Olivier Bournez

Version of October 28, 2021

Chapter 1

Preliminaries

These are Course Notes for MPRI Course 2.33.1.
Theories of Computation.

Any comment (even about orthography) welcome: send an email to bournez@lix.polytechnique.fr

Contents

Chapter 2

Computing over an arbitrary structure

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

We consider here the BSS model of computation over the real numbers introduced by Blum, Shub and Smale in their seminal paper [Blum et al., 1989]. The model was later on extended to a computational model over any arbitrary logical structure [Goode, 1994, Poizat, 1995a]. Refer to the monograph [Blum et al., 1998] for a general survey about the BSS model.

2.1 The Model

We introduce computability and complexity over an arbitrary structure. Detailed accounts can be found in [Blum et al., 1998] —for structures like real and complex numbers– or [Poizat, 1995a] –for considerations about more general structures.

Definition 2.1 *A* structure $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i\in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ *is given by some underlying set* $\mathbb{K}$*, some operations (i.e.* functions*)* $\{op_i\}_{i\in I}$*, and a finite number of relations* $r_1, \ldots, r_\ell$*. Constants correspond to operators of arity 0. While the index set I may be infinite, the number of operators with arity greater or equal to 1 needs to be finite, that is, only symbols for constants may be infinitely many.*

We will not distinguish here between operator and relation symbols and their corresponding interpretations as functions and relations respectively over the underlying set $\mathbb{K}$. We assume that the equality relation = is a relation of the structure,

and that there are at least two constant symbols, with different interpretations (denoted by **0** and **1**) in the structure.

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

Remark 2.1 *For any structure $\mathcal{K}$ as above, $(\{0,1\}, =, \mathbf{0}, \mathbf{1}) \subseteq \mathcal{K}$.*

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 $\mathbb{K}_*$ (this will use basically to represent "*tapes*"). 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

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

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

Example 2.1 *The word $\pi e \sqrt{\pi} 2$ is of length 4.*

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

Definition 2.2 *A* machine over $\mathcal{K}$ *consists of an input space $\mathcal{I} = \mathbb{K}^*$, an output space $\mathcal{O} = \mathbb{K}^*$, and a register space[a] $\mathcal{S} = \mathbb{K}_*$, together with a connected directed graph whose nodes labelled $0, \ldots, N$ correspond to the set of different* instructions *of the machine. These nodes are of one of the five following types:* input, output, computation, branching *and* shift *nodes. Let us describe them a bit more.*

1. Input nodes. *There is only one input node and is 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 $g_m : \mathcal{S} \to \mathcal{S}$ and some integer i. The function g_m replaces the component indexed by i of $\mathcal{S}$ by the value $op(w_1, \ldots, w_n)$ where $w_1, w_2, \ldots, w_n$ are components 1 to n of $\mathcal{S}$ and op is some operation of the structure $\mathcal{K}$ of arity n.*

   The other components of $\mathcal{S}$ are left unchanged. When the arity n is zero, m is called a constant node.

Remark 2.2 *A given machine uses only a finite number of constants, since it has finitely many instructions.*

[a] 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.

Remark 2.3 *In order to compare different machines and to denote the notion of reduction between them and completeness, one needs to include all possible constants in the underlying structure $\mathcal{K}$. Thus the possibly infinite index set I.*

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

5. Shift nodes. *Associated with a node m of this type there is a next node $\beta(m)$ and a map $\sigma : \mathcal{S} \to \mathcal{S}$. The 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 [Blum et al., 1998, Blum et al., 1989, Poizat, 1995a]. We will not dwell on these details but focus on the essential ideas in the proofs to come in the sequel: A simple convention is the following: $g_I : \mathcal{I} = \mathbb{K}^* \to \mathcal{S}$ with $g_I(w) = n.w_1 w_2 \dots w_n$ if word $w = w_1 w_2 \dots w_n$ is of length n.

Remark 2.4 *A machine over $\mathcal{K}$ is essentially a Turing Machine, which is able to perform the basic operations $\{op_i\}$ and the basic tests $r_1, \dots, r_\ell$ at unit cost, and whose tape cells can hold arbitrary elements of the underlying set $\mathbb{K}$ [Poizat, 1995a, Blum et al., 1998]. Note that the register space $\mathcal{S}$ above has the function of the tape and that its component with index 1 plays the role of the scanned cell. 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. In particular, a cell of the tape will sometimes also be called a* register.

Definition 2.3 *For a given machine M, the function φ_M associating its output to a given input $x \in \mathbb{K}^*$ is called the* input-output function. *We shall say that a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is* computable *when there is a machine M such that $f = \varphi_M$.*

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

We can now define some central complexity classes.

