Course Notes for MPRI Course 2.33.1: Blum Shub Smale Model

Olivier Bournez

Version of February 25, 2013.

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

Chapter 2

The Blum-Shub-Smale Model

2.1 The Model

Essentially, a BSS machine can be considered as a Random Access Machine (RAM) over $\mathbb{R}$, which is able to perform the basic arithmetic operations at unit cost and which registers can hold arbitrary real numbers.

Definition 1 *Let $Y \subset \mathbb{R}^\infty := \bigcup_{k\in\mathbb{N}} \mathbb{R}^k$. A* BSS-machine *over $\mathbb{R}$ with admissible input set Y (we will call in what follow such a machine a BSS-machine over Y) is given by a finite set I of instructions labelled by $0, \ldots, N$. A configuration of M is a quadruple $(n, i, j, x) \in I \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}^\infty$. Here n denotes the currently executed instruction, i and j are used as adresses (copy-registers) and x is the actual content of the registers of M. The initial configuration of M's computation on input $y \in Y$ is $(1, 1, 1, y)$. if $n = N$ and the actual configuration is (N, i, j, x) the computation stops with output x.*

The instructions of M can perform are of the following types:

computation $n : x_s \leftarrow x_k \circ_n x_l$, *where* $\circ_n \in \{+, -, *, /\}$. *or* $n : x_s \leftarrow \alpha$ *for some constant* $\alpha \in \mathbb{R}$.

The register x_s will tet the value $x_k \circ_n x_l$ or α respectively. All other register-entries remain unchanged. The next instruction will be $n + 1$ and the copy-registers are changed selectively according to $i \leftarrow i + 1$ or $i \leftarrow 1$ and similarly for j.

branch $n : if\ x_0 \geq 0$ *goto* $\beta(n)$ *else goto* $n + 1$*: according to the answer of the test, the next instruction is determined: here $\beta(n) \in I$. All other registers are unchanged.*

copy $n : x_i \leftarrow x_j$, *i.e. the content of the "read"-register is copied into the "write" register. The next instruction is $n + 1$. All other registers remain unchanged.*

All α appearing among the computation-instructions built up the (finite) set of *machine constants* of M.

Remark 1 *Such a machine is called a machine over structure* $(\mathbb{R}, +, -, *, /, \geq)$*: this means that the registers are assumed to take values in* $\mathbb{R}$*, and allowed operations are* $+, -, *, /$ *and allowed tests correspond to* $\geq$*. We will also call such machines BSS machines over* $\mathbb{R}$*. Of course, some restricted or more general operations/tests can also be considered. For example, a machine over structure* $(\mathbb{R}, +, -, \geq)$ *correspond to machines where programs do not use any multiplications or divisions. One can also consider machines working over other fields or rings: for example machines over* $\mathbb{C}$*, the field of complex numbers, instead of reals.*

Remark 2 *The kind of operations allowed depends on the underlying structure. A branch* $x \geq 0$*? for example does only make sense in an ordered set, see below.*

The copy-registers and copy-instructions are necessary in order to deal with arbitrary long inputs from $\mathbb{R}^\infty$. The way of changing the entries in the copy-registers (the "addressing") seems to be rather restrictive apart from the fact that there is no-indirect addressing. However actually it is general enough.

Now to any BSS-machine M over Y there corresponds in a natural way the function Φ_M computed by M. It is a partial function from Y to $\mathbb{R}^\infty$ and is given as the result of M's computation on an input $y \in Y$.

