Combinatorial Complexity Theory

May 7, 1998

Preface

These lecture notes were written when teaching the course Combinatorial Complexity Theory at Aarhus University in the fall 1993 and the spring 1995.

Acknowledgement. The notes are heavily based on earlier material written (in Danish) by Sven Skyum and Peter Bro Miltersen.

Request. I will appreciate any reader of these notes to report errors and other comments that you may have to `gsfrandsen@daimi.au.dk`.

Warning. The exercises vary a lot in difficulty. There are simple check exercises next to problems, whose original solution required a whole research paper to describe.

Aarhus, June 1995.

Gudmund S. Frandsen

Minor corrections 96/08/01 and 98/05/01. G.S.F.

Contents

1 Boolean Functions **1**

1.1 Disjunctive and Conjunctive Normal Form 1

1.2 Bases and Expressions 3

1.3 Monotone and symmetric functions 4

1.4 Representation of problems 5

2 Circuit Models and Complexity Measures **8**

2.1 Circuits and Straight Line Programs 8

2.2 Size and Depth 8

2.3 Upper Bounds 10

2.4 Depth and Formula Size 13

2.5 Depth and circuit size 14

2.6 The Complexity of Function Families 18

2.7 Projection 19

2.8 Bases with unlimited fan-in 21

2.9 Branching Programs 22

3 Constructions I **26**

3.1 Addition of two numbers 26

3.1.1 Method 1 26

3.1.2 Method 2 27

3.1.3 Method 3 (Carry-look-ahead) 28

3.2 Addition of many numbers 31

3.3 Symmetric functions 32

3.4 Comparator Networks 33
3.4.1 Merging Networks 34
3.4.2 Splitting Networks 37
3.4.3 Th_n^k for fixed k 38
3.5 Threshold Functions 40

4 Machine Models and Boolean Circuits 49

4.1 Turing Machines 49
4.2 Uniformity and nonuniformity 50
4.3 Time and Size 51
4.4 Space and Depth 54
4.5 Parallel Time and Depth 56
4.6 Probabilistic Machines and non-uniformity 56
4.7 Nondeterministic Circuits 61

5 Lower Bounds I 65

5.1 Lower bounds for almost all functions 65
5.2 Lower Bounds for Formula Size 68
5.2.1 Krapchenko's bound for the basis $\{+, \cdot, ^-\}$. 69
5.2.2 Nečiporuk's bound for the full binary basis B_2. 71
5.3 Lower bounds for circuit size 73
5.3.1 The substitution technique 74
5.4 Lower bounds for unbounded fan-in circuits 75

6 Constructions II 88

6.1 Arithmetic circuits 88

6.2 The power of subtraction 88
6.3 Elimination of division 90
6.4 Parallel computation of low degree functions 92
6.5 Degree bounded Boolean circuits 97

7 Lower Bounds II **105**

7.1 Communication Complexity 105
7.2 Lower bound on probabilistic communication complexity 108
7.3 Lower bound on monotone depth 112
7.4 Lower bound on Monotone Size 115
7.4.1 The k-clique problem 116

8 Overview **130**

8.1 Class Overview 130
8.2 General Literature 133

x	NEG
0	1
1	0

$x\ y$	AND
0 0	0
0 1	0
1 0	0
1 1	1

$x\ y$	OR
0 0	0
0 1	1
1 0	1
1 1	1

$x\ y$	IMP
0 0	1
0 1	1
1 0	0
1 1	1

Figure 1: Tables for selected functions.

1 Boolean Functions

Definition 1.1 *A Boolean function is a function* $f : \{0,1\}^n \to \{0,1\}^m$.

The set of Boolean functions from $\{0,1\}^n$ *into* $\{0,1\}^m$ *is denoted* $\mathrm{B}_{n,m}$.

We may regard $f \in \mathrm{B}_{n,m}$ as a vector $(f_1, f_2, \cdots, f_m)$ of m functions f_i from $\mathrm{B}_{n,1}$. For simplicity we will write B_n for $\mathrm{B}_{n,1}$ and B for $\bigcup_{n \geq 0} \mathrm{B}_n$. A simple way to describe $f \in \mathrm{B}_n$ consists in tabulating f (see figure 1).

If 0 and 1 denotes *false* and *true* respectively, then the functions in figure 1 are the well known Boolean functions *negation*, *and*, *or* and *implication*. For functions with two variables we will use infix notation, when convenient.

Notation 1.1 *We write* $\overline{x}$ *for* $\mathrm{NEG}(x)$, $x \cdot y$ *(or* xy*) for* $\mathrm{AND}(x,y)$, $x + y$ *for* $\mathrm{OR}(x,y)$ *and* $x \Rightarrow y$ *for* $\mathrm{IMP}(x,y)$.

There are 2^{m2^n} distinct functions in $\mathrm{B}_{n,m}$, since each of the 2^n table rows can take any of 2^m distinct values.

In section 5 we will show the (perhaps surprising) result that for most functions a table is a compact description without much redundancy. However, tables are impractical, and a lot of functions arising in practice have considerably more compact representations. The size of representations arising from or related to methods of computation is a major part of this course. We continue by defining two special representations.

1.1 Disjunctive and Conjunctive Normal Form

These are restricted formulas over $\{+, \cdot, ^-\}$.

Example 1.1 $\mathrm{IMP}(x,y) = \overline{x}\,\overline{y} + \overline{x}\,y + x\,y = \overline{x} + y$

Example 1.1 demonstrates that a single function can be described by several distinct formulas (in principle infinitely many). As a consequence, it may pose a problem to decide if two formulas describe the same function. We introduce canonical forms to circumvent this type of problem.

Let x^y be a description for $x \cdot y + \overline{x} \cdot \overline{y}$. With this notation $x^1 = x$ and $x^0 = \overline{x}$.

We define the relation $\leq$ on B_n by:

$$f \leq g \text{ if and only if } \forall \underline{x} \in \{0,1\}^n : f(\underline{x}) \leq g(\underline{x})$$

Let $f \in \mathrm{B}_n$ and $(c_1, c_2, \cdots, c_n) \in \{0,1\}^n$. We say that $x_1^{c_1} x_2^{c_2} \cdots x_n^{c_n}$ is a *minterm* for f, if $x_1^{c_1} x_2^{c_2} \cdots x_n^{c_n} \leq f(\underline{x})$.

It holds (exercise 1.2) that

$$f(\underline{x}) = \sum_{\underline{c} \in \{0,1\}^n, f(\underline{c})=1} x_1^{c_1} x_2^{c_2} \cdots x_n^{c_n}. \tag{1}$$

This is called *disjunctive normal form* (DNF).

Similarly, f may be written as a product of sums denoted *conjunctive normal form* (CNF). It holds (exercise 1.2) that

$$f(\underline{x}) = \prod_{\underline{c} \in \{0,1\}^n, f(\underline{c})=0} (x_1^{\overline{c_1}} + x_2^{\overline{c_2}} + \cdots + x_n^{\overline{c_n}}) \tag{2}$$

The product is taken over precisely those sums $(x_1^{\overline{c_1}} + x_2^{\overline{c_2}} + \cdots + x_n^{\overline{c_n}})$ for which $f(\underline{x}) \leq (x_1^{\overline{c_1}} + x_2^{\overline{c_2}} + \cdots + x_n^{\overline{c_n}})$. The function IMP from example 1.1 is only 0 for $(1,0)$ (see figure 1). So the CNF for IMP is $x^{\overline{1}} + y^{\overline{0}} = x^0 + y^1 = \overline{x} + y$.

The above normal forms are unique, but they share one problem with tabulation. They may be unreasonably large. For this reason, we will relax the definition of normal forms: In the following, DNF means a "sum of products" and CNF means a "product of sums". The price of the relaxation is a loss of uniqueness.

x	y	0	$\cdot$		x		y	$\oplus$	$+$		$\equiv$	$\overline{y}$	$\Leftarrow$	$\overline{x}$	$\Rightarrow$		1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 2: B_2

1.2 Bases and Expressions

In the last section, we have used that all Boolean functions (B) can be expressed in terms of the simple operators (functions) $+, \cdot$ and $^-$. The existence of the normal forms CNF and DNF provides an implicit proof of this fact.

As mentioned earlier, there are 2^{2^n} functions in B_n. B_0 is the constant functions 0 and 1. B_1 is the functions, 0, 1, x and $\overline{x}$. B_2 consists of the 16 functions listed in figure 2.

Definition 1.2 *A* basis $\mathcal{B}$ *is a subset of functions from* B. *The* fan-in *of a finite basis* $\mathcal{B}$ *is the highest arity of any function in* $\mathcal{B}$. *The* span *of a basis* $\mathcal{B}$*,* $\mathrm{span}(\mathcal{B})$*, are those functions that can be defined by expressions over* $\mathcal{B}$*. A basis* $\mathcal{B}$ *is* complete *for a class of functions* C*, if* $C = \mathrm{span}(\mathcal{B})$*. If* $\mathcal{B}$ *is complete for* B*, we simple say that* $\mathcal{B}$ *is complete.*

Definition 1.3 *A* $\mathcal{B}$*-expression over* $\{x_1, \cdots, x_n\}$ *is defined inductively by:*

- *Constants:* $0, 1$ *are* $\mathcal{B}$*-expressions.*
- *Variables:* $x_1, \ldots, x_n$ *are* $\mathcal{B}$*-expressions.*
- *Composite expressions: If* $e_1, e_2, \cdots, e_k$ *are* $\mathcal{B}$*-expressions and* $f \in \mathcal{B} \cap \mathrm{B}_k$*, then* $f(e_1, e_2, \cdots, e_k)$ *is a* $\mathcal{B}$*-expression.*

We have already observed that $\{+, \cdot, ^-\}$ is a complete basis. For the rest of the course, this will be our default basis. $\{\oplus, \cdot\}$ is also a complete basis (exercise 1.4). $\{+, \cdot\}$ is not a complete basis, since the function $\overline{x}$ is not a $\{+, \cdot\}$-expression (why not?).

1.3 Monotone and symmetric functions

Definition 1.4 *A function* $f \in \mathrm{B}_n$ *is* monotone, *if* $\underline{x} \leq \underline{y}$ *(component-wise) implies that* $f(\underline{x}) \leq f(\underline{y})$. *The set of monotone functions (in n variables) is denoted* MON *(*MON_n*).*

It holds that f is monotone, if and only if it is possible to define f by a $\{+,\cdot\}$-expression (exercise 1.6). $\{+,\cdot\}$ will be our default basis, when dealing with monotone functions

Definition 1.5 *A function* $f \in \mathrm{B}_n$ *is* symmetric, *if for all permutations* $\pi \in S_n$ *and* $\underline{x} \in \{0,1\}^n$ *it holds that* $f(x_{\pi(1)}, x_{\pi(2)} \cdots, x_{\pi(n)}) = f(\underline{x})$. *The set of symmetric functions is denoted* SYM *(*SYM_n*).*

The value of $f \in \mathrm{SYM}_n$ is only dependent on the number of 1's in the argument of f (and independent of the actual position of the 1's). This fact implies a one to one relation between SYM_n and the set of subsets of $\{0,1,\cdots,n\}$. Therefore the following definition makes sense:

Definition 1.6 *For* $M \subseteq \{0,1,\cdots,n\}$, *let* Eq_n^M *denote the symmetric function given by:*

$$Eq_n^M(\underline{x}) = \begin{cases} 1 & \text{if } |\{i|x_i = 1\}| \in M \\ 0 & \text{otherwise} \end{cases}$$

If $f \in \mathrm{MON}_n \cap \mathrm{SYM}_n$, then f is given by an $M \subseteq \{0,1,\cdots,n\}$ of the form $\{k, k+1, \cdots, n\}$. We call f a threshold function:

Definition 1.7 *The threshold function* Th_n^k *is given by:*

$$Th_n^k(\underline{x}) = \begin{cases} 1 & \text{if } |\{i|x_i = 1\}| \geq k \\ 0 & \text{otherwise} \end{cases}$$

Example 1.2 $Th_3^2(x,y,z) = xy + xz + yz$

1.4 Representation of problems

So far we have mentioned ways to describe Boolean functions, and introduced a few properties Boolean functions may possess. We will now sketch how problems arising in various areas can be represented as Boolean functions.

Problems at the bit level such as adders will be studied in section 3.

We take graph problems as an example.

Let $G = (V, E)$ be a directed graph, where the nodes V are numbered $1, 2, \cdots, n$. G may be represented by an incidence matrix $\{c_{ij}\}$ where $c_{ij} = 1$, if and only if $(i, j) \in E$. This means that a directed graph can be described by n^2 Boolean values. The graph represented by $\{c_{ij}\}$ (or $\underline{c}$) is denoted $G[\underline{c}]$.

Any decision problem on graphs, may now be defined by Boolean functions. Consider a concrete example. Does there exist a path in G (with n nodes) from node i to node j? This problem is defined by the function f, where

$$f(\underline{x}) = \begin{cases} 1 & \text{if graph } G[\underline{x}] \text{ has a path from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

Since graphs come in all sizes, a graph problem corresponds to a *family of functions*:

Definition 1.8 *A* family $F = \{F_i\}$ *of* Boolean functions *is a sequence of functions indexed by some subset of the natural numbers.*

Example 1.3 *The path problem for directed graphs may be described by the following family:*

$$DPATH = \{DPATH_1, DPATH_4, \cdots, DPATH_{n^2}, \cdots\}$$

where

$$DPATH_{n^2}(\underline{x}) = \begin{cases} 1 & \textit{if graph } G[\underline{x}] \textit{ has a path from node } 1 \textit{ to node } n \\ 0 & \textit{otherwise} \end{cases}$$

Note that $DPATH_{n^2}$ is monotone. (If there is a path from u to v in G, then that path will remain, when more edges are added to the graph).

If all functions in a family belong to a certain class, then we will also say that the family belongs to this class. In the case of the path problem, we write $DPATH \in \mathrm{MON}$, since $DPATH_{n^2} \in \mathrm{MON}_{n^2}$ for all n.

All formal problems that possess a representation that makes them solvable on a computer, can also be represented as a family of Boolean functions. Assume P is a program, that outputs yes or no on any input. By viewing the input as a binary string, we have a canonical method for associating a family of functions $\{f_i^{(P)}\}$ with P:

$$f_i^{(P)}(\underline{x}) = \begin{cases} 1 & \text{if } P \text{ outputs yes on input } \underline{x} \\ 0 & \text{otherwise} \end{cases}$$

The precise binary encoding of the input is implicitly defined by P, and it is not our concern at the moment. We will discuss standard representations for various problem classes later as the need arises.

Exercises

Exercise 1.1 *Show that the following equalities are valid:*

1. $x + y = y + x$ *and* $xy = yx$
2. $(x + y) + z = x + (y + z)$ *and* $(xy)z = x(yz)$
3. $x(y + z) = xy + xz$ *and* $x + yz = (x + y)(x + z)$
4. $x \cdot \overline{x} = 0, x + \overline{x} = 1, x \cdot x = x$ *and* $x + x = x$
5. $x \cdot 1 = x, x \cdot 0 = 0, x + 1 = 1$ *and* $x + 0 = x$
6. $\overline{(x + y)} = \overline{x}\,\overline{y}$ *and* $\overline{(xy)} = \overline{x} + \overline{y}$ *(DeMorgan's law)*
7. $x(x + y) = x$ *and* $x + xy = x$
8. $\overline{(\overline{x})} = x$

A Boolean algebra is a structure $\{M, 0, 1, +, \cdot, ^{-}\}$ *that satisfies 1-8 above.*

Exercise 1.2 *Show the correctness of equations (1) and (2).*

Exercise 1.3 *Let $sel \in \mathrm{B}_3$ be defined by:*

$$sel(x,y,z) = \begin{cases} y & \text{if } x = 1 \\ z & \text{otherwise} \end{cases}$$

Find DNF- *and* CNF-*expressions for $sel(x,y,u+v)$.*

Exercise 1.4 *Show that $\{\oplus, \cdot\}$ and $\{sel\}$ are complete bases.*

Exercise 1.5 *Show that for $f \in \mathrm{B}_n$ it holds that*

$$f(x_1, x_2, \cdots, x_n) = x_1 f(1, x_2, \cdots, x_n) + \overline{x_1} f(0, x_2, \cdots, x_n)$$

and if in addition $f \in \mathrm{MON}_n$ then

$$f(x_1, x_2, \cdots, x_n) = x_1 f(1, x_2, \cdots, x_n) + f(0, x_2, \cdots, x_n)$$

Exercise 1.6 *Show that $f \in \mathrm{B}_n$ is monotone, if and only if f can be written as a $\{+, \cdot\}$-expression, i.e. $\{+, \cdot\}$ is a complete basis for the monotone functions.*

Exercise 1.7 *Show that $\{Th_3^2\}$ is a complete basis for the monotone functions.*

Exercise 1.8 *Find a* DNF-*expression for $Eq_5^{\{1,2\}}(\underline{x})$.*

Exercise 1.9 *A Boolean function $f \in \mathrm{B}_n$ is* affine *if and only if*

$$f(x_1, x_2, \ldots, x_n) = x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_k} \oplus c$$

for some $c \in \{0, 1\}$.