Definition 2.4 *A set $S \subset \mathbb{K}^*$ is in class $\mathrm{P}_{\mathcal{K}}$ (respectively a function $f : \mathbb{K}^* \to \mathbb{K}^*$ is in class FP_K), if there exist a polynomial p and a machine M, so that for all $\overline{w} \in \mathbb{K}^*$, M stops in at most $p(|\overline{w}|)$ steps and M accepts iff $\overline{w} \in S$ (respectively, M computes function $f(\overline{w})$).*

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

Proposition 2.1 **(i)** *The class $\mathrm{P}_{\mathcal{K}}$ is the classical* P *when $\mathcal{K} = (\{0,1\}, =, \mathbf{0}, \mathbf{1})$.*

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

2.2 Non-determinism

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

2.2.1 Full non-determinism

One can introduce:

- 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 $\langle \overline{x}, \overline{y} \rangle$ is 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 $|\overline{y}| \leq p_B(|\overline{x}|)$, $\langle \overline{x}, \overline{y} \rangle$ is in B.

2.2.2 Digital non-determinism

One can also introduce:

- A decision problem A is in $\mathrm{NDP}_{\mathcal{K}}$ 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 $|\overline{y}| \leq p_B(|\overline{x}|)$ satisfying $\langle \overline{x}, \overline{y} \rangle$ is in B.

- A decision problem A is in $\mathrm{coNDP}_{\mathcal{K}}$ 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 $|\overline{y}| \leq p_B(|\overline{x}|)$, $\langle \overline{x}, \overline{y} \rangle$ is in B.

2.3 Space complexity?

One may want not only to talk about time, but other resources such as *space*. However, a result by Christian Michaux [Michaux, 1989] shows the irrelevancy of this notion in a broad context, and hence the difficulty in defining easily a counterpart for classical class PSPACE, at least when talking about structures over the reals:

Theorem 2.1 *In the BSS-model over the structure* $(\mathbb{R}, +, -, x \mapsto x/2, =, <)$*, every polynomial time decision problem can be solved by an algorithm working in polynomial time needing constant additional space.*

Chapter 3

Non Uniform Polynomial Time

3.1 Class $\mathrm{P}_{\mathrm{poly}}$

In order to talk about polynomial time, we need to talk about complexity class $\mathrm{P}_{\mathrm{poly}}$.

In computational complexity theory, an *advice* string is an extra input to a Turing machine which is allowed to depend on the length n of the input, but not on input itself. A decision problem is in the complexity class $P/f(n)$ if there is a polynomial time Turing machine M with the following property: for any n, there is an advice string A of length $f(n)$ such that, for any input x of length n, the machine M correctly decides the problem on the input x, given x and A.

$\mathrm{P}_{\mathrm{poly}}$ is obtained by considering there case where functions f correspond to polynomials.

More formally, this correspond to the following definition.

Definition 3.1 ($\mathrm{P}_{\mathrm{poly}}$) $\mathrm{P}_{\mathrm{poly}}$ *is the class of languages B such that there is some language A recognised in polynomial time (i.e. $A \in \mathrm{P}$), some function $f : \mathbb{N} \to \Sigma^*$ and some polynomial p such that for all n, $|f(n)| \leq p(n)$, and*

$$B = \{x | < x, f(|x|) > \in A\}.$$

3.2 Relating $\mathrm{P}_{\mathrm{poly}}$ to sparse and tally sets

A set $S \subset \Sigma^*$ is said to be *sparse* if there exists a polynomial $p(n)$ such that for every n, S has less than $p(n)$ words of length $\leq n$.

$\mathrm{P}(S)$ denotes the languages recognised in polynomial time with *oracle* S.

Theorem 3.1 $\mathrm{P}_{\mathrm{poly}} = \bigcup_{S\ sparse} \mathrm{P}(S)$.

Proof: We prove that a set L is in $\mathrm{P}_{\mathrm{poly}}$ iff there is a sparse set S such that $L \in P(S)$.

Assume that $L \in \mathrm{P}_{\mathrm{poly}}$ via the advice function f and the set $A \in \mathrm{P}$. Define S as follows:

$$S = \{< 0^n, w > | w \text{ is a prefix of } f(n) >\}.$$

S is sparse: each word in S of length m is of the form $< 0^n, w >$ for $n \leq m$. There are $m+1$ possible values of n. Each of them contributes at most $m+1$ different prefixes of $f(n)$, one for each length up to m. The total number of words of length m is at most $O(m^2)$.

Now L is in $P(S)$, as using S as oracle, one can easily build the advice z (extending bit by bit z by 0 or 1's, querying for each bit whether it should be 0 or 1), and then use it to compute L.

Conversely, assume that $L \in P(S)$, where S is sparse. Let p be the polynomial bounding the running time of the machine that decides L. Define the advice function such that, for each n, it gives the encoding of the set of words in S up to size $p(n)$. This is a polynomially long encoding. Using this advice, this is easy to simulate queries to S in polynomial time. □

A set $S \subset \Sigma^*$ is said to be *tally* if $S \subset \{a\}^*$ for some symbol a.