The following notions are then crucial for recursion and complexity theory:

Definition 2 *Let* $A \subset B \subset \mathbb{R}^\infty$ *and* M *be a BSS-machine over* B*.*

- *The* output-set *of* M *is the set* $\Phi_M(B)$*. The* halting-set *of* M *is the set of all inputs* y *for which* $\Phi_M(y)$ *is defined.*
- A *is recursively enumerable over* B *iff* A *is the outputs-set of a BSS-machine over* B*. (if* $B = \mathbb{R}^\infty$*, then* A *is simply called* recursively-enumerable*).*
- *A pair* (B, A) *is called a* decision-problem*. It is said* decidable *iff there exists a BSS-machine* M' *with admissible input* B *such that* $\Phi_{M'}$ *is the characteristic function of* A *in* B*. In that case,* M' *decides* (B, A)*.*

A can be seen easily (following classical arguments for classical computability for Turing machines), (B, A) is decidable iff A and $B \setminus A$ are both halting sets over B.

Previous notions are about computability theory.

2.2 Path Decomposition Theorem

Definition 3 *Let* M *be a BSS-machine over* $Y \subset \mathbb{R}^\infty$*,* $y \in Y$*. The* running time *of* M *is defined by*

$$T_M(y) := \begin{cases} \textit{number of operations executed by } M & \textit{if } \Phi_M(y) \textit{ is defined} \\ \infty & \textit{otherwise} \end{cases}$$

Definition 4 *A basic semi-algebraic set $S \subset \mathbb{R}^p$ is a set defined by a conjunction of polynomial equalities and inequalities:*

$$S = \{x \in \mathbb{R}^p; Q_1 > 0 \wedge Q_2 > 0 \wedge \dots Q_m > 0 \wedge R_1 = 0 \wedge \dots \wedge R_n = 0\}$$

where $Q_i, R_i \in \mathbb{Z}[X_1, \dots, X_p]$ are non-constant polynomials (m or n can be equal to 0).

A semi-algebraic set is a finite union of basic semi-algebraic set.

Theorem 1 *Any halting set A of a BSS machine over $\mathbb{R}$ is the countable union of basic semialgebraic sets.*

Proof: A halting set is the union of the Ω_T, for $T = 1, 2, \dots$, where Ω_T is the set of inputs accepted in a time $\leq T$. Now, when fixing T, Ω_T is the union over the possible paths γ of lengths T in the computation tree of the machine. Fixing a path γ, the set of inputs accepted following path γ is easily seen to be a basic semialgebraic set. □

Corollary 1 *A halting set of a machine over $\mathbb{R}$ has at most countably many connected components*

Proof: It is known that a semialgebraic set has a finite number of connected components. □

Recall that the Mandelbrot set is defined as the set of complex numbers c such that the sequence c, $c^2 + c$, $(c^2 + c)^2 + c$ ... remains bounded. It is co-recursively enumerable, as if x escapes the disk of radius 2, it will go off to infinity: hence, the following algorithm works to recognize the complement of the Mandelbrot set.

1. input c
2. $x \leftarrow x^2 + c$
3. if $|x| > 2$?, halt and outputs 1
4. go to line 2.

Corollary 2 *The Mandelbrot set is co-R.E. over $\mathbb{R}$ but is not decidable.*

Proof: It is known that the Mandelbrot set is not the countable union of a semialgebraic sets over $\mathbb{R}$. □

2.3 Complexity theory

In order to talk about complexity, we have to define a cost-measure for BSS-algorithms as a size-measure for problem instances.

Definition 5

- *For $x \in \mathbb{R}^\infty$ and such that $x = (x_1, x_2, \dots, x_k, 0, 0, \dots)$ it is*

$$size(x) := k.$$

The above definitions provides a first importance difference with classical complexity theory. First item states that any real number- independantly of its magnitude- is considered as entity. Similarly, second item defines the cost of any basic operation to be 1 - no matter about the operands.

We can now defined the most important complexity classes $P_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$ for real decision problems.