Prove that a basis $\mathcal{B} \subseteq \mathrm{B}$ is complete if and only if (i) $\mathcal{B}$ contains a non-monotone function and (ii) $\mathcal{B}$ contains a non-affine function. Hint: Prove the following subresults:

1. *The function* NEG *is a $\mathcal{B}$-expression if $\mathcal{B}$ contains a non-monotone function.*
2. *Any non-affine function $f \in \mathrm{B}_2$ has the form $f(x,y) = (x^a y^b)^c$ for some constants $a, b, c \in \{0, 1\}$*
3. *If $\mathcal{B}$ contains a non-affine function then there exists a non-affine function on two variables that is a $\mathcal{B}$-expression.*

2 Circuit Models and Complexity Measures

We have already used expressions for describing Boolean functions. In this section, we will concentrate on computing Boolean functions.

2.1 Circuits and Straight Line Programs

Our first model is a *straight line program*, which describes a sequential method for computing an expression.

Definition 2.1 *Let $\mathcal{B}$ be a basis. A straight line program over $\mathcal{B}$ with input (or indeterminates) $X = \{x_1, x_2, \cdots, x_n\}$, is a sequence of s assignments, where the i'th assignment has the form*

$$v_i \leftarrow \Box(y_1, \ldots, y_k)$$

where $y_i \in X \cup \{0,1\} \cup \{v_1, v_2, \cdots, v_{i-1}\}$ and $\Box \in \mathcal{B}$.

If $s > 0$ then the program computes the value v_s, and if $s = 0$ a special output statement specifies which value among $\{0, 1, x_1, x_2, \cdots, x_n\}$ is computed by the program.

We may draw a straight line program as a *circuit* (see figure 3). The v's in the straight line program correspond to the computation nodes in the circuit.

For a circuit N we will also let N denote the function computed by N.

2.2 Size and Depth

Definition 2.2 *Let N be a circuit (straight line program) with s computation nodes over the basis $\mathcal{B}$. The* size, *$S(N)$, of N is s. The* depth, *$D(N)$, of N is defined in terms of the depth of circuit nodes. The latter is defined inductively as follows:*

1. $D(0) = D(1) = 0$
2. $D(x_i) = 0$ *for* $i = 1, 2, \cdots, n$

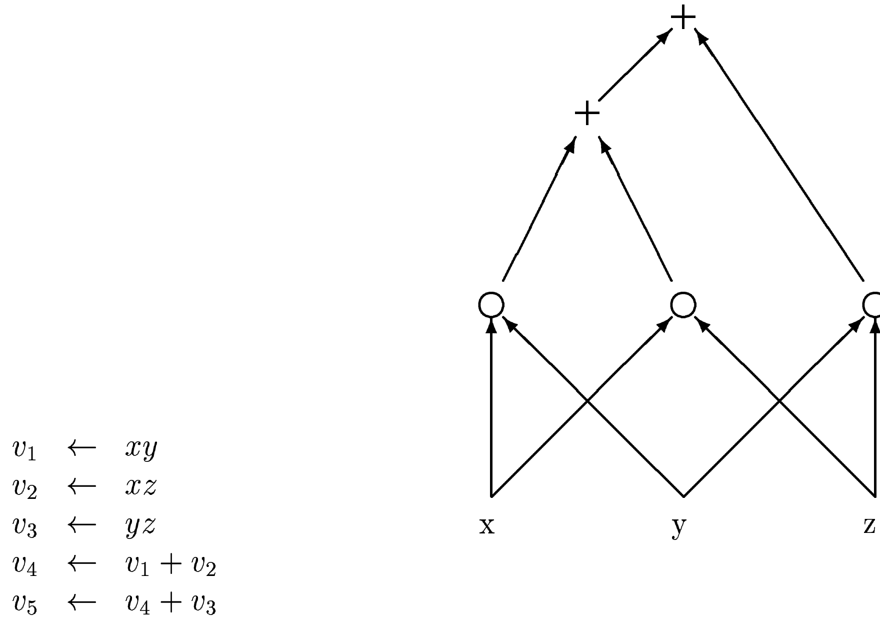

Figure 3: Straight line program and circuit for $Th_3^2(x, y, z)$

3. *If* $v_i \leftarrow \Box(y_1, y_2, \ldots, y_k)$, *then*

$$D(v_i) = max\ \{D(y_k)\} + 1.$$

$D(N)$ is defined to be $D(v_s)$.

The fan-out *of a circuit is the maximum number of times a single sub-result variable is used in the right-hand side of an assignment.*

A circuit with fan-out 1 *is called a* formula. *For a formula N we let $L(N)$ denote the size. L means formula* length.

The circuit N in figure 3 is a formula, where $S(N) = 5$ and $D(N) = 3$. Intuitively, the measures size and depth correspond to sequential time (or parallel work) and parallel time at the bit level, respectively. This intuition will be formalised in section 4.

Definition 2.3 *For a basis $\mathcal{B}$ and a function $f \in \mathrm{B}$ we define $M_{\mathcal{B}}(f)$, where M is S, D or L to be*

$$M_{\mathcal{B}}(f) = min\ \{M(N) | N \textit{ is a circuit over } \mathcal{B} \textit{ that computes } f\}$$

For the default basis $\{\cdot, +, ^-\}$ *we omit a basis specification. The monotone basis* $\{\cdot, +\}$ *is specified by an "m".*

Usually, we are only interested in results about the default basis (or some other complete basis) and the monotone basis. However, many of the results we prove can also be shown for arbitrary non-complete bases, but the proofs get rather technical, and some of them are only published in Russian!

2.3 Upper Bounds

Proposition 2.1 *a) For all* $f \in \mathrm{B}$ $(f \in \mathrm{MON})$ *it holds that*

a1) $S(f) \leq L(f) \leq n2^n$ $\quad (S_m(f) \leq L_m(f) \leq n2^n)$

a2) $D(f) \leq S(f)$ $\quad (D_m(f) \leq S_m(f))$

a3) $D(f) \leq n + \lceil \log n \rceil + 1$ $\quad (D_m(f) \leq n + \lceil \log n \rceil)$

b) For all $f \in \mathrm{MON}$ *it holds that*

$$M(f) \leq M_m(f) \text{ for } M \in \{S, D, L\}$$

One may verify the upper bounds in proposition 2.1 by considering DNF (the DNF-formula for either f or $\overline{f}$ will have size bounded by $n2^n$).

Formulas are circuits with fan-out at most 1. We have defined a special complexity measure L, based on formula. It is a natural question, whether one obtains significantly different complexity measures by bounding the fan-out to other constants such as 2 or 3. As we shall see that is not the case.

Theorem 2.2 *For all circuits N (over the default basis or the monotone basis) with unbounded fan-out, there exists an equivalent circuit N' with fan-out 2 such that* $S(N') = O(S(N))$.

Proof. Figure 4 shows how fan-out can be reduced in a special case. The construction generalises in the obvious way. For a node with fan-out k the transformation creates $(k-2)$ extra AND-nodes.

If N has s nodes with fan-out $k_1, k_2, \cdots k_s$ then $\sum_{i=1}^{s} k_i$ is a bound on the number of extra nodes. But since each node has fan-in at most two, we also have that $\sum_{i=1}^{s} k_i \leq 2s$, which implies the theorem. □

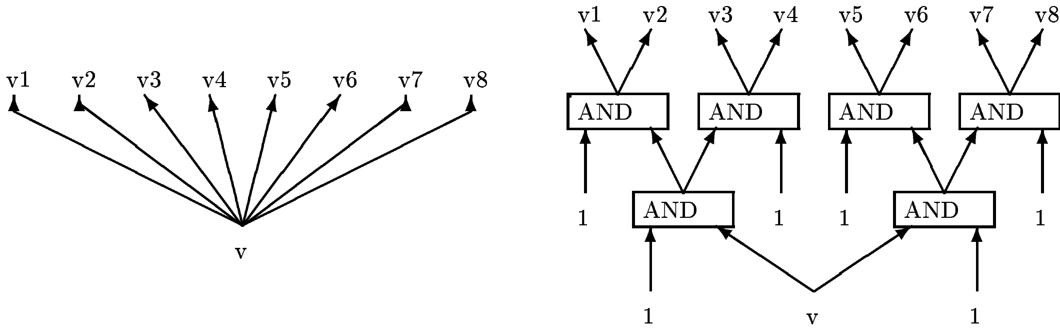

Figure 4: Fan-out 8 reduced to fan-out 2

There is more to be said about the complexity measures and their mutual relationship than what follows from the inequalities in proposition 2.1. We begin by improving the universal upper bound on size. Let $S_n = \max\{S(f) | f \in \mathrm{B}_n\}$ and $L_n = \max\{L(f) | f \in \mathrm{B}_n\}$.

Theorem 2.3 $L_n = O(2^n)$.

Proof. We know that

$$f(x_1, \ldots, x_n) = x_n f(x_1, \ldots, x_{n-1}, 1) + \overline{x_n} f(x_1, \ldots, x_{n-1}, 0) \tag{3}$$

This suggests a formula construction for f, where the two occurrences of f on the right-hand side are expanded recursively, by letting x_{n-1} assume the role of x_n.

This results in a recurrence

$$L_n \leq 2 * L_{n-1} + 4$$

$$L_1 \leq 1$$

that gives a bound

$$L_n \leq 5 \cdot 2^{n-1} - 4 = O(2^n)$$

which is an improvement compared to the DNF based bound. □

The construction can be improved by allowing multiple fan-out.

Theorem 2.4 (Lupanov, 1958) $S_n = O(\frac{2^n}{n})$.

Proof. Note that when the formula in the proof of the previous theorem has been expanded recursively i steps using equation (3), then we have 2^i sub-functions of f on the right-hand side, each containing $n-i$ variables. So for each recursion step we get more functions and fewer variables. There are only $2^{2^{n-i}}$ distinct functions of $(n-i)$ variables, and with i increasing towards n, we eventually get $2^i > 2^{2^{n-i}}$. For such i there are least 2 of the sub-functions that are functionally identical.

At this point we switch to look at circuits rather than formulas. This allows us to compute a function with multiple occurrences only once and reuse the result as appropriate (the dynamic programming technique). To simplify the analysis, we implement this approach as follows:

1. For $j := 1$ to $n-i$ we compute all functions in B_j on the given input (!)
2. For $j := n$ down-to $n-i+1$ we expand the formula as described above, but then, instead of expanding any further, we use the relevant precomputed function values.

Equation (3) tells us that when we have computed all functions in B_{j-1} then we may compute all the functions in B_j with only $O(|\mathrm{B}_j|)$ additional gates. Hence, for the implementation of the lower part of the circuit (item 1.), we need in total

$$O(\sum_{j=1}^{n-i} |\mathrm{B}_j|) = O(|\mathrm{B}_{n-i}|) = O(2^{2^{n-i}})$$

gates. For the upper part of the circuit (item 2.), we need $O(2^i)$ gates. In total, for any choice of i, it holds that

$$S_n = O(2^{2^{n-i}} + 2^i)$$

By choosing $i = \lceil n - \log n + 1 \rceil$ we get $S_n = O(\frac{2^n}{n})$. □

We shall later see (section 5) that the above construction is optimal (up to a multiplicative constant), i.e. there exists a family of functions $\{f_n\}$ with $S(f_n) = \Theta(\frac{2^n}{n})$. The bound on L_n can be improved to $O(\frac{2^n}{\log n})$, but we omit the construction.

2.4 Depth and Formula Size

In this section we initiate the study of relations between complexity measures. The first result shows a tight connection between depth and formula size.

Theorem 2.5 (Spira, 1971) *For all $f \in \mathrm{B}$ it holds that $D(f) = O(\log L(f))$. If $f \in \mathrm{MON}$ it holds in addition that $D_m(f) = O(\log L_m(f))$.*

Proof. We will show that we to a given monotone formula N, can construct an equivalent monotone circuit N' of depth $O(\log L(N))$. By using DeMorgan's law this result translates to non-monotone measures.

If $L(N) > 2$, then we can for any natural number $\alpha \leq L(N)$ find a sub-formula F of N with the property that $\alpha \leq L(F) < 2\alpha$ (The reader is urged to work out the details of this argument). In particular, we choose a sub-formula F such that

$$\lfloor \frac{1}{3} L(N) \rfloor \leq L(F) < 2 \lfloor \frac{1}{3} L(N) \rfloor$$

Let N_0 (N_1) be the circuit that results from N by replacing the sub-formula F with a 0 (1), i.e.

$$N(\underline{x}) = \begin{cases} N_1(\underline{x}) & \text{if } F(\underline{x}) = 1 \\ N_0(\underline{x}) & \text{otherwise} \end{cases}$$

In addition, it holds that $N_0(\underline{x}) \leq N_1(\underline{x})$, since N is monotone.

Define a new Boolean function *monotone selection* by $\mathrm{MONSEL}(x, y, z) = xy + z$. It holds that

$$N(\underline{x}) = \mathrm{MONSEL}(F(\underline{x}), N_1(\underline{x}), N_0(\underline{x}))$$

(see figure 5) When used recursively, the method gives a recurrence for the depth as a function of the size L

$$D(L) \leq \begin{cases} L & \text{if } L \leq 2 \\ D(\lceil \frac{2}{3} L \rceil) + 2 & \text{if } L > 2 \end{cases}$$

since $L(N_i) \leq L(N) - L(F) \leq \lceil \frac{2}{3} L(N) \rceil$ for $i = 0, 1$. The recurrence gives $D(L) = O(\log L)$. □

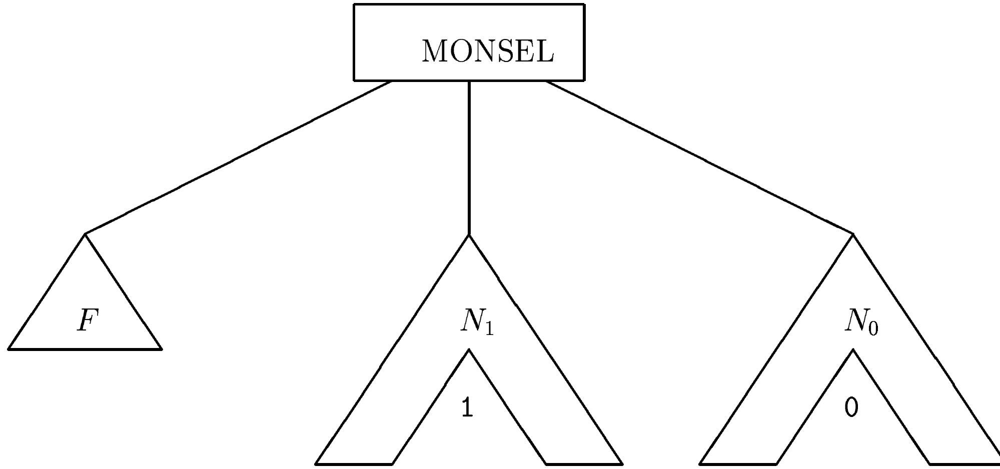

Figure 5: Reduction of formula depth.