It is possible to prove the following (left as an exercice):

Theorem 3.2 $\mathrm{P}_{\mathrm{poly}} = \bigcup_{S\ tally} \mathrm{P}(S)$.

Proof: <Left as an exercice> □

3.3 Known Facts about $\mathrm{P}_{\mathrm{poly}}$

- $\mathrm{P}_{\mathrm{poly}}$ contains some non-computable languages: consider some non-computable $A \subset \mathbb{N}$, and consider the language A made of words of length n with $n \in A$. A is non-computable, as A is. By definition, it is indeed in P poly, as the function that maps n to 0 or 1 according to whether $n \in A$ or not is a valid advice function.
- $\mathrm{P}_{\mathrm{poly}}$ contains P and BPP (Adleman's theorem).
- If NP $\subset \mathrm{P}_{\mathrm{poly}}$ then the polynomial hierarchy collapses to Σ_2^P (Karp-Lipton's theorem).

3.4 Relating $\mathrm{P}_{\mathrm{poly}}$ to boolean circuits

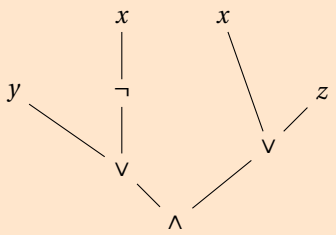
3.4.1 Boolean circuits

One can see a boolean circuit as a mean to describe a boolean function as a sequence of OR ($\vee$), AND ($\wedge$) and NOT ($\neg$) on bits given as input.

Definition 3.2 (Boolean circuit) *Let n be an integer. A* boolean circuit *with n inputs and m output is a DAG (directly oriented graph) with*

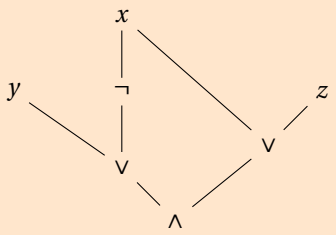
- n inputs, *that is to say n vertices without ingoing arc.*
- *m outputs, that is to say m vertices without outgoing arc*
- *each input is labeled either by constant* 0*, or by* 1 *or by some symbol of variable $x_1, x_2, \cdots, x_n$.*
- *any other vertex is a called a* gate *and is labeled either by $\vee$, $\wedge$ or $\neg$. The* fanin *is the ingoing degree of a gate.*
- *gates labeled by $\vee$ or $\wedge$ have fanin* 2*.*
- *gates labeled by $\neg$ have fanin* 1*.*

Example 3.1 *Here is an example of circuit corresponding to function $S(x, y, z) = (\neg x \vee y) \wedge (x \vee z)$.*

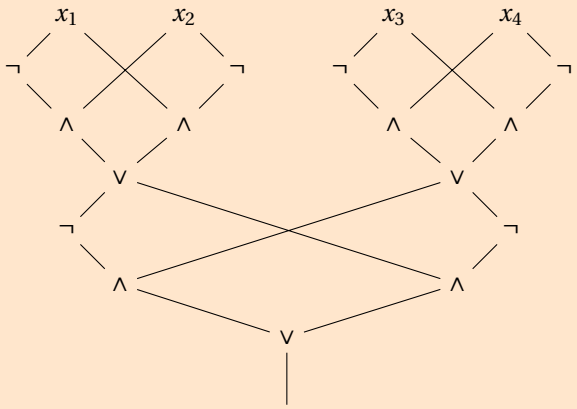


Observe that several inputs can be labeled by a same symbol. We authorise ourselves graphically to share inputs.

Example 3.2 *The circuit of example 3.1 can also be represented as*



Example 3.3 *Here is a less trivial example.*



Given a boolean circuit C with n inputs and m outputs, and $\overline{x} \in \{0,1\}^n$, *the output of* C on $\overline{x} = (x_1, \cdots, x_n)$, written $C(\overline{x})$ is defined inductively as expected:

Definition 3.3 (Boolean function associated to a circuit) *More formally, for every vertex v from C, one defines its value $val(v) \in \{0,1\}$ on $\overline{x} = (x_1, \cdots, x_n)$ as follows:*

- *if v is some input labeled with* 0*, then $val(v) = 0$;*
- *if v is some input labeled with* 1*, then $val(v) = 1$;*
- *if v is some input labeled with a variable x_i, then $val(v) = x_i$;*
- *if v is a gate $\wedge$, then $val(v)$ is the logical conjonction of values $val(e_1)$ and $val(e_2)$ of its inputs e_1 and e_2: $val(v) = \wedge(val(e_1), val(e_2))$;*
- *if v is a gate $\vee$, then $val(v)$ is the logical disjunction of the values $val(e_1)$ ad $val(e_2)$ of its inputs e_1 et e_2: $val(v) = \vee(val(e_1), val(e_2))$;*
- *if v is a gate $\neg$, then $val(v)$ is the logical negation of its input e: $val(v) = \neg(val(e))$.*