Definition 6

- *A decision problem* (B, A), $A \subset B \subset \mathbb{R}^\infty$ *belongs to class* $\mathrm{P}_{\mathbb{R}}$ *(deterministic polynomial time) iff there exists a BSS-machine* M *with admissible input-set* B *and constants* $k \in \mathbb{N}$, $c \in \mathbb{R}$ *such that* M *decides* (B, A) *and*
$$\forall y \in B, \quad T_M(y) \leq c * size(y)^k.$$
- (B, A) *belongs to class* $\mathrm{NP}_{\mathbb{R}}$ *(non-deterministic polynomial time) iff there exists a BSS-machine* M *with admissible input* $B \times \mathbb{R}^\infty$ *and constants* $k \in \mathbb{N}$, $c \in \mathbb{R}$ *such that the following conditions hold:*
  - $\Phi_M(y, z) \in \{0, 1\}$.
  - $\Phi_M(y, z) = 1 \Rightarrow y \in A$
  - $\forall y \in A$, $\exists z \in \mathbb{R}^\infty$, $\Phi_M(y, z) = 1$ *and* $T_M(y, z) \leq c * size(y)^k$.
- (B, A) *belongs to class* $\mathrm{coNP}_{\mathbb{R}}$ *iff* $(B, B \setminus A) in \mathrm{NP}_{\mathbb{R}}$.

The class $\mathrm{P}_{\mathbb{R}}$ can be considered as a theoretical formalization of problems being efficiently solvable; the running times increases only polynomially with the problem size.

Obviously $\mathrm{P}_{\mathbb{R}} \subset \mathrm{NP}_{\mathbb{R}}$. The question $\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$? can be considered as the main unsolved problem in real complexity theory and is the analogue to the classical P versus NP question in classical Turing theory.

Example 1 *For* $k \in \mathbb{N}$, *consider the sets:*

$$F^k := \{f \text{ polynomial in } n \text{ unknowns with real coefficients, } deg(f) \leq k, k \in \mathbb{N}\}$$

$$F^k_{zero} := \{f \in F^k | f \text{ has a real zero}\}$$

and

$$F^k_{zero,+} := \{f \in F^k | f \text{ has a real zero with all components nonnegative}\}$$

whee a polynomial is represented as an element of $\mathbb{R}^\infty$ *by writing down the coefficients of all its monomials in a given order. Then* (F^k, F^k_{zero}) *and* $(F^k, F^k_{zero,+})$ *belongs to the class* $\mathrm{NP}_{\mathbb{R}}$ *for all* $k \in \mathbb{N}$ *by guessing a (non-negative) zero* x, *plugging it into* f *and evaluating* $f(x)$. *If* $k \geq 4$, *the according problem lead into the heart of complexity theory as we will see in the next subsection.*

In order to compare problems with respect to the difficulty of solving them, the notion of polynomial time reducibility is central. The underlying idea is that a problem (B_1, A_1) is at least as hard to solve as problem (B_2, A_2) if each instance of the latter can be reduced fast (in polynomial time) into one of the first.

Definition 7 *Let (B_1, A_1), (B_2, A_2) be decision problems.*

- *(B_2, A_2) is reducible in polynomial time to (B_1, A_1) if there exists a BSS-machine M over B_2 such that $\Phi_M(B_2) \subset B_1$, $\Phi_M(B_2) \subset B_1$, $f\Phi_M(y) \in A_1 \Leftrightarrow y \in A_2$ and M works in polynomial time. Notation $(B_2, A_2) \leq_{\mathbb{R}} (B_1, A_1)$.*
- *$(B_1, A_1) \in \mathrm{NP}_{\mathbb{R}}$ is NP_R complete if any problem in $\mathrm{NP}_{\mathbb{R}}$ is reducible to in in polynomial time.*

Complete problems are essential for complexity classes because they represent the most hardest problems in int. As in the classical theory, the relation $\leq_{\mathbb{R}}$ is both transitive and reflexive. This implies that $\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$ iff there exists a $\mathrm{NP}_{\mathbb{R}}$ complete problem in class $P_{\mathbb{R}}$. That is the reason why studying complete problems bears such importance.

Observe that even the fact that $\mathrm{NP}_{\mathbb{R}}$ problems are decidable is not trivial and not so direct: one can not test all possible guess as the guesses live in a continuum.