Note that the size of the constructed formula is $O(L(N)^2)$. Using another method (we omit to do so) it is possible, simultaneously, to obtain circuit depth $O(\log L(N))$ and circuit size linear in $L(N)$.

By expanding a circuit into a formula, we can get a reverse relationship: $L(f) = O(2^{D(f)})$. To summarise, we have proven $L(f) = 2^{\Theta(D(f))}$, which means that depth and formula size are equivalent complexity measures up to a constant factor for depth and a constant exponent for formula size.

2.5 Depth and circuit size

Since formulas are a special type of circuit, one might naturally ask whether the result of the last section generalise to circuits with unbounded fan-out. We shall see a sort of generalised construction in the proof of the following theorem.

Theorem 2.6 (Paterson and Valiant, 1976) *For all $f \in \mathrm{B}$ it holds that*

$$D(f) = O(\frac{S(f)}{\log S(f)}).$$

Proof. Let N be a circuit computing the function f. Similar to the construction of low depth formulas, we try to divide the circuit in two approximately equally large parts. In the case of formulas, a division was possible such that only a single value needed to pass from the lower part of the circuit to the upper part of the circuit during evaluation. Such a division may not be possible for circuits in general.

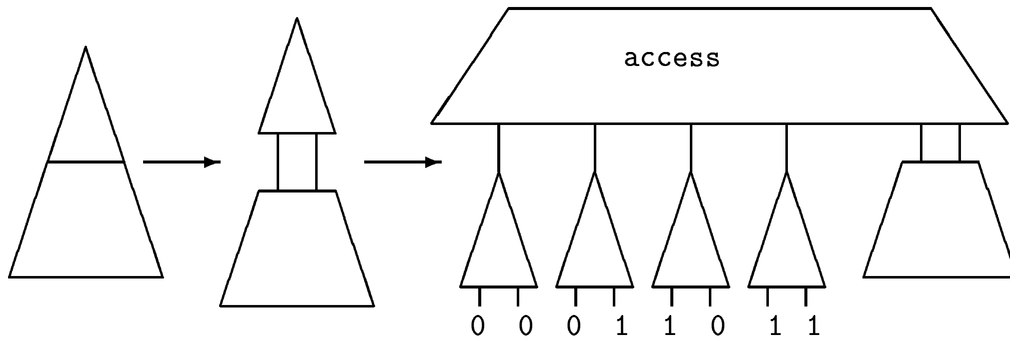

Figure 6: Reduction of circuit depth

However, let us assume that we have somehow got a division where values from k nodes only in the lower part of the circuit are used in the upper part of the circuit (we describe later, how to obtain a small k, simultaneously with making the parts almost equally large). This means that there are in principle 2^k distinct outcomes possible when evaluating the lower part. We shall make a construction, which in the first step evaluates the upper part for all these 2^k possible values in parallel, simultaneously to finding the actual value of the lower part. In the second step the value of the lower part is used to select the appropriate result for the entire circuit (see figure 6). For this purpose, we need a generalised select-function, namely *access* defined by $access(x_0, x_1, \ldots, x_{2^k-1}, y_{k-1}, y_{k-2}, \ldots, y_0) = x_{[\underline{y}]}$, where $[\underline{y}]$ denotes the number represented in binary by $\underline{y}$. The function *access* may be computed by a $\{+, \cdot, ^-\}$-circuit of depth $k + \lceil \log k \rceil + 2$ (Exercise 2.2).

In the cases, where k cannot be chosen sufficiently small for the construction using access to reduce depth, it turns out that an appropriate division leads to small depth by simply first computing the lower part and then afterwards the upper part.

In either case, the entire method is used recursively on both the lower and upper parts. The resulting circuit may be enormous in size, but our concern is depth only.

We will now describe how to find an appropriate division of the circuit. For technical simplicity, we will assume that the fan-out of any gate is at most 2 (using theorem 2.2), and of course, the fan-in is also at most 2. We need the concept of an *internal wire*. An internal wire is simply a wire between two gates in a given circuit. Let c be the number of internal wires in N. We will look at a division of N in two parts A and B such that A can be computed before B. Let a and b be the number of internal wires in A and B, respectively, and let k be the number of wires between A and B.

Let us now consider what happens, if we change the A-B division slightly by moving an arbitrary gate from A to B. The value of a is reduced by at most 2 and the value of b is increased by at most 2, so the value of $a-b$ is reduced by at most 4. If we start with all gates in A and none in B, and we move all gates from A to B one by one, then the value of $a-b$ will change from c down to $-c$, decreasing no more than 4 in any single step. Therefore, there must be some intermediate step at which

$$|a-b| \leq 2.$$

Let the corresponding A and B be the division we use in the recursive construction. Let us first argue intuitively for this choice. If we put $e = \max(a,b)$ and use that $a+b+k=c$, then we get

$$c \leq 2e+k \leq c+2.$$

This means that if k is large, then both A and B have few internal wires implying that they can both be computed by low depth circuits and their direct combination will also have fairly low depth. If k is small, we can use the *access*-construction.

For making a formal analysis, we introduce a function

$$d(c) = \max\{D(f) | f \text{ may be computed by a circuit with } c \text{ internal wires}\}$$

The construction using the *access*-circuit leads to an inequality:

$$\begin{aligned} d(c) &\leq \max_{c \leq 2e+k \leq c+2} (d(e)+k+\lceil \log k \rceil + 2) \leq \\ &\max_{c \leq 2e+k \leq c+2} (d(e)+2k+2) \leq \\ &\max_{1 \leq e \leq \frac{c+2}{2}} (d(e)+2(c-2e)+6) \end{aligned}$$

Since we may compute f by first computing A and then B, we get a second inequality

$$d(c) \leq \max_{1 \leq e \leq \frac{c+2}{2}} 2d(e)$$

When combining the two inequalities, the following recurrence for d results:

$$d(c) \leq \max_{1 \leq e \leq \frac{c+2}{2}} \min\{2d(e), d(e) + 2(c - 2e) + 6\} \tag{4}$$

Containing both *min* and *max*, (4) looks potentially difficult to solve. However, it is possible.

We will use that $f_1 : e \mapsto 2d(e)$ and $f_2 : e \mapsto d(e)+2(c-2e)+6$ are increasing and decreasing functions, respectively (exercise 2.3). For increasing f_1 and decreasing f_2 over an interval I, it holds that

$$\max_{x \in I} \min\{f_1(x), f_2(x)\} \leq \max\{f_1(y), f_2(y)\} \quad \text{for all } y \in I.$$

In our case, this implies

$$d(c) \leq \max\{2d(e_0), d(e_0) + 2(c - 2e_0) + 6\} \quad \text{for all } e_0 \in [1, \tfrac{c+2}{2}] \tag{5}$$

At this point, we dare make a qualified guess at a solution:

$$d(t2^{t+3} + 3) \leq 2^{t+5} \quad \text{for} \quad t \geq 0 \tag{6}$$

We will show (6) by induction in t. The basis $t = 0$ is obvious. For the induction step we use (5) with $c = (t+1)2^{t+4} + 3$ and $e_0 = t2^{t+3} + 3$

$$\begin{aligned} &d((t+1)2^{t+4} + 3) \leq \\ &\max\{2d(t2^{t+3} + 3), \\ &\quad d(t2^{t+3} + 3) + 2((t+1)2^{t+4} + 3 - 2(t2^{t+3} + 3)) + 6\} \leq \\ &\max\{2^{t+6}, 2^{t+6}\} = 2^{t+6} \end{aligned}$$

(6) implies that $d(c) = O(c/\log c)$. Since there are at most two internal wires for every gate in a circuit, we have proven the theorem. □

The above theorem is the best known result of its type. Since by proposition 2.1 all functions have depth $O(n)$, the theorem is only interesting for a small size

range. It is unknown, whether the result is optimal. It is consistent with our present knowledge if in fact $D(f) = O(\log S(f))$ for all f. Such an improvement would imply that all computations parallelise well. It is also unknown, whether the size blow-up in the present construction can be avoided.

Surprisingly, it is known that for almost all functions f that $D(f) = O(\log S(f))$ (without a size blow-up) implying that almost all functions parallelise well. Unfortunately, "almost all" is not known to include any of the relatively few functions that arise from natural problems. This may appear paradoxical and we discuss the situation further in section 5.

2.6 The Complexity of Function Families

The introduction of complexity measures puts significance to function families compared to single functions. Some fixed function f of 7 variables has complexity, say $S(f) = 42$, which is not very interesting. However, for a family of functions $\{f_n\}$ it may hold that $S(f_n) \leq 3n^2 + n$, which is much more enlightening and allows comparison with other families to improve understanding of the first family. In addition, we have the possibility of using asymptotic notation when making statements about $\{f_n\}$, e. g. $S(f_n) = O(n^2)$ or $S(f_n) = 3n^2 + o(n^2)$, if we want greater precision.

We can also form classes of function families by putting restrictions on their complexity in various complexity measures. Some basic classes are:

- pC, the class of function families that are computed by polynomial size circuit families. Formally, $\mathrm{pC} = \{\{f_n\} | S(f_n) = n^{O(1)}\}$.
- pF, the class of function families that are computed by polynomial size formula families. Formally, $\mathrm{pF} = \{\{f_n\} | L(f_n) = n^{O(1)}\}$.
- NC^i, the class of function families that are computed by polynomial size circuit families of depth $O(\log^i n)$.

From section 2.4 it follows that

Theorem 2.7 $\mathrm{pF} = \mathrm{NC}^1 \subseteq \mathrm{pC}$

We will define more complexity classes as the need arises.

2.7 Projection

The concept of complexity classes plays an important role in identifying hard functions. The reader is assumed to be familiar with the theory of NP-completeness (say from dAlg). Recall that every problem in NP reduces to any given NP-complete problem by a simple reduction such that an efficient solution for the NP-complete problem implies an efficient solution for all problems in NP. Since it is widely believed that not all problems in NP are easy, we are lead to believe that any single NP-complete problem is hard.

Similarly, we do not know whether pC is contained in pF, i.e. whether all functions in pC have good solutions of small depth, but experts in parallel algorithms would be surprised if that is the case. The full analogy to the NP-completeness theory could be carried through, if we had a suitable kind of reduction, by which we could show certain problems to be complete for pC and hence unlikely to have good parallel solutions (and we have understood why that is so). We need a reduction that preserves membership of pF.

Definition 2.4 *Let* $f \in \mathrm{B}_n$, $g \in \mathrm{B}_m$ *be two Boolean functions. We say that* f *is a* projection *of* g*, if*

$$f(x_1, x_2, \ldots x_n) = g(\sigma(y_1), \sigma(y_2), \ldots, \sigma(y_m)),$$

where

$$\sigma : \{y_1, y_2, \ldots, y_m\} \rightarrow \{0, 1, x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}.$$

Example 2.1 $+$ *and* $\cdot$ *are projections of* Th_3^2*, since* $x + y = Th_3^2(1, x, y)$ *and* $x \cdot y = Th_3^2(0, x, y)$.

The function $f(x, y, z) = x(\bar{y} + z)$ *is a projection of* Th_4^3 *since* $f(x, y, z) = Th_4^3(x, x, \bar{y}, z)$.

Intuitively, the existence of a projection of g onto f expresses that a chip for g can be used for computing f in a simple way (see figure 7).

Definition 2.5 *Let* $\{f_n\}, \{g_n\}$ *be families of functions. We say that* $\{f_n\}$ *is a projection of* $\{g_n\}$*, if for all* n *there exists an* m *such that* f_n *is a projection of* g_m*. The family* $\{g_n\}$ *is said to be* universal *for a class of families* C*, if for all* $\{f_n\} \in C$ *it holds that* $\{f_n\}$ *is a projection of* $\{g_n\}$.

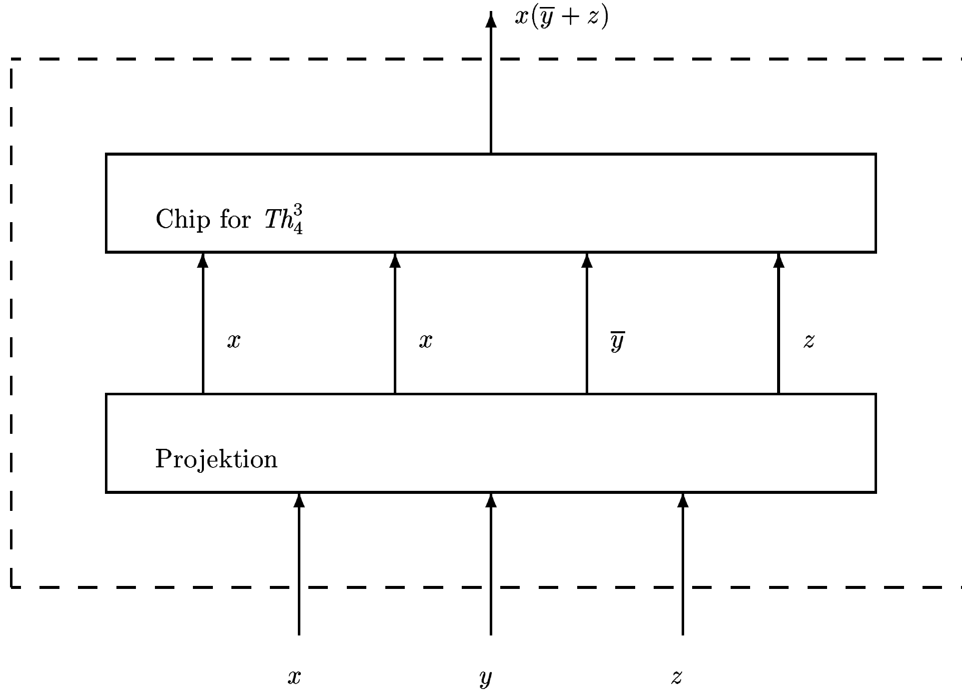

Figure 7: Chip construction by projection

Intuitively, chips for members of universal families are general purpose chips that are capable of computing many different functions. Consider the example

$$\mathrm{DNF}_{n^2}(\underline{x}) = \sum_{i=1}^{n} \prod_{j=1}^{n} x_{ij}$$

Since every function has a DNF-expression, we see that DNF is universal for B. However, for most functions occurring in practice, it would be rather inefficient to use a DNF-chip for their computation. To deal with this problem, we introduce restricted versions of projection and universality.

Definition 2.6 *Let $\{f_n\}, \{g_n\}$ be families of functions. We say that $\{f_n\}$ is a* p-projection *of $\{g_n\}$ ($\{f_n\} \leq_p \{g_n\}$), if there exists a polynomial q, such that for all n there is an $m \leq q(n)$ such that f_n is a projection of g_m. The family $\{g_n\}$ is said to be* p-universal *for a class of families C, if for all $\{f_n\} \in C$ it holds that $\{f_n\} \leq_p \{g_n\}$. If, in addition, it holds that $\{g_n\} \in C$, we say that $\{g_n\}$ is p-complete for C.*

p-projection turns out to be the proper type of reduction, since the class pF is closed under p-projections (the composition of two polynomials is a polynomial).

Hence, when a problem is p-complete for pC, it can not be in pF unless pF = pC. We will see examples of such problems later. The concept of projection was introduced by Skyum and Valiant (1985).

2.8 Bases with unlimited fan-in

Our favourite basis $\{+, \cdot, ^-\}$ and all other bases we have seen so far are finite. However, an infinite basis is conceivable and for some purposes more appropriate than a finite one.

Definition 2.7 *Let $+_n$ be the function in B_n that takes the OR of n variables, i.e. $+_n(x_1, \ldots, x_n) = x_1 + x_2 + \ldots + x_n$. Similarly, let $\cdot_n$ be the AND of n variables and define the basis $\mathcal{U} = \{^-, +_2, \cdot_2, +_3, \cdot_3, \ldots\}$.*

A circuit over $\mathcal{U}$ is a circuit of unbounded fan-in. We shall later make a correspondence between such circuits and the CRCW-model of parallel computations (see section 4).

There are functions (such as OR of n variables) that have size $O(n)$ in the basis $\{+, \cdot, ^-\}$, but have size $O(1)$ in $\mathcal{U}$. However, there is a bound to the possible discrepancy between the two size measures.