The value $C(\overline{x})$ of the circuit is then given by the value $val(v_1), val(v_2) \ldots val(v_m)$ of its m outputs: this is an element of $\{0,1\}^m$

Example 3.4 *The circuit of Example 3.1 computes the function from $\{0,1\}^3$ to $\{0,1\}$ defined by $S(x,y,z) = (\neg x \vee y) \wedge (x \vee z)$,*

Example 3.5 *The circuit of Example 3.3 computes the function $PARITY$ from $\{0,1\}^4$ to $\{0,1\}$ that values* 1 *iff an odd number of its arguments values* 1*.*

The *size* of a circuit C, is the number of vertices of the circuit. Its *depth* is the length of the longest path from an input to the output.

Example 3.6 *The circuit of example 3.3 is of size* 19 *and of depth* 6.

One terms *sub-circuit* of a circuit C what is expected: this is to say, a subset C' of the circuit that has the property that if a gate is among this subset, then any gate with some outgoing link towards this gate is also in this subset. In particular, the *principal sub-circuit* associated to a gate p is formed by the set of the gates of C that are on a path that goes to p. The *immediate sub-circuits* of a circuit C are the principal sub-circuit(s) corresponding to the gate(s) with an outgoing link to the output.

A given circuit with one input recognizes only words over $\{0,1\}$ of fixed length. If one wants to consider recognition of languages, one needs to talk about family of circuits (see e.g. [Balcázar et al., 1988, Papadimitriou, 1994]) : A family of Boolean circuits $\mathscr{C} = (C_i)_{i\in\mathbb{N}}$, with C_i with i inputs and 1 output, recognizes a language $L \subset \Sigma^*$, iff for all $w \in \Sigma^*$, $w \in L$ if and only if $C_{|w|}$ accepts w.

We assume fixed a reasonable way to encode circuits: evaluation of a circuit C on some input x can be done in polynomial time. Denote by CIRCUITVALUE the decision problem that consists in evaluating C on input $\overline{x}$, given C and $\overline{x}$.

3.4.2 Fundamental relation between circuits and computations

Polynomial time can be characterised by circuits:

Theorem 3.3 *[P versus* $\mathrm{P}_{\mathrm{poly}}$ *(see e.g. [Papadimitriou, 1994])] A language* $L \subset \Sigma^*$ *is recognised in polynomial time by a Turing machine, iff*

1. *L is recognised by a family of circuits of polynomial size: there exists some polynomial p, with* $size(C_n) = p(n)$ *for all n.*
2. *the function that maps* 1^n *to the encoding of circuit* C_n *is computable in polynomial time (and even in logarithmic space).*

Proof: The idea of the direct sense of the proof is to see that for a given length, as the Turing machine M works in less than $p(n)$ steps for some polynomial p, using less than $q(n)$ cells of the tape, by unfolding the program of M (as in the proof of Cook's Theorem (NP-completeness of SAT)), one can build a circuit C_n that decides if a word w of length n is accepted by M.

Then to observe that the circuit C_n, being obtained as a simple unfolding of the program of M, the function that maps 1^n to the encoding of circuit C_n is indeed computable in polynomial time.

Conversely, given a word w, one can compute its length $n = |w|$, then C_n and simulates C_n on w. With the two hypotheses, all of this can be done in polynomial time, and hence the language L is in P. □

3.4.3 Non uniform polynomial time

Class $\mathrm{P}_{\mathrm{poly}}$ corresponds to non-uniform polynomial time, since it consists in relaxing second condition in next characterization of polynomial time.

Theorem 3.4 *A language $L \subset \Sigma^*$ is in $\mathrm{P}_{\mathrm{poly}}$ iff L is recognised by a family of circuits of polynomial size: there exists some polynomial p, with $size(C_n) = p(n)$ for all n.*

Proof: Let L be a set with polynomial size circuits. Let C_n be the encoding of the polynomial size circuit that recognises $L \cap \Sigma^n$, where Σ^n is the set of words of length n. We have $|C_n| \leq p(n)$ for some polynomial p. Define the advice function f as $f(n) = C_n$. For any length n, and for any w of length n, we have $w \in L$ iff C_n outputs 1 on input w iff $< w, f(|w|) > \in CVP$, where CVP is the Circuit Value Problem (evaluate the value of a circuit on some input). As CVP is in P, L is in $\mathrm{P}_{\mathrm{poly}}$.

