London Math. Society*, volume 42 of *2*, pages 230–265, 1936.

[TW92] S. Toda and O. Wanatabe. Polynomial time 1-Turing reductions from #PH to #P. *Theoret. Comput. Sci.*, 100:205–221, 1992.

[Val79a] L. Valiant. The complexity of computing the permanent. *Theort. Comput. Sci.*, 8:189–201, 1979.

[Val79b] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Computing*, 8:410–421, 1979.

[Var82] M. Vardi. Complexity of relational query languages. In *ACM Symposium on Theory of Computing*, pages 137–146, 1982.

[Weg87] I. Wegener. *The complexity of boolean functions.* Woley-Teubner, 1987.

[ZS79] O. Zariski and P. Samuel. *Commutative Algebra*, volume 1 and 2. Springer-Verlag, 1979. Reprint of the 1958-1960 edition.