Theorem 2.8 *For all $f \in B_n$ it holds that $S(f) \leq n(S_{\mathcal{U}}(f))^2$*

Proof. Exercise 2.6. □

For depth the situation is different.

Theorem 2.9 *For all f, $D_{\mathcal{U}}(f) \leq 3$.*

Proof. The DNF-expression for f can be implemented as a fan-in 2^n OR of fan-in n AND's. □

Since the size over $\mathcal{U}$ is significantly less than the size over the standard basis, and the depth measure is trivialised, one might think that the concept of unbounded fan-in is ill-conceived. However, when one limits depth and size simultaneously, an interesting complexity measure results. Allowing less than exponential size, it is not clear how to compute all functions in constant depth. In fact, the best result known is

Theorem 2.10 *Let $f \in \mathrm{B}_n$ be a function that is computed by an ordinary circuit of size s and depth d. For every ϵ, f is computed by a $\mathcal{U}$-circuit of size $O(sn^\epsilon)$ and depth $O(\lceil d/\log\log n\rceil)$.*

Proof. Partition C into d layers $C_1, C_2, \ldots C_d$ such that no gate in layer j takes input from a gate in layer k if $k \geq j$. Group the layers into blocks with $b = \frac{1}{2}\log\log n$ consecutive layers in each block. A gate in some block may be regarded as computing a function of at most 2^b gates in previous layers or inputs. Such a function has a DNF-expression, which is computed by a $\mathcal{U}$-circuit of depth 2 and size $2^{2^b}+1$, the latter value being $o(n^\epsilon)$ for every ϵ. We may replace the original gate with this unbounded fan-in circuit. This is done for all gates in the original circuit. The resulting depth is twice the number of layers. □

In general, this construction is optimal. However, the proof is non-trivial and we defer it till section 5

We will in the later sections use a few complexity classes defined in terms of $\mathcal{U}$-circuits.

Definition 2.8 *Let AC^i be the class of function families that are computed by a family of $\mathcal{U}$-circuits of polynomial size and depth $O(\log^i n)$.*

Let $\mathcal{T}$ be the basis $\{^-\} \cup \{Th_{2n}^{n+1}\}_{n\geq 1}$. Let TC^i be the class of function families that are computed by a family of $\mathcal{T}$-circuits of polynomial size and depth $O(\log^i n)$.

TC^0 may be regarded as the class of functions that can be computed by artificial neural networks (of the type currently used). We will later see (exercise 2.7 and section 3), that $\mathrm{NC}^i \subseteq \mathrm{AC}^i \subseteq \mathrm{TC}^i \subseteq \mathrm{NC}^{i+1}$.

2.9 Branching Programs

To make a precise combinatorial characterisation of the machine resource space (in section 4), we will need a new model.

Definition 2.9 *Let n be a natural number. A (nondeterministic)* branching program *is a directed graph where edges are labelled either "true", "$x_i = 0$" or "$x_i = 1$", for some $1 \leq i \leq n$, and the graph has two nodes labelled s and t, respectively.*

The program computes the function $f_n \in \mathrm{B}_n$ defined by

$$f_n(\underline{x}) = \begin{cases} 1 & \textit{if there is a path from } s \textit{ to } t \textit{ labelled} \\ & \textit{with true assertions about } \underline{x}\textit{, exclusively,} \\ 0 & \textit{otherwise} \end{cases}$$

A branching program is deterministic, if (i) every node has at most two outgoing edges, and if (ii) there is two outgoing edges from a node v, they must be labelled $x_i = 0$ and $x_i = 1$ (for some i), respectively.

Note that for a deterministic branching program it is possible to decide the existence of a path without backtracking. A deterministic branching program may be regarded as a generalisation of *decision trees*, since a decision tree can be converted into a deterministic branching program by adding edges labelled "true" from all leaves labelled "true" to a new node designated t.

We can define a new complexity measure based on branching programs. It turns out to be related to a measure defined in terms of projections.

Definition 2.10 *We define the complexity measure $B(f)$ to denote the size (number of nodes) of the smallest branching program computing $f \in \mathrm{B}$.*

Given a universal family $\{g_m\}$, let the projection complexity $C_g(f)$ of f with respect to $\{g_m\}$ be the minimum m such that f is a projection of g_m.

Theorem 2.11 $\quad (B(f))^2 = C_{DPATH}(f)$

Proof. One may claim that the theorem holds per definition: If f is a projection of $DPATH_{n^2}$, then we can from the projection directly construct a branching program with n nodes for f, and conversely. □

Corollary 2.12 *A family $\{f_n\}$ of Boolean functions has polynomial size branching programs if and only if $\{f_n\}$ is a p-projection of $DPATH$.*

If a family $\{f_n\}$ of Boolean functions has polynomial size branching programs then $\{f_n\} \in \mathrm{AC}^1$.

Proof. The first part is immediate from the previous theorem. For the second part, we provide an argument: By exercise 2.8, $DPATH \in \mathrm{AC}^1$, and by exercise 2.9, AC^1 is closed under p-projections. We conclude that functions with polynomial size branching programs are in AC^1 (and therefore have small depth). □

Exercises

Exercise 2.1 *Find* $S(Eq_5^{\{1,2\}})$, $D(Eq_5^{\{1,2\}})$ *and* $L(Eq_5^{\{1,2\}})$.

Exercise 2.2 *Show that the access-function can be computed within the asserted depth.*

Exercise 2.3 *Show that the functions* $2d(e)$ *and* $d(e)+(c-2e)+6$ *in the proof of theorem 2.6 are monotone as asserted. (Hint: show that* $d(e) \leq d(e+1) \leq d(e)+1$).

Exercise 2.4 *Formulate and prove monotone variants of theorems 2.4 and 2.6.*

Exercise 2.5 *Let* $f \in \mathrm{B}_n$. *The* dual *function to* f, f^*, *is defined by*

$$f^*(x_1, x_2, \ldots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)}$$

f *is called* self dual, *if* $f^* = f$.

1. *Find the dual function to* $Th_3^2(x,y,z)$.
2. *Find* f^{**}, $(fg)^*$ *and* $(f+g)^*$.
3. *Show that* $f \to f^*$ *is a bijection on the sets* B, MON *and* SYM.
4. *Let* $\mathcal{S} = \{^-, Th_3^2\}$. *Show that the set of functions definable by* $\mathcal{S}$*-expressions without the use of the constants* $0,1$ *coincides with the set of self dual functions.*
5. *Let* $\tilde{S}_{\mathcal{S}}(f)$ $(\tilde{D}_{\mathcal{S}}(f))$ *denote minimum size (depth) of an* $\mathcal{S}$*-circuit (that does not use the constants* $0,1$) *for* f. *Show that for self dual* f *it holds that* $\tilde{S}_{\mathcal{S}}(f) = O(S(f))$ *and* $\tilde{D}_{\mathcal{S}}(f) = O(D(f))$.

Exercise 2.6 *Prove theorem 2.8.*

Exercise 2.7 *Show that* $\mathrm{AC}^i \subseteq \mathrm{TC}^i$.

Exercise 2.8 *Show that* $DPATH \in \mathrm{AC}^1$.

Exercise 2.9 *Show that* AC^i *is closed under p-projection.*

Exercise 2.10 *Let brackets be the family of functions, defined by* $brackets_{2n}(\underline{x}) = 1$*, if and only if* $\underline{x}$ *is a correctly balanced string of parentheses, when* 0 *represents a left parenthesis and* 1 *represents a right parenthesis.*

Show that $brackets \in \mathrm{TC}^0$.

Exercise 2.11 *For every constant* $k > 0$*, let* AC^0_k *be the class of function families that can be computed by a* $\mathcal{U}$*-circuit of depth* $\leq k$*. It holds that* $\mathrm{AC}^0_k \neq \mathrm{AC}^0_{k+1}$ *(this may be assumed without proof).*

Show that AC^0 *can not have a p-complete family.*

Exercise 2.12 *Show that* pF *has a p-complete family.*

Literature

1. Lupanov, O. B. (1958) A Method of Circuit Synthesis. *Izv. V. U. Z. Radiofiz.* **1**, 120–140.

2. Paterson M. S. and Valiant, L. G. (1976) Circuit Size is Nonlinear in Depth. *Theoretical Computer Science* **2**, 397–400.

3. Skyum S. and Valiant, L. G. (1985) A Complexity Theory Based on Boolean Algebra. *Journal of the ACM* **32**, 484–502.

4. Spira, P. M. (1971) On Time-Hardware Complexity Trade-offs for Boolean Functions. In *Proceedings 4th Hawaii International Symposium on System Sciences*, 525–527.

3 Constructions I

In the first part of this section, we describe the construction of efficient circuits for addition of binary numbers. Such circuits are fundamental – one can hardly imagine a computer without some hardwired addition function. We will need the circuits in section 4, when simulating Turing machines and in the present section they are used for efficient computation of general symmetric functions.

In the second part we look at efficient computation of functions that are both monotone and symmetric. They can be computed efficiently as any other symmetric function using the standard $\{+,\cdot,^{-}\}$-basis. It is more difficult to find efficient circuits when we restrict ourselves to use a monotone basis (binary addition is no longer applicable). We construct monotone solutions based on sorting networks, and we prove the existence of small monotone formula by a (non-constructive) probabilistic method. With this knowledge one might conjecture that if a monotone problem has an efficient general solution (using negation) then it also has an efficient monotone solution, but such a conjecture is known to be false. We are later going to prove super-polynomial lower bounds for monotone solutions to some monotone problems (see section 7).

3.1 Addition of two numbers

Let $n \in \{1,2,..\}$, $\underline{x} = (x_{n-1},\ldots,x_1,x_0)$ and $\underline{y} = (y_{n-1},\ldots,y_1,y_0)$. We define $[\underline{x}] = \sum_{i=0}^{n-1} x_i 2^i$, i.e. $[\underline{x}]$ is the natural number with binary representation $\underline{x}$. Conversely, if $t < 2^n$ is a natural number then $b_n(t)$ is the n-bit binary representation of t. The *addition function* $add_n \in \mathrm{B}_{2n,n+1}$ is given by

$$[add_n(\underline{x},\underline{y})] = [\underline{x}] + [\underline{y}],\ \ \underline{x},\underline{y} \in \{0,1\}^n.$$

In this section we will determine the complexity of the addition function by constructing a series of addition circuits.

3.1.1 Method 1

The most obvious construction is perhaps a simulation of the "school" method for adding numbers. Let $\underline{z} = add_n(\underline{x},\underline{y})$. We may compute $\underline{z}$ by the following algorithm.

$\underline{z} \leftarrow add_n(\underline{x},\underline{y})$:

$c_0 \leftarrow 0$
for $i := 0$ **to** $n-1$ **do**
  $z_i \leftarrow x_i \oplus y_i \oplus c_i$
  $c_{i+1} \leftarrow Th_3^2(x_i, y_i, c_i)$
od
$z_n \leftarrow c_n$

Figure 8 illustrates an execution of the algorithm. In the algorithm, c_i denotes

i	5	4	3	2	1	0
c_i	1	0	0	1	1	0
x_i	–	1	0	0	1	1
y_i	–	1	1	0	0	1
z_i	1	0	1	1	0	0

Figure 8: The "school" method for addition

the carry at the i'th position. The program may be unfolded into a straight line program of size $O(n)$ and depth $O(n)$ over an arbitrarily chosen complete basis. The size is obviously optimal:

Proposition 3.1 $S(add_n) = \Theta(n)$

3.1.2 Method 2

In section 2 it was shown that *any* function of n variables has depth at most $n + \lceil \log n \rceil$, and therefore the depth of the construction in method 1 gives us no interesting information. We want to improve this depth.

Since c_{i+1} depends on c_i it seems necessary to compute c_i before computing c_{i+1}, which inevitably results in depth $\Omega(n)$. However, we may use the fact that a carry can take only two distinct values. Let $addc_n \in \mathrm{B}_{2n+1,n+1}$ be given by

$$[addc_n(\underline{x}, \underline{y}, c)] = [\underline{x}] + [\underline{y}] + [c], \quad \underline{x}, \underline{y} \in \{0,1\}^n, c \in \{0,1\}.$$

An $addc_n$ circuit may be projected into an add_n circuit by giving c the value 0. We are going to construct a shallow circuit for $addc_n$ using divide and conquer.

We want to compute $\underline{z} = addc_n(\underline{x}, \underline{y}, c)$. Let $\underline{x}^{\text{right}} = (x_{\lfloor n/2 \rfloor - 1}, \ldots, x_0)$, $\underline{x}^{\text{left}} = (x_{n-1}, \ldots, x_{\lfloor n/2 \rfloor})$ and let $\underline{y}^{\text{right}}$ be $\underline{y}^{\text{left}}$ defined similarly.

Define $\underline{z}^{\text{right}} = addc_{n/2}(\underline{x}^{\text{right}}, \underline{y}^{\text{right}}, c)$, $\underline{z}^{\text{left},0} = addc_{n/2}(\underline{x}^{\text{left}}, \underline{y}^{\text{left}}, 0)$ and $\underline{z}^{\text{left},1} = addc_{n/2}(\underline{x}^{\text{left}}, \underline{y}^{\text{left}}, 1)$. It holds that

$$z_i = \begin{cases} z_i^{\text{right}} & \text{for } 0 \le i < \lfloor n/2 \rfloor \\ sel(z_{\lfloor n/2 \rfloor}^{\text{right}}, z_{i-\lfloor n/2 \rfloor}^{\text{left},1}, z_{i-\lfloor n/2 \rfloor}^{\text{left},0}) & \text{for } \lfloor n/2 \rfloor \le i \le n \end{cases}$$

This suggests a recursive construction of a circuit for $addc_n$ (note the similarity to the proof of theorem 2.5). Let s_n and d_n denote the size and depth respectively of this circuit. We find that

$$\begin{aligned} s_n &= 2s_{\lfloor n/2 \rfloor} + s_{\lceil n/2 \rceil} + O(n), \\ d_n &= \max(d_{\lfloor n/2 \rfloor}, d_{\lceil n/2 \rceil}) + O(1). \end{aligned}$$

These recurrences have the solutions $s_n = O(n^{\log_2 3})$ and $d_n = O(\log n)$. Since, obviously, $D(add_n) = \Omega(\log n)$ we also have that

Proposition 3.2 $\qquad D(add_n) = \Theta(\log n)$

By the use of dynamic programming, the construction may be improved to give a circuit of size $O(n \log n)$ and depth $O(\log n)$ (Exercise 3.1).

3.1.3 Method 3 (Carry-look-ahead)

We have determined $S(add_n)$ and $D(add_n)$, but so far we have not managed to construct a single family of circuits that are optimal with respect to both measures. If such a family did not exist, we would say that add_n had a *size-depth trade-off*. However, such a family does exist as we shall see.

The bottleneck is the computation of the carry bits. Once we know all the c_i's we may compute $\underline{z}$ by a circuit of size $O(n)$ and depth $O(1)$. In order to understand how a carry c_{i+1} from position i is determined, it is helpful to distinguish three types of positions.

- Both x_i and y_i are 0. In this case $c_{i+1} = 0$, independent of the value of c_i. We call i a C-position (carry clear).
- Precisely one of x_i and y_i is 1. In this case $c_{i+1} = c_i$, and we call i a P-position (carry propagate).

i	5	4	3	2	1	0
c_i	1	0	0	1	1	0
x_i	–	1	0	0	1	1
y_i	–	1	1	0	0	1
v_i	–	S	P	C	P	S
z_i	1	0	1	1	0	0

Figure 9: Values of v_i

$\circ$	C	P	S
C	C	C	C
P	C	P	S
S	S	S	S

Figure 10: $(M, \circ)$

- Both x_i and y_i are 1. In this case $c_{i+1} = 1$, independent of the value of c_i. We call i an S-position (carry set).

Let M denote the set of symbols $\{C, P, S\}$. To any pair of inputs $\underline{x}, \underline{y}$ we attach a vector $\underline{v} \in M^n$ such that v_i denotes the type of position i (see figure 9).