Conversely, let L be in $\mathrm{P}_{\mathrm{poly}}$. By definition there exists a set A in P and an advice function f such that $x \in L$ iff $w \in L$ iff $< w, f(|w|) > \in A$. Since A is in P, there exists a polynomial time Turing machine M such that $A = L(M)$. By the same idea as in the proof of Theorem 3.3, there is a circuit that outputs 1 iff its input is in B.

The idea is then to consider as family of circuits the family of circuits that corresponds to each length, plugging the advice $f(n)$ in the circuit corresponding to length n. □

3.5 $\mathrm{P}_{\mathrm{poly}}$ and "Analog/Advice" Automata

3.5.1 "Analog/Advice" Automata

It may help to consider the following (rather artificial) model (following [Bournez and Cosnard, 1996]).

Definition 3.4 *An analog automaton (or advice automaton) (with k stacks) is exactly like a pushdown automata with k stacks, except that it has the possibility in addition of making appear in time 1 an infinite word $W \in \Sigma^\omega$ (that we can call* advice*) over some stack: in time 1, the content of the stack is replaced by W.*

As the program of an analog automaton is finite, there is only a finite number of possible advices that a given machine can make appear.

If you prefer considering that pushdown automata are actually Turing machines, if you prefer talking only about Turing machines, this corresponds to consider Turing machines that can replace the content of the tape at the right (or left) of the head by some infinite words W in time 1.

More formal definitions can be found in [Bournez and Cosnard, 1996], but basically they are only formalising the previous ideas. The question is then to understand what can be computed by analog automata.

3.5.2 In exponential time

Theorem 3.5 *Every language $L \subset \{0,1\}^*$ can be recognised by a (deterministic) analog two stack automaton in exponential time.*

Proof: Let $L \subset \{0,1\}^+$ be a language. Let the word γ, be the concatenation, with delimiters, by increasing word length order, of all the words of L. Let M be an analog automaton that, on input $w \in \{0,1\}^+$ on its first stack, makes advice γ appear on its second stack. Then M seeks in γ if w is present. If it is, M accepts. M stops processing as soon as it encounters a word of length greater than the length of w. L is recognised by M in exponential time. □

3.5.3 In polynomial time

Theorem 3.6 *The languages $L \subset \{0,1\}^*$ accepted by analog (deterministic) two stack automata in polynomial time are exactly the languages belonging to the complexity class $P/poly$.*

Proof: Let k be the number of different advices that the analog automaton M can possibly use. In polynomial time $p(n)$, M can at most read the $p(n)$ first letters of the k advices. So it is possible to simulate M with a Turing machine M', which gets as advice of polynomial size $kp(n)$ the $p(n)$ first letters of each of the k advices of M, and then simulates M. Hence the computational power of analog automata in polynomial time is bounded by $P/poly$.

Let L be a language in $P/poly$. By definition, L is recognised by a Turing machine M' with an advice function $f : \mathbb{N} \to \{0,1\}^*$. We can construct a word γ of infinite length as the concatenation, with delimiters, of $f(1), f(2), etc....$ In order to recognise L, an analog automaton M, on input $w \in \{0,1\}^+$, first makes advice γ appear. Then M seeks in γ the value of $f(|w|)$. This operation can be done in polynomial time, since there exists a polynomial p, such that, for all $i \in \mathbb{N}$, the size of $f(i)$ is bounded by $p(i)$: so M has at most to read $p(1) + p(2) + \cdots + p(|w|)$ characters, that is at most a polynomial number of characters. Finally, M simulates Turing machine M' on $(w, f(|w|))$. Hence L is recognised by M in polynomial time. □

Chapter 4

Circuits over a structure

4.1 Computing with circuits

In this section we introduce the notion of circuit over $\mathcal{K}$, and recall some links of this computational device with the BSS model of computation.

Remark 4.1 *A structure $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i\in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ has, by definitions, operations (i.e., functions) and relations. For practical reasons, and mainly to simplify the presentation, we will consider from now on relations also as particular functions, i.e., as functions that take values in $\{\mathbf{0}, \mathbf{1}\}$: $\mathbf{0}$ corresponds to $false$, $\mathbf{1}$ to $true$.*

4.1.1 Circuits

Definition 4.1 *A circuit over the structure $\mathcal{K}$ is an acyclic directed graph whose nodes, called* gates*, are labeled either as* input *gates of in-degree* 0*,* output *gates of out-degree* 0*,* 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, all gates behaving as one would expect. We just note that any selection gate tests whether its first parent is labeled with $\mathbf{1}$, and returns the label of its second parent if equality with $\mathbf{1}$ holds, or the label of its third parent if not. This evaluation defines a function from $\mathbb{K}^n$ to $\mathbb{K}^m$ where n is the number of input gates and m that of output gates. See [Poizat, 1995b, Blum et al., 1998] for formal details.