However, there are indeed decidable and even in single exponential time. Furthermore, NP-complete problems exist:

Theorem 2
- *For any $k \geq 4$, the problem (F^k, F^k_{zero}) is $\mathrm{NP}_{\mathbb{R}}$-complete.*
- *All problems in class $\mathrm{NP}_{\mathbb{R}}$ are decidable in single exponential time.*

2.4 Space complexity?

Beside the time measure for computations, also the used space resources can be considered. However, a result by Christian Michaux shows the irrelevancy of this notion in a broad context, and hence the difficulty in defining easily a counterpart for classical class PSPACE.

Theorem 3 *In the BSS-model over the structure $(\mathbb{R}, +, -, x/2, =, <)$, every polynomial time decision problem can be solved by an algorithm working in polynomial time needing constant additional space.*

Chapter 3

Relations with discrete complexity theory

3.1 Boolean part of a real class

Let $\mathcal{C}$ be some complexity class over the reals. The *boolean part* $BP(\mathcal{C})$ of $\mathcal{C}$ denotes

$$BP(\mathcal{C}) := \{L \cap \{0,1\}^*; L \in \mathcal{C}\}.$$

If $\mathcal{C}$ is a non-deterministic complexity class, then $D\mathcal{C}$ (digital $\mathcal{C}$) denotes the subclass of those problems in $\mathcal{C}$ which membership can be established by guessing only elements from $\{0,1\}^*$ (for example $D\mathrm{NP}_{\mathbb{R}}$)

The aim is now to compare classical (boolean) complexity classes to the boolean parts of real ones.

3.2 Weak Blum-Shub-Smale model

Consider the following program

- input $x \in \mathbb{R}$
- for $i := 1$ to n

  $x \leftarrow x^2$
- output x

Here within n steps, the value x^{2^n} is computed, i.e. a polynomial of degree 2^n can be computed in n steps. To avoid such an effect (which is very untypical for the Turing model), Koiran introduced a different cost measure for the allowed operations. The key idea is to consider the real constants of a machine as separated inputs in order to speak about the bit size of numbers appearing during a computation.

Let M be a BSS-machine with real constants $\alpha_1, \ldots, \alpha_s$. For any input size n, M realizes a algebraic computation tree: if any node/register is passed by M during this computation, the value computed by M up to this node is of the form $f_v(\alpha_1, \ldots, \alpha_s, x_1, \ldots, x_n)$ where $f_v \in \mathbb{Q}(\alpha_1, \ldots, \alpha_s, x_1, \ldots, x_n)$ is a rational function with rational coefficients only. Now the *weak cost* of the according operation is fixed as maximum of $\deg(f_v)$ (where deg is for degree) and the maximum height of all coefficients of f_v (here the height of rational p/q is given by $\lceil \log(|p|+1) + \log(|q|) \rceil$).

Definition 8 *The* weak BSS-model *is given as the BSS-model with the weak cost-measure, i.e. the weak running time of a BSS-machine M on input $x \in \mathbb{R}^\infty$ is the sum of the weak costs related to all operations M performs until $\Phi_M(x)$ is computed. Weak deterministic and non-deterministic polynomial time as as weak polynomial time reducibility are defined in a straightforward manner, and denoted by an index w. For example,* P_w, NP_w.

3.3 Linear model

A linear machine is a machine over structure $(\mathbb{R}, ., +, -, \geq)$, where . denotes multiplication by a constant.

If one prefers, this exactly the same as previous definitions for BSS-machine except that computation instructions are of the form

$n : x_s \leftarrow x_k \circ_n x_l$, where $\circ_n \in \{+, -, \}$ or $n : x_s \leftarrow k_n * x_l$ for some $k_n \in \mathbb{Q}$ or $n : x_s \leftarrow \alpha$ for some constant $\alpha \in \mathbb{R}$.

We write by an index *lin* the corresponding complexity classes: for example P_{lin} denotes deterministic polynomial time for linear machines.