$\underline{v}$ alone determines all the carry bits and the following theorem is the basis for the carry-look-ahead method.

Theorem 3.3 *Let the composition $\circ : M^2 \to M$ be given by the table in figure 10. $(M, \circ)$ is a semi group (i.e. $\circ$ is associative). Let $\underline{v}$ be defined as above and let*

$$m_i = (v_{i-1} \circ v_{i-2} \circ v_{i-3} \circ \cdots \circ v_0) \circ C$$

It holds that

$$c_i = \begin{cases} 0 & \text{if } m_i = C \\ 1 & \text{if } m_i = S \end{cases}$$

Proof. Exercise 3.2. □

This means that we have reduced addition to a *prefix computation* in a finite semigroup M: Given $v_1, \ldots, v_n \in M$, compute $\underline{a} = \mathit{prefix}_n(\underline{v})$, where $a_i = \prod_{j=1}^{i} v_j$,

$i = 1, \ldots, n$. In the statement of theorem 3.3 we use suffixes, but it is more usual to talk about a prefix problem (which is clearly computationally equivalent). Note that the prefix problem is not a Boolean problem, and a solution consists in a circuit (using $\circ$-gates) that works for all semigroups M. However, for a concrete semigroup the elements can be represented by bit vectors, and the $\circ$-gates can be represented by small constant size Boolean circuits, giving rise to a Boolean solution.

By using the abstract semigroup formulation of the prefix problem, we get a solution that can be used for other problems besides addition (see exercises).

In the sequel it is assumed that n is a power of 2. $prefix_n(\underline{v})$ is computed by a recursive algorithm:

$\underline{a} \leftarrow prefix(\underline{v})$:

```
  z ← prefix_{n/2}(v_1 ∘ v_2, v_3 ∘ v_4, ..., v_{n-1} ∘ v_n)
  a_1 ← v_1
  for i := 1 to n/2 − 1 do
        a_{2i} ← z_i
        a_{2i+1} ← z_i ∘ v_{2i+1}
  od
  a_n ← z_{n/2}
```

The algorithm may be unfolded into a circuit of size s_n and depth d_n, where

$$\begin{aligned} s_n &= s_{n/2} + n - 1 \\ d_n &= d_{n/2} + 2 \\ s_2 &= 1 \\ d_2 &= 1 \end{aligned}$$

The recurrences imply that $s_n = 2n - \log_2 n - 2$ and $d_n = 2\log_2(n) - 1$, and we get:

Theorem 3.4 *For n a power of 2, $prefix_n$ is computed by a circuit of size $s_n < 2n$ and depth $d_n < 2\log_2 n$.*

Corollary 3.5 *add_n is computed by a circuit of size $O(n)$ and depth $O(\log n)$.*

Proof. The result follows from theorem 3.4 and the construction of exercise 3.3. □

3.2 Addition of many numbers

Let $\underline{x}^1, \ldots \underline{x}^r$ be r bit vectors of length n. Define $addm_{r,n} \in \mathrm{B}_{rn,n+\lceil \log r \rceil}$ by

$$[addm_{r,n}(\underline{x}^1, \ldots, \underline{x}^r)] = \sum_{i=1}^{r} [\underline{x}^i]$$

Corollary 3.5 and a divide and conquer approach results in a circuit of size $O(rn)$ and depth $O(\log r \log n)$, when $n \geq \log r$. The size is optimal, but the depth may be improved by the *carry-save* construction.

Lemma 3.6 *There exists a circuit of size $O(n)$ and depth $O(1)$ that when input $\underline{x}, \underline{y}, \underline{z} \in \{0,1\}^n$ outputs $\underline{u}, \underline{v} \in \{0,1\}^{n+1}$ such that*

$$[\underline{u}] + [\underline{v}] = [\underline{x}] + [\underline{y}] + [\underline{z}]$$

Proof. The basic idea is the following: When adding 3 bits the result is $0, 1, 2$ or 3 which can be represented by 2 bits only. Formally:

$$\begin{aligned} u_i &= x_i \oplus y_i \oplus z_i, 0 \leq i \leq n-1 \\ u_n &= 0 \\ v_0 &= 0 \\ v_{i+1} &= Th_3^2(x_i, y_i, z_i), 0 \leq i \leq n-1 \end{aligned}$$

□

The circuit of the lemma reduces a $addm_{r,n}$-problem to a $addm_{\lceil 2r/3 \rceil, n+1}$-problem. By iterating this construction, we get a problem of type $addm_{2,n'}$ that may be solved using corollary 3.5. Let the size and depth of the final solution be $s_{r,n}$ and $d_{r,n}$. We get the recurrences

$$\begin{aligned} s_{r,n} &= s_{\lceil 2r/3 \rceil, n+1} + O(rn) \\ s_{2,n} &= O(n) \\ d_{r,n} &= d_{\lceil 2r/3 \rceil, n+1} + O(1) \\ d_{2,n} &= O(\log n) \end{aligned}$$

that leads to

Theorem 3.7 *The $addm_{r,n}$-function may be computed by a circuit of size $O(rn)$ and depth $O(\log r + \log n)$.*

3.3 Symmetric functions

Let $f = Eq_n^M$ be a symmetric function. We intend to find an upper bound for the complexity of f. As noted in section 1 the value of a symmetric function is determined by the number of 1-bits in the input, i.e. the value is determined by the function $count_n \in \mathrm{B}_{n,\lfloor \log n \rfloor + 1}$ defined by

$$[count_n(\underline{x})] = |\{i | x_i = 1\}|.$$

$count_n$ is the addition of n 1-bit numbers, and according to theorem 3.7, it is computed by a circuit of size $O(n)$ and depth $O(\log n)$.

Given the binary representation of a number $t \le n$ we still have to construct a circuit that decides whether $t \in M$. Since M can be any subset of $\{0, 1, 2, \ldots, n\}$, we need a general construction for any function $f \in \mathrm{B}_m$, where $m = \lceil \log_2 n \rceil + 1$. By theorem 2.4, any function in B_m can be computed by a circuit of size $\mathrm{O}(2^m/m)$ and depth $\mathrm{O}(m)$.

Theorem 3.8 *All functions in* SYM_n *are computed by circuits of size* $O(n)$ *and depth* $O(\log n)$*.*

Corollary 3.9 *Every function in* SYM_n *has a formula of size* $O(n^{O(1)})$*.*

Binary sorting, $SORT_n \in \mathrm{B}_{n,n}$, is a particularly interesting symmetric problem given by

$$SORT_n(\underline{x}) = (Th_n^n(\underline{x}), Th_n^{n-1}(\underline{x}), \cdots, Th_n^1(\underline{x}))$$

Though theorem 3.8 does not apply ($SORT_n$ has n outputs) it is the case that $SORT_n$ is computed by a circuit of size $O(n)$ and depth $O(\log n)$ (see exercise 3.19), which is optimal.

Since both threshold functions and binary sorting are monotone, we could also ask for their complexities over the monotone basis $\{+, \cdot\}$. For this purpose, we introduce comparator networks, which have traditionally been used for describing solutions to the sorting problem.

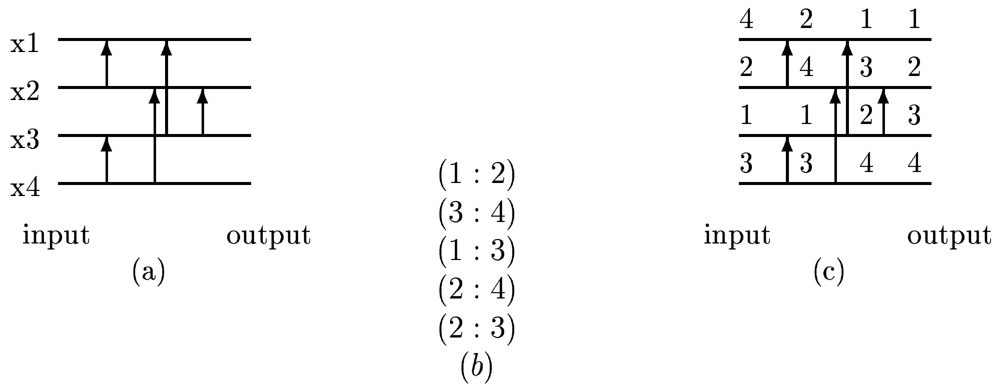

Figure 11: Comparator network computing $SORT_4$

3.4 Comparator Networks

By a comparator network we mean a circuit whose gates are comparators $COMP \in \mathrm{B}_{2,2}$, where

$$COMP(x, y) = (xy, x + y) = (\min(x, y), \max(x, y)).$$

The fan-out of each output from a comparator must be *one*, and we may draw a comparator network as illustrated in figure 11(a) that shows a network with 4 horizontal wires corresponding to 4 inputs (and 4 outputs). A gate is represented by a vertical arrow between two wires. The input comes on the wires to the left of the gate and the output is emitted on the wires to the right with the minimum value at the point of the arrow and the maximum value at the root of the arrow. Figure 11(b) shows the equivalent straight line program, where the notation $(i : j)$ represents an arrow from wire j to wire i.

Definition 3.1 *A* sorting network *with n inputs is a comparator-network that computes $SORT_n$.*

It holds that:

Proposition 3.10 (0-1 principle) *A sorting network can be used to sort elements from an arbitrary ordered set $(S, <)$, when the COMP-gate is replaced by a gate, that computes $(\min(s_1, s_2), \max(s_1, s_2))$ for inputs $s_1, s_2 \in S$.*

Proof. Exercise 3.22. □

The 0-1-principle essentially states that a (straight line) program for sorting over some ordered set by comparison, is correct if and only if it sorts all $(0,1)$ sequences (of the proper length) correctly. Figure 11(c) illustrates sorting of the vector $(4,2,1,3)$.

We introduce the usual complexity measures:

Definition 3.2 *For a function $f_n^m : \mathrm{B}_n \to \mathrm{B}_m$, $S_C(f_n^m)$ and $D_C(f_n^m)$ denotes the smallest size and depth respectively of any comparator network that computes f_n^m.*

Observe that $S_m(f) \leq 2 \cdot S_C(f)$ and $D_m(f) \leq D_C(f)$.

The usual efficient sorting algorithms have time complexity $O(n \log n)$, and we could reasonably expect the same bound to hold for sorting networks. In fact it does, but the result is highly nontrivial. None of the classical sorting algorithms with time complexity $O(n \log n)$ leads to straight line programs. As an example, consider insertion sort (using binary search). It leads to a program that branches dependent on the result of earlier comparisons.

The existence of a sorting network of size $\mathrm{O}(n \log n)$ and depth $\mathrm{O}(\log n)$ was proven by Ajtai, Komlôs and Szemeredi (1983). The constants hidden in the $O(\cdot)$-notation are astronomic, so the result is of theoretical interest only. Paterson (1990) later reduced the constants to be "just" enormous.

We will here present a relatively simple network of size $O(n \log^2 n)$ and depth $O(\log^2 n)$.

3.4.1 Merging Networks

The basic idea underlying merge sort is very simple. When given n elements (for technical simplicity, we will assume that n is a power of 2), we divide the inputs in two groups, which are sorted recursively (in parallel), and the final result is computed by a merging network:

Definition 3.3 *A* merging network *is a comparator network that computes $SORT_n$ when the inputs $(\underline{x}, \underline{y})$ are both sorted and $|\underline{x}| = |\underline{y}| = \frac{n}{2}$.*

Lemma 3.11 (Batcher) *There exists a merging network of size $\frac{n}{2} \log \frac{n}{2} + 1$ and depth $\log n$.*

Proof. For the description of the network we use the notation:

$$\underline{v}_{odd} = (v_1, v_3, v_5, \ldots, v_{2k-1}) \quad \text{and} \quad \underline{v}_{even} = (v_2, v_4, v_6, \ldots, v_{2k}),$$

for $\underline{v} = (v_1, v_2, v_3, \ldots, v_{2k})$.

The network is based on the following algorithm:

merge($\underline{x}, \underline{y}$)
  { INVARIANT: $\underline{x}, \underline{y}$ are both sorted and $|\underline{x}|, |\underline{y}| = \frac{n}{2}$}
if $n = 2$ **then return** $COMP(x_1, y_1)$;
else
  do in parallel
    $\underline{z}_{odd}$:= merge($\underline{x}_{odd}, \underline{y}_{odd}$);
    $\underline{z}_{even}$:= merge($\underline{x}_{even}, \underline{y}_{even}$);
  od;
  for $i := 1$ **to** $\frac{n}{2} - 1$ **do in parallel**
    $(\underline{w}_{2i}, \underline{w}_{2i+1}) := COMP(\underline{z}_{2i}, \underline{z}_{2i+1})$;
  od;
  $w_1 := z_1$; $w_n := z_n$;
  {INVARIANT: $\underline{w} = SORT_n(\underline{x}, \underline{y})$};
  return $\underline{w}$;
fi;
end

We need to prove that the algorithm performs a merge correctly. We may assume that $\underline{x} = 0^{2i+\delta}1^{\frac{n}{2}-2i-\delta}$ and $\underline{y} = 0^{2j+\epsilon}1^{\frac{n}{2}-2j-\epsilon}$, where $i, j \in \{0, 1, \ldots, \frac{n}{2}\}$ and $\delta, \epsilon \in \{0, 1\}$. Let $\gamma = \delta + \epsilon \in \{0, 1, 2\}$.

The result of the recursive calls to merge is $\underline{z} = 0^{2(i+j)+\gamma}1^{n-2(i+j)-\gamma}$ if $\gamma \in \{0, 1\}$, and the result is $\underline{z} = 0^{2(i+j)+1}101^{n-2(i+j)-3}$ if $\gamma = 2$.

The sorted vector $\underline{w}$ is therefore computed from $\underline{z}$ by possibly swapping the values at positions $z_{2(i+j+1)}$ and $z_{2(i+j+1)+1}$.

All branching and loops in the algorithm depend on n only, so the algorithm may be unfolded into a straight line program (merging network) for any given n.

The size of the network satisfies $S(n) = 2S(\frac{n}{2}) + \frac{n}{2} - 1$ and $S(2) = 1$, giving $S(n) = \frac{n}{2} \log \frac{n}{2} + 1$. The depth satisfies $D(n) = D(\frac{n}{2}) + 1$ and $D(2) = 1$, giving $D(n) = \log n$. □

Batcher's construction is usually named odd-even-merge and it leads to

Theorem 3.12 *It is possible to construct a sorting network of size* $\frac{n}{4}\log^2 n - \frac{n}{4}\log n + n - 1$ *and depth* $\frac{1}{2}\log^2 n + \frac{1}{2}\log n$.

Proof. Merge sort based on odd-even-merge leads to recurrences for the size of a sorting network: $S(n) \leq 2S(\frac{n}{2}) + \frac{n}{2}\log\frac{n}{2} + 1$ and $S(2) = 1$. Similarly for depth: $D(n) \leq D(\frac{n}{2}) + \log n$ and $D(2) = 1$. The theorem follows from solving the recurrences. □

Note that the sorting network in figure 11 is constructed using odd-even-merge.

It is possible to merge n elements using only $n-1$ comparisons, when branching is allowed, but for straight line programs odd-even-merge is essentially optimal. Let $M(n)$ denote the minimum size of a merging network with n inputs. Miltersen, Paterson and Tarui (1992) have shown that $M(n) \geq \frac{n}{2}\log\frac{n}{2} - O(n)$. We present a slightly weaker result with a simpler proof:

Theorem 3.13 (Floyd) $M(n) \geq \frac{n}{4}\log 2n$.

Proof. Let C be an optimal merging network. Using the result of exercise 3.23, we may assume that C takes the two sorted input vectors $\underline{x}$ and $\underline{y}$ in the following fixed permutation: $\underline{z} = (x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_{\frac{n}{2}}, y_{\frac{n}{2}})$, i.e. $\underline{z}_{odd} = \underline{x}$ and $\underline{z}_{even} = \underline{y}$.