We say that a family $\{\mathcal{C}_n \mid n \in \mathbb{N}\}$ of circuits computes a function $f : \mathbb{K}^* \to \mathbb{K}^*$ when the function computed by the nth circuit of the family is the restriction of f to $\mathbb{K}^n$. We say that this family is *P-uniform* when there exist constants $\alpha_1, \ldots, \alpha_m \in \mathbb{K}$ and a deterministic Turing machine M satisfying the following. For every $n \in \mathbb{N}$, the constant gates of $\mathcal{C}_n$ 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).

Remark 4.2 *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.*

We are now going to use the notion of circuits as a computational model and compare this model to algorithms over some structure $\mathfrak{M}$.

We assume that some encoding of circuits is fixed: Recall that a circuit corresponds to a labeled graph, and hence, this basically only require to fix some encoding of label graphs.

In [Poizat, 1995b] the following result is proved.

Theorem 4.1 ([Poizat, 1995b]) *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 $t(n) \geq n$, and that the length of an output depends only on the size of its input.*

Then, there exists a family $\{\mathscr{C}_n \mid n \in \mathbb{N}\}$ of circuits 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 $\overline{x} = x_1 \ldots x_n$, $\mathscr{C}_n(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m)$ equals the output of f_M on input $\overline{x}$.

Moreover, there exists a deterministic Turing machine computing a description of the ith gate of the nth circuit in time polynomial in $t(n)$.

Remark 4.3 *The requirements of a homogeneous computation time bound and output size are not too strong: clocking an arbitrary BSS machine, and adding extra idle characters to its output allows one to build a BSS machine which complies with these requirements.*

The proof consists in unfolding the computation of the machine on inputs of size n, and is basically a generalisation of Theorem 3.3.

4.2 NP-completeness

The problem of *circuit satisfiability with parameters* is the following: given a circuit $C(\overline{x}, \overline{y})$ over structure $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i \in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$, and a word $\overline{c} = (c_1, \ldots, c_k) \in \mathbb{K}^*$ of same length than $\overline{x}$, decide if it possible to set values in M to variables $\overline{y} = (y_1, \cdots, y_n)$ such that $C(\overline{c}, \overline{y}) = 1$.

The problem of *circuit satisfiability with parameters with digital inputs* is the following: given a circuit $C(\overline{x}, \overline{y})$ over structure $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i \in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$, and a word $\overline{c} = (c_1, \ldots, c_k) \in \mathbb{K}^*$ of same length than $\overline{x}$, decide if it possible to set values in $\{\mathbf{0}, \mathbf{1}\}$ to variables $\overline{y} = (y_1, \cdots, y_n)$ such that $C(\overline{c}, \overline{y}) = 1$.

Theorem 4.2 *Let $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i\in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ be a structure. Circuit satisfiability with parameters over $\mathcal{K}$ is* NP*-complete over structure $\mathcal{K}$.*

Theorem 4.3 *Let $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i\in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ be a structure. Circuit satisfiability with parameters with digital inputs over $\mathcal{K}$ is* NDP*-complete over structure $\mathcal{K}$.*

Proof:[Of Theorem 4.2] The problem is in NP, since a circuit C with parameters $\overline{c}$ is satisfiable if and only if there exists a word $\overline{y}$ of length n, where n is the number of inputs of the circuit such that $\langle C, \overline{c}, \overline{y}\rangle \in CVP$, where CVP designs Circuit Value Problem (determine the value of a given circuit on some given input). CVP is easily shown to be in $\mathrm{P}_{\mathcal{K}}$.

Now, let L be a language of $\mathrm{NP}_{\mathcal{K}}$. By definition, there exists a problem A in $\mathrm{P}_{\mathcal{K}}$ and a polynomial p such that for all word $\overline{x}$, $\overline{x} \in L$ iff there exists some $\overline{y} \in \mathbb{K}^*$, length$(\overline{y}) \leq p($length$(\overline{x}))$ with $\langle \overline{x}, \overline{y}\rangle \in A$. According to equivalence of polynomial time with circuits of polynomial size, determining if $\langle \overline{x}, \overline{y}\rangle \in A$ corresponds to a circuit C (possibly with parameters $\overline{c}'$) of size polynomial in length $\langle \overline{x}, \overline{y}\rangle$, and hence in length$(\overline{x})$.

The function that maps $\overline{x}$ to $C(\overline{x}, \overline{c}', \overline{y})$, where $\overline{x}, \overline{c}'$ denote the parameters of circuit C, realises a reduction from L to the problem of circuit satisfiability. Indeed, $\overline{x} \in L$ if and only if $C(\overline{x}, \overline{c}', \overline{y})$ is satisfiable for some $\overline{y}$. □

The proof of the second theorem is similar, restricting existential quantification to $\{\mathbf{0}, \mathbf{1}\}^*$.