3.4 Relations with classical complexity classes

Theorem 4

$$BP(P_w) = \mathrm{P}/poly.$$

Corollary 3

$$BP(P_{lin}) = \mathrm{P}/poly.$$

The fact that the boolean part of these real classes contains $\mathrm{P}/poly$ can be seen as the fact that this is easy to simulate an analog pushdown automaton using the considered class of models: basically, an advice correspond to a particular real constant; such a constant can be put in some real registers in time 1: this corresponds to making the advice appear in an analog pushdown automaton. Simulating all other operations of (analog) pushdown automata can be done as we did with PAMs or PCD.

The less trivial statement is the reverse inclusion: that no more than $\mathrm{P}/poly$ can be computed. As $P_{lin} \subset P_w$, the corollary is indeed a corollary of the theorem.

The proof idea to prove the theorem is substitute real constants of a weak BSS-machine by rationals, of small bit-sizes, and to perform the same computations, but now with the rational constants instead of the original real constants.

In order to realize such a substitution keeping the guarantee the same results of the computation at least on inputs from $\{0,1\}^n$ for every $n \in \mathbb{N}$, the following result on semi-algebraic sets is established: Basically, the theorem states that there exists small rational points in semi-algebraic set defined by "small" polynomials over $\mathbb{Z}$.

Theorem 5 *Let $S \subset \mathbb{R}^s$ be a semi-algebraic set with defining polynomials*

$$P_i(\alpha_1, \ldots, \alpha_s) > 0,\ i = 1, \ldots, N\ P_i \in \mathbb{Z}[\alpha_1, \ldots, \alpha_s].$$

Let D be the maximum of $\deg(P_i)$ and H the maximum height of the coefficients in the P_i's. If S is non-empty, then there exists constants a, b depending only on s and an $\overline{\alpha} \in \mathbb{Q}^s \cap S$ such that the height of $\overline{\alpha}$ is bounded by $a.H.D^b$.

Note that the height of $\overline{\alpha}$ is independent of the number N of polynomials. Analyzing weak polynomial time computations for a fixed input dimension is a situation as in above theorem can be derived. Due to the fact that only inputs from $\{0,1\}^*$ have to be considered, the real constants $\alpha_1, \ldots, \alpha_s$ are exchanged with $\overline{\alpha}$. Since $\overline{\alpha}$ is small, the arising algorithm can be executed in polynomial time also by a Turing machine. However, for different input dimensions different rationals points $\overline{\alpha}$ can appear. This forces the resulting process to be non-uniform. The reason why P/*poly* comes into play.

3.5 Other similar results

Definition 9 (Additive machines) *If the set of operations in the BSS-models is reduced to additions/subtractions, we get* additive *BSS-machines. If only test-operations $x = 0$? can be performed (instead of more general $x \geq 0$?) we get BSS machines* branching on equality.

Notationally, we will indicate the the fact of branching on equality as upper index $=$, whereas the kind of model correspond to lower index: for example P_{add} an $\mathrm{P}^{=}_{add}$ correspond to deterministic polynomial time, and to deterministic polynomial time branching on equality.

Theorem 6

- $\mathrm{P}/poly = BP(\mathrm{P}_{add}) = BP(\mathrm{P}_{lin}) = BP(\mathrm{P}_w)$
- $\mathrm{NP}/poly = BP(D\mathrm{P}_{add}) = BP(D\mathrm{P}_{lin}) = BP(D\mathrm{P}_w)$
- $P = BP(P^{=}_{add}) = BP(P^{=}_{lin})$
- $NP = BP(NP^{=}_{add}) = BP(NP^{=}_{lin})$

It can also be proved that:

Theorem 7

- $D\mathrm{NP}_{lin} = \mathrm{NP}_{lin}$, $D\mathrm{NP}^{=}_{lin} = \mathrm{NP}^{=}_{lin}$
- $D\mathrm{NP}_{add} = \mathrm{NP}_{add}$, $D\mathrm{NP}^{=}_{add} = \mathrm{NP}^{=}_{add}$

The basic idea to switch from arbitrary to digital nondeterminism is guessing the coding of an accepting computation path instead of guessing real numbers which force the according machine to take that path. Then it has to be checked whether the guessed path can appear during a computation.