We divide the comparators in C in three groups:

$$C_1 = \{(i:j) \mid 1 \leq i, j \leq \frac{n}{2}\}$$

$$C_2 = \{(i:j) \mid \frac{n}{2} < i, j \leq n\}$$

$$C_3 = \{(i:j) \mid i \leq \frac{n}{2} < j \quad \text{or} \quad j \leq \frac{n}{2} < i\}$$

Consider all possible inputs $\underline{x}, \underline{y} \in \{0,1\}^{\frac{n}{4}} 1^{\frac{n}{4}}$. On such inputs no comparator in C_2 or C_3 makes any swapping, so all the work is done by comparators in C_1, which therefore must form a complete merging network on $\frac{n}{2}$ input, i.e. $|C_1| \geq M(\frac{n}{2})$. By a symmetric argument we have that $|C_2| \geq M(\frac{n}{2})$.

A correct merge of $\underline{x} = 0^{\frac{n}{2}}$ and $\underline{y} = 1^{\frac{n}{2}}$ requires that $|C_3| \geq \frac{n}{4}$.

In total, we have shown the recurrence $M(n) \geq 2M(\frac{n}{2}) + \frac{n}{4}$ and $M(2) = 1$, which implies the theorem. □

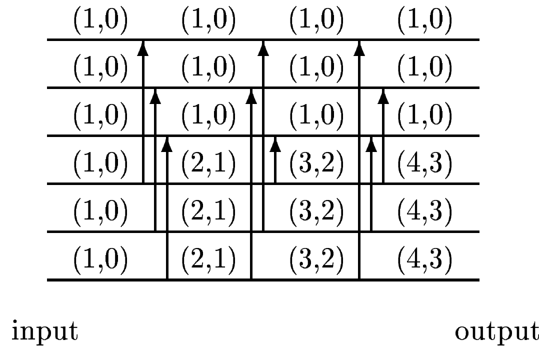

Figure 12: (l, m)-values for splitting network $(n = 6)$

3.4.2 Splitting Networks

One may sort by repeated merging, but it is also possible to sort by repeated splitting. One starts by dividing the input into two groups containing the $\frac{n}{2}$ smaller elements and the $\frac{n}{2}$ larger elements, respectively. The two groups are sorted recursively (in parallel).

Definition 3.4 *A* splitting network *is a comparator network that when input* $\underline{z} \in \{0,1\}^n$ *computes a permutation* $(\underline{x}, \underline{y})$ *of* $\underline{z}$*, such that* $|\underline{x}| = |\underline{y}| = \frac{n}{2}$ *and* $x_i \leq y_j$ *for all* i, j*. (The output from a splitting network is not uniquely determined).*

By computing the median, we can construct a *non* straight line program of size $O(n)$ that splits the input, but for straight line programs, Jimbo and Maruoka (ca. 1993) have the so far best construction of size $< 1.9n \log n$ and depth $O(\log n)$. This is optimal within a constant factor, since:

Theorem 3.14 *A splitting network with* n *inputs has size at least*

$$\frac{n}{2}\lceil \log(\frac{n}{2} + 1) \rceil.$$

Proof. Given a splitting network, we may label each wire by a pair of numbers (l, m), as follows (figure 12 shows an example where $n = 6$):

The l-value of a wire is the minimum possible number of 0's in an input vector that can make the wire transport the value 0.

For input wires the l-values are all 1, and for a correct splitting network the lower $\frac{n}{2}$ output wires have $l \geq \frac{n}{2} + 1$.

If the l-values of the wires into a given gate $(i:j)$ are l_i and l_j, respectively, and the l-values out of the gate are l'_i and l'_j, respectively, then

$$l'_i = \min(l_i, l_j)$$

and

$$l'_j \leq l_i + l_j.$$

The last inequality is proven in exercise 3.24.

The m-value for a wire is defined inductively: The input value of an input wire is 0. If the m-values before a gate $(i:j)$ are m_i and m_j, respectively, then the m-values after the gate are $m'_i = \min(m_i, m_j)$ and $m'_j = \max(m_i, m_j) + 1$, respectively.

Note that the sum of the m-values on the output wires is equal to the number of gates in the network.

By induction one may prove the following connection between l-values and m-values on any single wire:

$$l \leq 2^m.$$

For a correct splitting network, we have: At least $\frac{n}{2}$ of all l-values on output wires must be $\geq \frac{n}{2}+1$, and therefore at least $\frac{n}{2}$ of the m-values on output wires must be $\geq \lceil \log(\frac{n}{2}+1) \rceil$, which implies that the network has size at least $\frac{n}{2} \lceil \log(\frac{n}{2}+1) \rceil$. □

3.4.3 Th_n^k for fixed k

A comparator network for $SORT_n$ includes a comparator network for Th_n^k for all k. A better construction is possible for fixed k:

Theorem 3.15 *Assume that n and k are powers of two. For fixed k it holds that* $S_C(Th_n^k) \leq n(\log k + 1) + o(n)$

Proof. The first published proof is due to Yao (1980). Compared to this proof, we are going to use a less efficient construction and be rather careless in our analysis, but that has only implications for the size of the $o(n)$-term.

We will solve a more general problem by constructing a comparator network that computes $SORT_n^k(\underline{x}) = (Th_n^k(\underline{x}), Th_n^{k-1}(\underline{x}), \cdots, Th_n^1(\underline{x}))$.

The construction is described by a recursive algorithm:

```
function SORT_n^k(x)
if k = 1 then return the largest element in x.
if k = n then return SORT_n(x).
if 2 ≤ k ≤ n/2 then
  for i ∈ {1, 2, ..., n/2} do (y_i, y_{i+n/2}) := COMP(x_i, x_{i+n/2})
  return MERGE^k_{k/2,k}(SORT^{k/2}_{n/2}(y_1, y_2, ···, y_{n/2}),
     SORT^k_{n/2}(y_{n/2+1}, y_{n/2+2}, ···, y_n)).
end
```

For $k = 1$, we have that $S_C(SORT_n^1) = n - 1$, since the result depends on all the inputs.

For $k = n$, we use a sorting network, and by theorem 3.12 we have $S_C(SORT_n^n) \leq n \log^2 n$.

For $1 < k < n$, we start by making $\frac{n}{2}$ comparisons. As a result at least $\frac{k}{2}$ of the k largest elements are placed in $(y_{n/2+1}, y_{n/2+2}, \ldots, y_n)$. Hence, it suffices to merge the result of the two recursive calls that are specified in the algorithm. $MERGE_{n,m}^k$ finds the k largest elements in two sorted input lists of length n and m. It holds that $S_C(MERGE_{k/2,k}^k) \leq k \log k$ (see exercise 3.27).

In total we get a recurrence for $s(n, k) = S_C(SORT_n^k)$

$$s(n, 1) = n - 1$$

$$s(n, n) \leq n \log^2 n$$

$$s(n, k) \leq \frac{n}{2} + s(\frac{n}{2}, \frac{k}{2}) + s(\frac{n}{2}, k) + k \log k \quad \text{for} \quad 2 \leq k \leq \frac{n}{2}$$

We solve it by guessing a solution

$$s(n, k) \leq n(\log k + 1) + k \log^2 k \binom{\log n}{\log k}$$

and verify the correctness of this guess by induction: The induction basis, i.e. the values $s(n,1)$ and $s(n,n)$ are obviously correct. For the induction step we need a calculation:

$$s(n,k) \leq$$

$$\frac{n}{2} + s(\frac{n}{2},\frac{k}{2}) + s(\frac{n}{2},k) + k\log k \leq$$

$$\frac{n}{2} + \frac{n}{2}(\log k) + \frac{k}{2}(\log k - 1)^2 \binom{\log n - 1}{\log k - 1} +$$

$$\frac{n}{2}(\log k + 1) + k\log^2 k \binom{\log n - 1}{\log k} + k\log k \leq$$

$$n(\log k + 1) + k\log^2 k \binom{\log n}{\log k}$$

Since $\binom{\log n}{\log k}$ is vanishingly small compared to n (for fixed k !) we have proven the theorem. □

The lower bound technique for splitting networks can also be used to show the optimality of the construction in theorem 3.15 (up to a lower order term):

Theorem 3.16 $S_C(Th_n^k) \geq (n-k)\lceil \log k + 1 \rceil$.

Proof. Exercise 3.28 □

3.5 Threshold Functions

From sorting networks of depth $O(\log n)$ it is possible to construct polynomial size formulae (over the monotone basis $\{+,\cdot\}$) for threshold functions, but the huge constants hidden under the "O" leads to an equally huge exponent.

Valiant (1984) has proven the existence of much smaller formulae by a probabilistic argument. In this section, we present a simplified (and slighter weaker) version of his result.

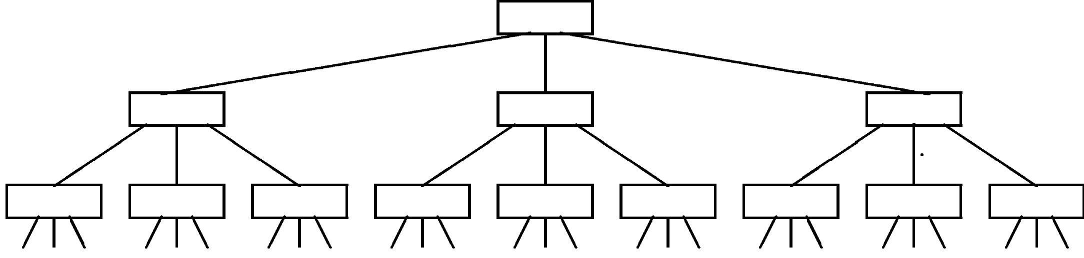

Figure 13: A Th_3^2 formula in A_3

We know that $\{Th_3^2\}$ is a complete basis for MON. We are going to show that the majority function $MAJ = \{Th_{2m+1}^{m+1}\}$ has a small formula over this basis, which implies the existence of small $\{+,\cdot\}$-formulae for MAJ_n and Th_n^k for all n, k.

Let $n = 2m+1$ be fixed in the following.

Let A_d be the set of all totally balanced formulae of depth d over Th_3^2 with input from $\{x_1, x_2, \cdots, x_n\}$. Assume that F is a random formula in A_d and let $\underline{c} \in \{0,1\}^n$ be fixed. Let

$$f_d = \Pr(F(\underline{c}) \neq MAJ(\underline{c})).$$

We have a recurrence for f_d (F_1, F_2 and F_3 are random formulae in A_d):

$$f_{d+1} = \Pr(Th_3^2(F_1(\underline{c}), F_2(\underline{c}), F_3(\underline{c})) \neq MAJ(\underline{c})) =$$
$$f_d^3 + 3f_d^2(1-f_d) = 3f_d^2 - 2f_d^3$$

and

$$f_0 = \Pr(c_i \neq MAJ(\underline{c})) \leq \frac{m}{2m+1} = \frac{1}{2} - \frac{1}{2n}$$

The graph of $3x^2 - 2x^3$ is drawn in figure 14. It shows that $f_d \to 0$ for $d \to \infty$, i.e. for sufficiently large d we have that $F = MAJ_n$ with high probability and therefore A_d contains a formula that computes MAJ_n.

Being more precise, for $f_d < 2^{-n}$ we have that

$$\Pr(F = MAJ_n) \geq 1 - \sum_{\underline{c}} \Pr(F(\underline{c}) \neq MAJ_n(\underline{c})) > 1 - 2^n 2^{-n} = 0$$

In the sequel we are going to analyse how large d must be to ensure that $f_d < 2^{-n}$.

Note that $3x^2 - 2x^3$ has fixed points at $0, \frac{1}{2}$ and 1.

If $f_d = \frac{1}{2} - \epsilon$ then $f_{d+1} = 3(\frac{1}{2} - \epsilon)^2 - 2(\frac{1}{2} - \epsilon)^3 = \frac{1}{2} - \frac{3}{2}\epsilon + \mathrm{O}(\epsilon^2)$. This implies that for all $0 < \gamma < \frac{3}{2}$ there exists an $\epsilon_0 > 0$ such that

$$f_d \leq \tfrac{1}{2} - \epsilon \Rightarrow f_{d+1} < \tfrac{1}{2} - \gamma\epsilon$$

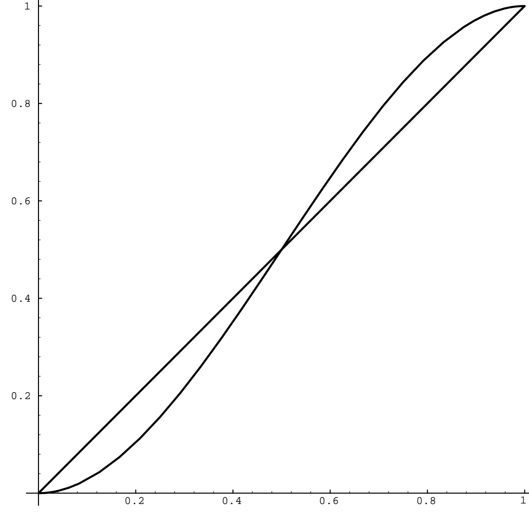

Figure 14: The function $3x^2 - 2x^3$

for $0 < \epsilon < \epsilon_0$. This means that $f_d < \frac{1}{2} - \gamma^d \frac{1}{2n}$ provided $\gamma^d \frac{1}{2n} < \epsilon_0$ or $d < \log_\gamma \epsilon_0 + \log_\gamma 2n$, which implies that $f_{d_0(n)} < \frac{1}{2} - \epsilon_0$ after $d_0(n) = \log_\gamma n + \mathrm{O}(1)$ steps. Given an arbitrary constant $0 < \alpha < \frac{1}{2} - \epsilon_0$, we find that $f_{d_0(n)+d_1} < \alpha$ after an additional $d_1 = \mathrm{O}(1)$ steps, since $\frac{1}{2} - \epsilon_0$ is a constant (independent of n). α will be fixed later.

After an additional j steps, we have that $f_{d_0(n)+d_1+j} < 3^{2^j} \alpha^{2^j}$, since $f_{d+1} < 3f_d^2$. We want to obtain $f_{d_0(n)+d_1+j} < 2^{-n}$. This holds if $(3\alpha)^{2^j} < 2^{-n}$ or if $2^j \log_2(3\alpha) < -n$, which requires that $\log_2(3\alpha) < 0$ or $3\alpha < 1$. Therefore, let $\alpha = \frac{1}{6}$. For such choice of α we find that $f_{d_0(n)+d_1+j} < 2^{-n}$, when $j > \log_2 n$.

Combined we see that $\log_\gamma n + \log_2 n + \mathrm{O}(1) = \mathrm{O}(\log n)$ iterations suffice to reduce f to 2^{-n}. This implies that there exists a Th_3^2-formula of depth $\mathrm{O}(\log n)$ and size $n^{O(1)}$ that computes MAJ_n.

When substituting $\gamma = \frac{3}{2}$, we find that the depth is bounded by $2.71 \log_2 n$ and the size is bounded by $\mathrm{O}(3^{2.71 \log_2 n}) = \mathrm{O}(n^{4.49})$.

One may transform this result to the basis $\{+, \cdot\}$ and prove the existence of a $\{+, \cdot\}$-formula of size $\mathrm{O}(n^{8.13})$ for MAJ_n.

Valiant (1984) builds a proof on the basis $(x+y)(z+v) \in \mathrm{B}_4$ rather than Th_3^2. The analysis is similar to ours, but more complicated. In return, the result is stronger. Valiant proves the existence of a $\{+,\cdot\}$-formula of size $\mathrm{O}(n^{5.3})$ for MAJ_n. Boppana (1989) has shown that this result is the best possible for such proof technique.