4.2.1 Rudimentary formulas

We can rewrite this result in the following way:

Definition 4.2 (Rudimentary formulas) *Let $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i\in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ be a structure. We call* rudimentary formula over $\mathcal{K}$ *a finite conjunction of formulas of the following types:*

1. *$x = a$, where a is an element of $\mathbb{K}$.*
2. *$y = f_i(\overline{x})$, where f_i is a function of the structure.*
3. *$r(\overline{x}) \vee y = \mathbf{0}$, where r_j is a relation of the structure (including the equality $=$)*
4. *$\neg r(\overline{x}) \vee y = \mathbf{1}$, where r_j is a relation of the structure (including the equality $=$).*
5. *$x = \mathbf{0} \vee x = \mathbf{1}$.*

6. $x = \epsilon \vee y = \epsilon'$, *where* $\epsilon, \epsilon' \in \{\mathbf{0}, \mathbf{1}\}$.

7. $x = \epsilon \vee y = \epsilon' \vee z = \epsilon''$, *where* $\epsilon, \epsilon', \epsilon'' \in \{\mathbf{0}, \mathbf{1}\}$.

A rudimentary formula is said to be *satisfiable* if it is possible to assign its variables in such a way that all its formulas becomes true.

Theorem 4.4 *Let* $\mathcal{K} = \left(\mathbb{K}, \{op_i\}_{i \in I}, r_1 \ldots, r_\ell, \mathbf{0}, \mathbf{1}\right)$ *a structure. The problem to determine if a given rudimentary formula is satisfiable is* $\mathrm{NP}_{\mathcal{K}}$*-complet.*

Proof: This is clearly a problem of $\mathrm{NP}_{\mathcal{K}}$.

To prove that it $\mathrm{NP}_{\mathcal{K}}$-complete, one uses a reduction from circuit satisfiability: One proves that a circuit is satisfiable if and only if a certain rudimentary formula of polynomial size is satisfiable.

The trick is to introduce one variable for each gate of the circuit to represent the value that it outputs, and to write that each such variable is expressible from the inputs of the gate, using the operator realised by the gate. Observe that every time this can be done using a conjunction of a formulas of the above types. One then write the conjunction of all the constraints and of the constraints that the output must be $\mathbf{1}$. The obtained rudimentary formula is of polynomial size, and is satisfiable if and only if the circuit is. □

Remark 4.4 *There is no reason that the satisfaction of rudimentary formula by boolean entries is* $\mathrm{NDP}_{\mathcal{K}}$*-complete since we have introduced variables such as* $y = f_i(\overline{x})$ *which are not booleans, but elements of* $\mathbb{K}$*, the base set of the structure.*

Corollary 4.1 *The problem of satisfiability of a formula of* $\leq$ 3-SAT *is* NP*-complete.*

Proof: Since $\leq$ 3-SAT is in NP, it suffices to prove that any rudimentary formula can always be written as a formula $\leq$ 3-SAT of polynomial size. A classical algorithms corresponds to the structure $\mathfrak{B} = (\{0,1\}, \mathbf{0}, \mathbf{1}, =)$. A component of the rudimentary formula of the form $x = y \vee u = \mathbf{0}$ can be replaced by $(x = \mathbf{0} \vee y = \mathbf{1} \vee u = \mathbf{0}) \wedge (x = \mathbf{1} \vee y = \mathbf{0} \vee u = \mathbf{0})$. A composant of the rudimentary formula of the form $\neg x = y \vee u = \mathbf{1}$ can be replaced by $(x = \mathbf{0} \vee y = \mathbf{0} \vee u = \mathbf{1}) \wedge (x = \mathbf{1} \vee y = \mathbf{1} \vee u = \mathbf{1})$. □

The problem k-SAT is the problem of satisfiability of a formula of a propositional formula in conjonctive normal form with exactly k-literals per clause.

Theorem 4.5 *The problem of the satisfiability of a* 3-SAT *formula is* NP*-complete.*

Proof: Each clause of less than 3 literals can be replaced by a conjunction of clauses with three literals. For example, $\alpha \vee \beta$ can be replaced by $(\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$. The first is satisfiable if and only if the second is. □

Corollary 4.2 *The problem of the satisfiability of a propositional calculus in conjonctive normal form (*SAT *formula) is* NP*-complete.*

Proof: Clearly 3-SAT ≤ SAT via the identity function. It suffices to observe that SAT ∈ NP. □

4.2.2 P = NP? and quantifier elimination

The problem of satisfiability of (free) formulas with parameters, consists in, given some free formula $\varphi(\overline{x},\overline{y})$ and some tuple $\overline{a}$ of elements of $\mathbb{K}$, to determine if there is some tuple $\overline{b}$ such that $\varphi(\overline{a},\overline{b})$ is satisfied.