Chapter 4

Circuits and Complexity Theory

4.1 Circuits

Consider a machine working over a structure $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$: M is a set, $f_1, f_2, \ldots, f_u$ are functions with arities, and $r_1, r_2, \ldots, r_v$ relations with arities.

If you prefer, consider the case of machines over $\mathbb{R}$, where you can consider that $M = \mathbb{R}$,$u = 4$, $f_1 = +$, $f_2 = -$, $f_3 = *$, $f_4 = /$, and $v = 1$, $r_1 = \geq$.

We assume that M contains 0 and 1, with $0 \neq 1$. Relations can be considered as functions that values 0 or 1. Assume that a function $S(x, y, z)$ such that $S(1, y, z) = y$ and $S(0, y, z) = z$ can be built from functions $f_1, f_2, \ldots, f_u$: over $\mathbb{R}$, take $S(z, y, z) = x * y + (1 - x) * z$.

The notion of boolean circuit can be generalized to the notion of circuit over $\mathfrak{M}$:

Definition 10 (Circuit over a structure) *Let n be some integer. A* circuit with n inputs and m outputs over structure $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$ *is a directly oriented graph with*

- n inputs*, that is to say n vertices without ingoing arc.*
- *exactly m* outputs*, that is to say m vertices without outgoing arc.*
- *each input is labeled either by a constant (an element of M), or by a variable symbole $x_1, x_2, \cdots, x_n$.*
- *any other vertex is called a* gate *and is labeled by a function or a relation of the structure. The fanin of each vertex correspond to the arity of the function symbol or relation symbol corresponding to its label.*

We leave to the read the definition of the evaluation of a circuit on an element of M^n.

4.2 Circuits vs Complexity

A fundamental remark is the following:

Proposition 1 *Let A be an algorithm over structure $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$). Let n be an integer. Let t be an integer.*

There exists a circuit $C_{n,t}$ over structure $\mathfrak{M}$ that accepts exactly the words of length n accepted by A in less than t steps.

The proof consists in unfolding the flowchart of the machine up to time t.

A given circuit recognizes only words over $\{0,1\}$ of fixed lengths. If one want consider recognition of languages, one needs to talk about family of circuits.

Recall (see e.g. [1, 5]) that a family of boolean circuits $\mathcal{C} = (C_i)_{i\in\mathbb{N}}$, with C_i with i inputs and 1 output, recognizes a language $L \subset M^*$, 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 (of size equal to the number of inputs of C) can be done in polynomial time. Denote by CVP the decision problem that consists in evaluation C on input x.

Polynomial time can be characterized by circuits:

Theorem 8 *A language $L \subset M^\infty$ is recognized in polynomial time by a machine over $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$, iff*

1. *L is recognized 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 (by a machine over $\mathfrak{M}$)*

Proof: Direct sense is obtained as in previous proposition (similarly to the classical settings).

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. □

4.3 NP-completness

The problem of *circuit satisfability with parameters* is the following: given a circuit $C(\mathbf{x}, \overline{y})$ over structure $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$, and a word $\overline{c} = (c_1, \ldots, c_k) \in M^*$ of same length than $\mathbf{x}$, 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 satisfability with parameters with digital inputs* is the following: given a circuit $C(\mathbf{x}, \overline{y})$ over structure $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$, and a word $\overline{c} = (c_1, \ldots, c_k) \in M^*$ of same length than $\mathbf{x}$, decide if it possible to set values in $\{0,1\}$ to variables $\overline{y} = (y_1, \cdots, y_n)$ such that $C(\overline{c}, \overline{y}) = 1$.