Exercises

Exercise 3.1 *Improve the construction of the $addc_n$-circuit using dynamic programming to get a circuit of size $O(n \log n)$ and depth $O(\log n)$.*

Exercise 3.2 *Prove theorem 3.3 using the observations stated prior to the theorem.*

Exercise 3.3 *Describe a Boolean representation of the semigroup from theorem 3.3 and show how a solution of the prefix computation problem leads to a solution of the Boolean add_n-problem.*

Exercise 3.4 *Prove the correctness of the prefix computation algorithm.*

Exercise 3.5 *Draw the prefix computation circuit for $n = 16$.*

Exercise 3.6 *Due to the general applicability of the prefix circuit, we are interested in reducing the size of the constant factors occurring in the complexity bound. A possible trade-off between the size constant and the depth constant would also be interesting. Can you reduce the depth constant (while increasing the size constant)?*

Exercise 3.7 *Show that the function $less_n \in \mathrm{B}_{2n}$ defined by*

$$less_n(\underline{x}, \underline{y}) = \begin{cases} 1 & \text{if } [\underline{x}] \leq [\underline{y}] \\ 0 & \text{if } [\underline{x}] > [\underline{y}] \end{cases}, \quad \underline{x}, \underline{y} \in \{0,1\}^n$$

is computed by a circuit of size $O(n)$ and depth $O(\log n)$.

Exercise 3.8 *Show that the function $monus_n \in \mathrm{B}_{2n,n}$ defined by*

$$[monus_n(\underline{x},\underline{y})] = \begin{cases} [\underline{x}] - [\underline{y}] & \text{if } [\underline{x}] \geq [\underline{y}] \\ 0 & \text{if } [\underline{x}] < [\underline{y}] \end{cases}, \quad \underline{x},\underline{y} \in \{0,1\}^n$$

is computed by a circuit of size $O(n)$ and depth $O(\log n)$.

Exercise 3.9 *Show that addition is in AC^0. How much can you reduce the size?*

Exercise 3.10 *The prefix circuit requires that the operation is associative. A different construction avoids this condition. Let $\Box$ be some (not necessarily associative) binary operator on a finite set M, and assume that the elements of M have some fixed Boolean representation.*

1. *Show the existence of a Boolean circuit of size $O(n)$ and depth $O(\log n)$ that when input $x_1, x_2, \ldots x_n \in M$ computes $x_1, x_1 \Box x_2, (x_1 \Box x_2) \Box x_3, \ldots$. Hint: The relevant semigroup is $\mathcal{M} = \{\tau : M \to M\}$ with composition of functions as the semigroup operation.*
2. *Let L be a regular language over a finite alphabet Σ (so the strings over Σ have a natural Boolean representation). Show the existence of a family of circuits of size $O(n)$ and depth $O(\log n)$ that decides membership of L. (This implies that all regular languages are in NC^1.)*

Exercise 3.11 *Let $f \in \mathrm{B}_n$ be the function*

$$f(x_1, x_2, \ldots, x_n) = x_1(x_2 + (x_3(x_4 + (x_5(\ldots x_n \ldots)))))$$

Show that f is computed by a circuit of size $O(n)$ and depth $O(\log n)$.

Exercise 3.12 *Show that the function $\{brackets_{2n}\}$ (see exercise 2.10) is computed by a circuit of depth $O(\log n)$ and size $O(n \log n)$.*

Exercise 3.13 *Show using a simple divide and conquer approach that $addm_{r,n}$ is computed by a circuit of size $O(rn)$ and depth $O(\log r \log n)$ when $n \geq \log r$.*

The *multiplication function $mult_n$* $\in \mathrm{B}_{2n,2n}$ is defined by

$$[mult_n(\underline{x},\underline{y})] = [\underline{x}][\underline{y}], \ \underline{x},\underline{y} \in \{0,1\}^n.$$

Exercise 3.14 *The "school" method for multiplication of two numbers x and y uses the identity*

$$xy = \sum_{i=0}^{n-1} x_i y 2^i \quad \textit{(assuming that } x = \textstyle\sum_{i=0}^{n-1} x_i 2^i\textit{)}.$$

Show using $addm_{r,n}$ that $mult_n$ is computed by a circuit of size $O(n^2)$ and depth $O(\log n)$.

Exercise 3.15 *Show using a divide and conquer approach that $mult_n$ is computed by a circuit of size $O(n^{\log_2 3})$ and depth $O(\log^2 n)$. Hint: Given the numbers a, b, c and d one may compute bd, ac and $ad + bc$ using only 3 multiplications.*

The techniques of exercises 3.14 and 3.15 are asymptotically inferior to a technique using fast Fourier Transform:

Theorem 3.17 *$mult_n$ is computed by a circuit of size $O(n \cdot \log n \cdot \log\log n)$ and depth $O(\log n)$.*

Proof. Omitted □

It is currently unresolved whether $mult_n$ is computed by a circuit of size $O(n)$.

Exercise 3.16 *The positional notation for t, $(0 \leq t \leq n)$, $p_n(t)$ is the binary string $(x_n, \ldots, x_0)$, where*

$$x_i = \begin{cases} 0 & \textit{if } i \neq t \\ 1 & \textit{if } i = t \end{cases}$$

Let $bp_m \in \mathrm{B}_{m,2^m}$ where $bp_m(\underline{x}) = p_{2^m}([\underline{x}])$ be the function that converts a number from binary to positional notation. Show that bp_n is computed by a circuit of size $\mathrm{O}(2^n)$ and depth $\mathrm{O}(\log n)$

Exercise 3.17 *Let t be a natural number $0 \leq t \leq n$. The unary notation for t, $u_n(t)$, is the binary string $(y_n, \ldots, y_1)$, where*

$$y_i = \begin{cases} 0 & \textit{if } t < i \\ 1 & \textit{if } t \geq i \end{cases}$$

Let $pu_n \in \mathrm{B}_{n,n-1}$ be the function that converts a number from positional to unary notation. Show that pu_n is computed by a circuit of size $O(n)$ and depth $O(\log n)$.

Exercise 3.18 *Let* $pb_{2^m} \in \mathrm{B}_{2^m,m}$ *be the function that converts a number from positional notation to binary notation. Show that* pb_{2^m} *is computed by a circuit of size* $O(2^m)$ *and depth* $O(m)$.

Exercise 3.19 *Show that* $SORT_n$ *is computed by a circuit of size* $O(n)$ *and depth* $O(\log n)$.

Why is this result not inconsistent with the lower bound $\Omega(n \log n)$ *on the sequential time for sorting (proven in the course dAlg)?*

Exercise 3.20 *Show by a single argument that the circuits from theorem 3.4, corollary 3.5, theorem 3.7, exercise 3.16, exercise 3.17, exercise 3.18 and exercise 3.19 all are size and depth optimal (up to a constant factor).*

Exercise 3.21 *Show that the existence of a (comparator based) sorting network of size* s *and depth* d *implies the existence of a (*$\{+,\cdot\}$*-based) circuit for binary sorting of size* $2s$ *and depth* d.

Exercise 3.22 *Show proposition 3.10*

Exercise 3.23 *Show that if there exists a merging network* C *that sorts the input vector* $(x_1, x_2, \ldots, x_{\frac{n}{2}}, y_1, y_2, \ldots, y_{\frac{n}{2}})$ *provided that each of* $\underline{x}$ *and* $\underline{y}$ *are sorted in advance, then there also exists a network* C' *that sorts the input vector* $(x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_{\frac{n}{2}}, y_{\frac{n}{2}})$ *provided that each of* $\underline{x}$ *and* $\underline{y}$ *are sorted in advance, and* C' *has the same size and depth as* C.

Exercise 3.24 *Show that* $l'_j \leq l_i + l_j$ *as asserted in the proof of theorem 3.14.*

Exercise 3.25 *Show that the existence of a sorting network* C *implies the existence of a sorting network* C' *where all gates* $(i : j)$ *satisfy that* $i < j$, *and* C' *has the same size and depth as* C.

Exercise 3.26 *Show that a splitting network can be made from only* $\frac{n}{2}$ *gates, if the input is known to be in the form* $(\underline{x}, \underline{y})$, *where* $\underline{x}$ *and* $\underline{y}$ *are both sorted.*

Exercise 3.27 *A sequence* $x_1, x_2, \ldots, x_n$ *is bitonic if there exists* j *such that* $x_1 \geq x_2 \geq x_3 \geq \cdots \geq x_j$ *and* $x_j \leq x_{j+1} \leq \cdots \leq x_n$.

1. *Construct a comparator network of size* $\frac{n}{2}\log n$ *and depth* $\log n$ *that computes* $SORT_n$ *assuming that the input* $\underline{x}$ *is a bitonic sequence. (Hint:* $\underline{x}_{odd}$ *and* $\underline{x}_{even}$ *are also bitonic).*

2. *Show the bound* $S_C(MERGE^k_{k/2,k}) \leq k \log k$ *that was used in the proof of theorem 3.15.*

Exercise 3.28 *Prove theorem 3.16.*

Exercise 3.29 *For fixed* k*, find a bound for* $S_m(Th^k_n)$*.*

Exercise 3.30 *Let* neg_n *be the* n*-fold negation function defined by*

$$neg_n(x_1, \ldots, x_n) = (\bar{x}_1, \ldots, \bar{x}_n)$$

The object of this exercise is to show that neg_n *can be implemented over the basis* $\{+, \cdot, ^-\}$ *using only* $\lfloor \log n \rfloor + 1$ *negation gates.*

1. *Appreciate for a moment how surprising this result really is.*

2. *Construct the circuit from the following components:*

   (a) *Binary sorting without the use of negation (exercise 3.21).*

   (b) *Unary to binary conversion of the sorted bits, while computing the inverse binary representation as well.*

   (c) *Binary to positional conversion of the result.*

   (d) *Binary sorting of the* n *vectors*

   $$(x_2, x_3, \ldots, x_n), (x_1, x_3, x_4, \ldots, x_n), \ldots, (x_1, x_2, \ldots, x_{n-1}).$$

   (e) *Combination of the results from pt. 2c and pt. 2d.*

3. *Find the size and depth of the circuit (in terms of* n*).*

4. *Draw the circuit for* $n = 3$*.*

5. *Show that the result implies that* every *function on* n *variables can be computed using only* $\lfloor \log n \rfloor + 1$ *negation gates.*

Exercise 3.31 *The halting problem K takes for input the name of a Turing machine T with an input x and reports whether T stops on x. It is well known that K is not solvable by a Turing machine. Fortunately, in a small village near Copenhagen, there lives an old man that can solve instances of K using divine inspiration. You have three instances of the Halting problem for which you want an answer. You can only afford to pay the old man for two answers. What can you do ?*

Literature

1. Ajtaj, M., Komlós, J. and Szemerédi, E. (1983) An $O(n \log n)$ Sorting Network. *Combinatorica* **3**, 1–19.

2. Boppana, R. (1989) Amplification of Probabilistic Boolean Formulas. In *Advances in Computing Research 5: Randomness and Computation,* Micali, S., ed., JAI Press, 27–45.

3. Jimbo, S. and Maruoka, A. (ca. 1993), A Method of Constructing Selection Networks with $O(\log n)$ Depth. *SIAM J. Comput.* **25** (1996) 709–739.

4. Miltersen P. B., Paterson M. and Tarui, J (1992), The Asymptotic Complexity of Merging Networks. *Proc.* 33*rd Ann. IEEE Symp. Foundations of Computer Science,* 236–246.

5. Paterson, M. (1990) Improved Sorting Networks with $O(\log N)$ Depth. *Algorithmica* **5**.

6. Valiant, L. G. (1984) Short Monotone Formulae for the Majority Function. *Journal of Algorithms* **5**, 363–366.

7. Yao, A. C. (1980) Bounds on Selection Networks. *SIAM Journal on Computing* **9**, 566–582.

4 Machine Models and Boolean Circuits

In the first chapters, we presented a model of computation based on Boolean circuits. From other courses you should be familiar with abstract machines such as the Turing Machine (TM) and the Random Access Machine (RAM). One might expect a close relationship between complexity measures defined in terms of machines and circuits, respectively, since the hardware of existing machines are based on Boolean circuits and it is possible to simulate a Boolean circuit with software. Indeed, we shall see that sequential time correspond to circuit size, and parallel time correspond to circuit depth. More surprisingly, space is related to depth as well.

To establish the above correspondences, we introduce a nonuniform machine model. First of all we remind the reader of Turing Machine complexity measures.

4.1 Turing Machines

The Turing Machine consists of a finite control unit and an infinite tape with a tape head (for technical simplicity we use a machine with one tape only).

The control unit can be in any state taken from a finite set Q.

The non blank portion of the tape is finite and is described as a string over the alphabet $\Gamma = \{0, 1, \#\}$. The Turing Machine can read and write to the tape cell currently below the tape head.

A configuration is a tuple (q, t, h), where q is a state, t is the contents of the tape, and h is the position of the tape head. The dynamics of the Turing machine is described by a partial transition function $\delta : Q \times \Gamma \to Q \times (\{L, R\} \cup \Gamma)$. If in the current configuration the Turing Machine is in state q and the character α is below the tape head, then $\delta(q, \alpha)$ decides the next possible configuration of the machine. The first component specifies the state of the finite control. The second component either specifies movement of the tape head a single cell to the left (L) or right (R) or specifies a new symbol ($\gamma \in \Gamma$) to be written under the tape head, which remains on the spot.

The are three special states, the initial state, the accept state and the reject state. δ is undefined in the accept or reject states.

Let $L \subseteq \{0, 1\}^*$ be a language. We say that M recognises L, if the machine reaches an accept state when started in on an input $x \in L$ and halts in the reject

state, when started on an input $y \notin L$. The machine is started on an input $\underline{x} \in \{0,1\}^n$ by letting the tape initially contain $\#x_1x_2\dots x_n\#\#\#\dots$, with the tape head positioned over the blank symbol immediately to the left of x_1. The control unit is placed in the initial state.

A nondeterministic Turing Machine is defined similarly to a deterministic one, apart from allowing δ to be a relation. The machine accepts an input x, if there exists a possible computation that accepts x.

Measuring the time usage of a Turing Machine is fairly straight forward. Measuring the space usage is more subtle. It seems natural to count the number of tape cells that the Turing Machines touches, but since it has to read all of the input, it would necessarily use linear space with this measure.

Instead we let the Turing Machine have two tapes.

- A read-only tape, where the input is placed. The machine is not allowed to write on this tape and the tape head can not leave the part of the tape, where input is written.
- A work-tape for general use.

The two heads move independently, as specified by the control unit. The space usage on such an *off-line* Turing Machine is measured by the number of cells on the work tape that are scanned by the work tape head during a computation.

Definition 4.1 P *(*NP*) is the class of languages that can be recognised by a (nondeterministic) Turing Machine using polynomial time.*

L *(*NL*) is the class of languages that can be recognised by a (nondeterministic) Turing Machine using logarithmic space.*

4.2 Uniformity and nonuniformity

Usual machine models are *uniform* in that a single finite program must work for all input sizes. As a consequence some problems have no programs. They are undecidable. In contrast every problem (language $L \in \{0,1\}^*$) can be solved (decided) by a *nonuniform* family of Boolean circuits.

It is even possible to construct an undecidable problem with a trivial nonuniform solution: Let H be an arbitrary undecidable language, and let $s_0, s_1, s_2, \dots$ be

an effective enumeration of $\{0,1\}^*$. Let H^* be defined by $x \in H^*$ if and only if $s_{|x|} \in H$. Since H reduces to H^* (by an effective though not polynomial time reduction), H^* is undecidable. For every n, H^* is constant on $\{0,1\}^n$ such that $S(H^*_n)$ and $D(H^*_n)$ are both $O(1)$.

The example shows us that if we want to establish a tight relationship between circuit based and machine based complexity measures, we must either make a uniform version of the circuits or a nonuniform version of the machines. Both approaches have been taken in the literature. We choose a variant of the latter.

Definition 4.2 *A nonuniform Turing Machine is a family of Turing Machines $\{T_n\}$, one for each input size such that the number of states in T_n is $n^{O(1)}$.*

P/poly *(*NP/poly*) is the class of languages that can be recognised by a nonuniform (and nondeterministic) Turing Machine using polynomial time.*

L/poly *(*NL/poly*) is the class of languages that can be recognised by a nonuniform (and nondeterministic) Turing Machine using logarithmic space.*

In general all complexity classes comes in both a uniform version and a nonuniform version. However, for technical simplicity, we phrase the relations between machine complexity measures and circuit complexity measures in the coming sections in terms of polynomial time and logarithmic space only.