We can easily deduce from the previous discussion:

Theorem 4.6 *In any structure $\mathcal{K}$, the problem of the satisfiability of (free) formulas with parameters is* $\mathrm{NP}_{\mathcal{K}}$*-complete.*

Proof: The rudimentary formulas are particular (free) formulas, and hence the problem of the satisfiability of rudimentary formulas reduces to this problem. Now the problem is clearly in $\mathrm{NP}_{\mathcal{K}}$. □

We obtain the following results that relates formulas and circuits, via the question of efficient elimination of quantifiers.

Theorem 4.7 *A structure $\mathcal{K}$ satisfies $\mathrm{P}_{\mathcal{K}} = \mathrm{NP}_{\mathcal{K}}$ if and only if there exists some algorithm in $\mathrm{P}_{\mathcal{K}}$ with parameters $\overline{c}$ which transforms any existential formula (without parameters) $\exists\overline{y}\ \varphi(\overline{x},\overline{y})$ into an equivalent circuit $C(\overline{c},\overline{x})$.*

Proof: If this latter property is satisfied, to test if $\varphi(\overline{a},\overline{y})$ is satisfiable, one starts by transforming the formula in it equivalent circuit, and one determines whether $C(\overline{c},\overline{a})$ values **1** or **0**.

Conversely, let A be some polynomial algorithm, using the parameters $\overline{c}$, that, when given some formula $\varphi(\overline{x},\overline{y})$, and $\overline{a}$, determines if $\varphi(\overline{a},\overline{y})$ is satisfiable. When the length of the formula and of the tuple $\overline{a}$ are fixed, this can be translated into a circuit $C(\overline{c},\varphi,\overline{x})$ such that $C(\overline{c},\varphi,\overline{a})$ values **1** if $\varphi(\overline{a},\overline{y})$ is satisfiable, **0** otherwise (here we abusively confuse φ with its encoding). The circuit $C(\overline{c},\varphi,\overline{x})$ is then equivalent to the formula $\exists\overline{y}\ \varphi(\overline{x},\overline{y})$. □

4.2.3 NP-complete problems over the reals

Theorem 4.8 ($4-FEAS$ is NP-complete) *In $(\mathbb{R},+,-,\times,=,<)$, the problem of the existence of a (real) root to a polynomial in n variables with real coefficients of total degree 4 is $\mathrm{NP}_{(\mathbb{R},+,-,\times,=,<)}$-complete.*

Proof: We prove first that any component of a rudimentary formula is equivalent to a formula of the type $\exists u\ Q(\overline{x},\overline{u})$ where Q is a polynomial.

This is clear for the cases 1. and 2. Now, $x = y \vee u = 0$ can be replaced by $u \times (y - x) = 0$. $x \neq y \vee u = 1$ can be replaced by $\exists v\ (u-1) \times ((x-y) \times v - 1) = 0$.

$x < y \vee u = 0$ can be replaced by

$$\exists v \exists w\ u \times ((y - x - v^2)^2 + (v \times w - 1)^2) = 0$$

$y \leq x \vee u = 1$ can be replaced by

$$\exists v\ (u-1) \times (x - y - v^2) = 0$$

Hence, the rudimentary formula can be replaced, modulo some new variables that are existentially quantified, by a conjunction of polynomial equations of total degree less than 6.

This can be decomposed in turn, by adding some new variables in equations of type $x - a = 0$, $x - y - z = 0$, $x - y \times z = 0$, which are all of degree 2. In the field of the reals, expressing that all these polynomials are null is equivalent to state that the sum of their squares is null. This sum is of total degree 4.

□

Other statements

Theorem 4.9 *The structure* $(\mathbb{R}, +, -, =)$ *satisfies* NP = NDP *et* P ≠ NP*, and has no* NP*-complete problem. Same think for the non-uniform case.*

Theorem 4.10 *The structure* $(\mathbb{R}, +, -, =, <)$ *satisfies* NP = NDP.

4.3 Bibliographic notes

The results of this chapter are from [Poizat, 1995b]. The relations between circuits and algorithms is a classical result developed in most of the complexity manuals. The application to the real knapsack problem is taken from [Koiran, 1994].

Bibliography

[Balcázar et al., 1988] Balcázar, J. L., Diáz, J., and Gabarró, J. (1988). *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer.

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

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

[Bournez and Cosnard, 1996] Bournez, O. and Cosnard, M. (1996). On the computational power of dynamical systems and hybrid systems. *Theoretical Computer Science*, 168(2):417–459.

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

[Koiran, 1994] Koiran, P. (1994). Computing over the reals with addition and order. *Theoretical Computer Science*, 133(1):35–47.

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

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

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

[Poizat, 1995b] Poizat, B. (1995b). *Les petits cailloux: Une approche modèle-théorique de l'Algorithmie*. Aléas Editeur.