Theorem 9 *Let $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$ be a structure. Circuit satisfability with parameters over $\mathfrak{M}$ is* NP*-complet over structure $\mathfrak{M}$.*

Theorem 10 *Let $\mathfrak{M} = (M, f_1, \cdots, f_u, r_1, \cdots, r_v)$ be a structure. Circuit satisfability with parameters with digital inputs over $\mathfrak{M}$ is* NDP*-complete over structure $\mathfrak{M}$.*

Proof: The problem is in NP, since a circuit C with parameters $\overline{c}$ is satisfiable iff there exists a word $\mathbf{x}$ of length n, where n is the number of inputs of the circuit such that $< C, \overline{c}\ \mathbf{x} > \in CVP$.

Now, let L be a language of NP. By definition, there exists a problem A in P and a polynomial p such that for all word w, $w \in L$ iff there exists some $u \in M^*$, $|u| \leq p(|w|)$ with $< w, u > \in A$. According to equivalence of polynomial time with circuits of polynomial size, determining if $< w, u > \in A$ corresponds to a circuit C (possibly with parameters $\overline{c}'$) of size polynomial in $| < w, u > |$, and hence in $|w|$.

The function that maps w to $C(w, \overline{c}', \mathbf{x})$, where $\mathbf{x}$ denotes the inputs, $w, \overline{c}'$ the parameters, realize a reduction from L to the problem of circuit satisfiability. Indeed, $w \in L$ iff $C(w, \overline{c}', \mathbf{x})$ is satisfiable for some u. □

Chapter 5

Some results

5.1 Separation results

Several separation results can be proved. We focus on a simple proof.

Theorem 11
- $\mathrm{P}^{=}_{lin} \neq \mathrm{NP}^{=}_{lin}$
- $\mathrm{P}^{=}_{add} \neq \mathrm{NP}^{=}_{add}$

Consider the following decision problem KP (Knapsack):

- Inputs: $x_1, \ldots, x_n$.
- Question: Do there exist $\epsilon \in \{0,1\}^n$ such that

$$\epsilon_1 x_1 + \cdots + \epsilon_n x_n = 1$$

The problem is clearly in NP and NDP.
It cannot be solved in polynomial time by a linear or additive machine.

Lemma 1 *Let T be a machine over structure $S = (\mathbb{R}, +, -, =)$ on input $x = (x_1, \ldots, x_n)$.*
At every time t:

- *registers have values of the form*

$$y(t) = \lambda_1(t)x_1 + \cdots + \lambda_n(t)x_n + \beta$$

- *all tests are of the form*

$$\lambda_1(t)x_1 + \cdots + \lambda_n(t)x_n + \beta = 0?$$

with $\lambda_1(t), \ldots, \lambda_n(t) \in \mathbb{Q}$.

Proof: By an easy induction. □

Suppose that problem KP is recognized by a machine in time $p(n)$.

All tests are of the form $l_t(x) = 0$ with L_t affine.

In a finite time $p(n)$, machine does at most a finite number m of tests.

There exists some input $x \notin KP$ such that the result of all tests $l_1(x) = 0?, \ldots, l_m(x) = 0?$ are all negative, with $m \leq p(n)$.

Consider $\epsilon \in \{0,1\}^n$ such that $\epsilon_1 x_1 + \cdots + \epsilon_n x_n = 1$ is not among the m tests.

Then the machine is wrong on an $y \in \mathbb{R}^n$ such that $\epsilon_1 y_1 + \cdots + \epsilon_n y_n = 1$ et $l_1(y) \neq 0, \ldots, l_m(y) \neq 0$, as the answer must be the same on y as on x, as results of tests are the same.

Such an y exists since the last m constraints remove only $n-2$-dimensional subspaces from the hyperplane satisfying the first constraint.

Chapter 6

Bibliography

This document has been written using (often copying) mainly [4], and book [2].
Last chapter is based on [3].

Bibliography

[1] José Luis Balcázar, Josep Diáz, and Joaquim Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

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

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

[4] K. Meer and C. Michaux. A survey on real structural complexity theory. *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 4(1), 1997.

[5] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.