There is quite some disagreement between complexity theorists whether the uniform models or the nonuniform models are the more fundamental. Protagonists of nonuniformity argue that the simplicity of the circuit model will make it easier to develop new techniques for proving nontrivial lower bounds. As we shall see in later sections, their has indeed appeared new techniques for proving lower bounds in nonuniform models in recent year. One should keep in mind, though, that diagonalisation, a successful technique for separating complexity classes, works only for uniform models.

4.3 Time and Size

In this section we shall establish that a machine of polynomial time complexity corresponds to a polynomial size circuit family, speaking nonuniformly.

We use the Turing Machine. Corresponding results for the Random Access Machine is contained in exercises 4.6 and 4.7.

We identify a language L with the family of characteristic functions $\{L_n \in \mathrm{B}_n\}$, where $L_n(x) = 1$ if and only if $x \in L$. We can ask for the size and depth of L.

Theorem 4.1 *Let T be a deterministic Turing Machine that recognises a language L. If T stops within $t(n)$ steps on any input of size n then $S(L_n) = O(t(n)^2)$.*

The theorem was first shown by Savage (1972) and later improved by Pippenger and Fischer (1979) to $S(L_n) = O(t(n) \log t(n))$.

Proof. We start by giving a suitable Boolean encoding of a T-configuration. The state of the control unit is represented by a sufficiently large bit vector $\underline{q}$. Since T halts within $t(n)$ steps, it can touch no more than $t(n)$ cells to the right of the initial head position. Hence, it suffices to represent the $t(n) + 1$ first tape cells. Let the contents of these be a string τ, and represent τ by the bit vector $\underline{t} = \gamma(\tau)$, where γ is defined by $\gamma(0) = (0, 0)$, $\gamma(1) = (1, 1)$, $\gamma(\#) = (0, 1)$ and γ is extended to strings in the natural manner (by homomorphism). The position of the head is represented by a vector $\underline{a}$ holding the binary representation of a number between 1 and $t(n) + 1$. Combined the three vectors represent a unique configuration. Let $< k >$ denote the representation of configuration k. We intend to implement a Boolean function

$$next(\underline{q}, \underline{t}, \underline{a}) = (\underline{q}', \underline{t}', \underline{a}')$$

such that $next(< k >) =< l >$, if the configuration following k is l. If k is a halted configuration then $next(< k >) =< k >$. For the implementation of *next* we need a few auxiliary functions. Let $access_n \in \mathrm{B}_{n+\lceil \log n \rceil, 1}$ as in the proof of theorem 2.6. In exercise 4.3, it is shown that $access_n$ can be computed by a circuit of size $O(n)$ and depth $O(\log n)$. Let $modify_n \in \mathrm{B}_{n+\lceil \log n \rceil+1, n}$ be defined by

$$modify_n(\underline{y}, \underline{a}, b) = \underline{z}$$

where

$$|a| = \lceil \log n \rceil, b \in \{0, 1\},$$

$$\underline{y} = y_1 y_2 \cdots y_n,$$

$$\underline{z} = z_1 z_2 \cdots z_n$$

and

$$z_i = \begin{cases} y_i & \text{if } i \neq [a] \\ b & \text{if } i = [a] \end{cases}$$

modify describes writing at a specified address in a store of 1-bit words similar to the way *access* describes reading. In exercise 4.3 it is shown that $modify_n$ can be implemented by a circuit of size $O(n)$ and depth $O(\log\log n)$.

We sketch a circuit for *next*:

1. Read the current symbol below the tape head by using 2 copies of the circuit for *access* (since each symbol on the tape is represented by two bits).
2. Based on what is read and the current state, compute the following values:
   - The new symbol below the tape head.
   - The new state.
   - The value $d \in \{-1, 0, 1\}$ that must be added to the address of the tape head.
3. 2 copies of the circuit for *modify* compute the new tape value, while an *add*-circuit adds d to the head address. (See section 3 for the construction of an add-circuit).

Point 1 is implemented by a circuit of size $O(t(n))$, point 2 by a circuit of size $O(1)$, and point 3 by a circuit of size $O(t(n))$. *next* may also be computed by a circuit of size $O(t(n))$. In total, we have built the following circuit for L_n:

$\underline{z \leftarrow L_n(\underline{x}) :}$
  $\underline{v}^0 \leftarrow$ "Representation of the initial configuration with input $\underline{x}$"
  for $i := 1$ **to** $t(n)$ **do**
    $\underline{v}^i \leftarrow next(\underline{v}^{i-1})$
  od
  $z \leftarrow$ "The truth value of: Is $\underline{v}^i$ an accepting configuration"

Since the first step may be implemented by a circuit of size $O(t(n))$ and the last step by a circuit of size $O(1)$, we find, that L_n can be computed by a circuit of size $O(t(n)^2)$. □

Corollary 4.2 $\mathrm{P} \subseteq pC$

By using nonuniform Turing Machines, we can get a reverse inclusion

Theorem 4.3 P/poly = pC

Proof. $\subseteq$ is proven similarly to the proof for uniform machines. $\supseteq$ is shown by letting Turing Machine T_n have a built-in program that simulates the n'th circuit in a polynomial size family of circuits for $\{f_n\}$. Such a built-in program (represented in a finite control of polynomial size) can be constructed using techniques from dAlg. See exercise 4.5 for details. □

4.4 Space and Depth

This section establishes a close (and perhaps surprising) connection between the space usage necessary to compute a function and the depth complexity of the same function. For the proof we use branching programs.

Theorem 4.4 *Let T be a (possibly nondeterministic) Turing Machine that recognises a language L. If T uses at most $s(n)$ cells on the work tape for any input of size n, where $s(n) \geq \log(n)$, then L_n is computed by a branching program of size $2^{O(s(n))}$. If the Turing Machine is deterministic, so is the branching program.*

Proof. Let n be fixed. A *semi-configuration* is a 3-tuple containing the contents of the $s(n)$ cells used on the work tape, the state of the control unit, and the positions of the two heads. For a fixed input x, the semi-configuration determines a complete configuration. A semi-configuration is described by $O(s(n)) + O(\log n) = O(s(n))$ bits. Hence, there are at most $2^{O(s(n))}$ distinct semi-configurations.

Let V be the set of semi-configurations and let $v \in V$. When the machine is in semi-configuration v, the input head is in a position i_v determined by v. Combined v and the symbol in input cell i_v determines which semi-configurations can follow from a single transition step of the machine.

We define a directed graph with nodes V, where edges are marked by subsets of $\{0, 1, \#\}$: The symbol α is included in the mark on edge (v, w) if and only if the machine can make the transition from v to w, when the input head reads symbol α.

From this graph and knowledge of the initial semi-configuration s and knowledge of the set of accepting semi-configurations F, we have sufficient information to simulate the Turing Machine on all inputs of size n.

Actually, we do not need the symbol # when marking the edges. On the input tape, symbol # occurs in positions 1 and $n+1$, but nowhere else. Hence, we know in advance, for which semi-configurations the input head reads a # and for which it does not. We may therefore assume that all markings are reduced to be subsets of $\{0,1\}$.

We can transform the graph to a branching program. If an edge out of v is marked $\{0\}$, we will write the mark as "$x_{i_v} = 0?$". Similarly, if an edge out of v is marked $\{1\}$, we will write the mark as "$x_{i_v} = 1?$". If an edge is marked $\{0,1\}$, we will write the mark as "true", but if an edge is marked $\emptyset$, we will remove the edge from the graph. By possibly adding a single node with incoming edges from accepting semi-configurations, we can assume that F consists of a single node t.

The constructed branching program computes L_n. □

To get the reverse inclusion, we need to work with nonuniform Turing Machines:

Theorem 4.5 $L \in \mathrm{NL/poly}$ *if and only if* L *has polynomial size branching programmes.*

Proof. The proof of $\Rightarrow$ is similar to the proof of 4.4. Concerning $\Leftarrow$: We may implement a branching program by a nondeterministic program that uses a single variable to hold the current position of the input head. This variable needs $O(\log n)$ bits (space). The finite control unit holds the current node of the branching program, and guesses nondeterministically which edge of the branching program to follow in the next step. The legality of this step is checked by moving the input head to the appropriate position, and reading the symbol under the head. □

This theorem characterises space complexity in terms of branching program complexity. Using a result of section 2.9 we can make a direct connection between space and depth.

Theorem 4.6 $\mathrm{NC}^1 \subseteq \mathrm{L/poly} \subseteq \mathrm{NL/poly} \subseteq \mathrm{AC}^1$

Proof. $\mathrm{NC}^1 \subseteq \mathrm{L/poly}$: Let $L \in \mathrm{NC}^1$ be a family of circuits $\{C_n\}$. The circuit C_n gives rise to an equivalent polynomial size, logarithmic depth formula in reverse Polish notation. The program of Turing Machine T_n evaluates this formula by means of the usual stack algorithm. On the work tape the machine maintains the current stack and the current position of the input head.

$\mathrm{NL/poly} \subseteq \mathrm{AC}^1$: By the previous theorem and corollary 2.12 □

Combined with the result of the next section theorem 4.6 can be seen as a confirming example of the "Parallel Computation Thesis" that asserts *the equivalence of sequential space and parallel time* (up to polynomial blowup/shrinking).

4.5 Parallel Time and Depth

As a model for parallel computation, we use the PRAM known from the dAlg course. As in the case of Turing Machines, we will need a nonuniform version to state our result:

Theorem 4.7 *A language L can be recognised on a nonuniform priority CRCW PRAM using polynomially many processors in time $O(\log^i n)$ if and only if $L \in AC^i$.*

Proof. See exercise 4.8. □

4.6 Probabilistic Machines and non-uniformity

There are problems that we can solve efficiently using a randomised algorithm, and yet no efficient deterministic solution is known.

In this section, we characterise problems having an efficient probabilistic solution, and we show that such problems are computed by a (nonuniform) family of small circuits. One interpretation of this result is that randomisation is a discount version of nonuniformity.

Definition 4.3 *A polynomial time probabilistic Turing Machine (PTM) M that accepts a language L with error probability ϵ is a nondeterministic TM, where*

1. *M has precisely two possible transitions from every non-halting configuration.*
2. *There exists a polynomial p such that every computation on input x has the same length $p(|x|)$ (and ends in a halted accept or reject configuration).*
3. *If $x \in L$ then at most a fraction ϵ of all the $2^{p(|x|)}$ distinct computations on input x leads to a reject. Symmetrically, if $x \notin L$ then at most a fraction ϵ of the distinct computations on input x leads to an accept.*

Let BPP *(Bounded away from $\frac{1}{2}$ Probabilistic Polynomial time) denote the class of languages that are accepted by a PTM with error probability $\epsilon < \frac{1}{2}$.*

BPP is our formalisation of the fuzzy concept "problems that have an efficient randomised solution.

The error probability can be made exponentially small efficiently, also in cases where the given probabilistic algorithm has an error probability close to $\frac{1}{2}$.

Lemma 4.8 *If $L \in$ BPP is accepted by a PTM M with error probability $\epsilon < \frac{1}{2}$ in time $p(n)$, then for every k there exists a PTM M' that accepts L with error probability $\delta = 2^{-k}$ in time $O(k \cdot p(n))$.*

Proof. A machine M' is constructed to simulate M repeatedly, m times, for some odd number m. Among the m results, accept/reject, M' selects the most frequent one. We can bound the error probability for M':

$$\Pr(M' \text{ fails}) \leq$$

$$\sum_{j>\frac{m}{2}} \binom{m}{j} \epsilon^j (1-\epsilon)^{m-j} \leq$$

$$2^m \epsilon^{\frac{m}{2}} (1-\epsilon)^{\frac{m}{2}} \leq$$

$$(4\epsilon(1-\epsilon))^{\frac{m}{2}}$$

Finally, $(4\epsilon(1-\epsilon))^{\frac{m}{2}} < 2^{-k}$ provided

$$m > \frac{2k}{\log(\frac{1}{4\epsilon(1-\epsilon)})} = O(k)$$

□

Randomisation is no stronger than nonuniformity:

Theorem 4.9 (Adleman, 1978) BPP $\subseteq$ P/poly.

Proof. Let $L \in \text{BPP}$. According to lemma 4.8, for every k, we can find a PTM M that accepts $L_k = L \cap \{0,1\}^k$ in time $O(k \cdot p(k))$ with error probability $< 2^{-k-1}$.

There are 2^k distinct inputs, $x_1, x_2, \ldots, x_{2^k} \in \{0,1\}^k$, of length k. For each such input of length k, there are $2^{O(k \cdot p(k))}$ distinct computations in M. These computations may be represented uniquely (and efficiently) with the $2^{O(k \cdot p(k))}$ distinct bit strings, $y_1, y_2, \ldots, y_{2^{O(k \cdot p(k))}}$, of length $O(k \cdot p(k))$.

We can imagine (don't construct it!) a $0,1$-matrix with a row for each x_i and a column for each y_j, where the i,jth entry contains a 1 if and only if the computation represented by y_j leads to the correct answer for input x_i.

Since the error probability of M is at most 2^{-k-1}, the fraction of 0's in any matrix row is upper bounded by the same value 2^{-k-1}. Hence, by a counting argument, there exists at least one matrix column (say the cth corresponding to bit string y_c), where the fraction of 0's is also upper bounded by 2^{-k-1}. However, there are precisely 2^k entries in every column including the cth which implies that the cth column contains 1's only. Hence the bit string y_c represents a computation that leads to the correct answer for all the 2^k distinct inputs.

We construct a Turing Machine T_k that has the string y_c built into its finite control, and on any input (of length k) it simulates the single computation of M represented by y_c.

This can be done for all input sizes k and results in a nonuniform Turing Machine $\{T_k\}$ that accepts L in polynomial time. □

Primality testing is the prototypical example of a problem with a good randomised solution, and no (known) efficient deterministic solution:

Definition 4.4 *Let $composite_n \in \mathrm{B}_n$ be defined by*

$$composite_n(\underline{x}) = \begin{cases} 1 & \textit{if } [\underline{x}] \textit{ is a composite number} \\ 0 & \textit{if } [\underline{x}] \textit{ is a prime number} \end{cases}$$

Theorem 4.10 (Rabin,1980) $\{composite_n\} \in \text{BPP}$

Proof. The algorithm is based on an algebraic result, which we state without proof.

Let $m \geq 3$ be an odd number. Let k, r be defined by $m = 2^k \cdot r + 1$ and r is odd. Let $W(x)$ be the predicate "$x^r \equiv 1 \bmod m$ or $x^{2^i r} \equiv -1 \bmod m$ for some $i \in \{0,1,2,\ldots,k-1\}$".

For composite m it holds that

$$\Pr_{x \in \{1,2,3,\ldots,m-1\}} (W(x)) \leq \frac{1}{4}$$

and for prime m it holds

$$\Pr_{x \in \{1,2,3,\ldots,m-1\}} (\neg W(x)) = 0$$

□

Rabin's test is used a lot in practice. Lemma 4.8 means that a few independent tests can reduce the error probability substantially.

The proofs of lemma 4.8 and theorem 4.9 means that for a fixed input length n, there exists $O(n)$ x-values such that the knowledge of $W(x)$ on these $O(n)$ values determines the primality of x. The computer algebra system *Maple* used in an early version a deterministic "primality test", based on computing $W(x)$ for the 5 specific values $x = 2, 3, 5, 7, 11$. Actually, this suffices to decide the primality of m correctly, for all m less than approximately 10^{10}, but it does not give the correct answer for all m.

By putting space restrictions on randomised algorithms one obtains a variant of theorem 4.9:

Theorem 4.11 *If $L \in \mathrm{BPP}$ is accepted by a PTM with error probability $\epsilon < \frac{1}{2}$ in space $O(\log n)$, then $L \in \mathrm{L/poly}$*

Proof. See exercise 4.9. □

We will show a space efficient randomised solution to a problem, for which all known deterministic solutions use super logarithmic space.

Definition 4.5 *Let $UPATH_{\binom{n}{2}} \in \mathrm{B}_{\binom{n}{2}}$ be defined by*

$$UPATH_{\binom{n}{2}}(\underline{x}) = \begin{cases} 1 & \textit{if the undirected graph with} \\ & \textit{incidence matrix represented by } \underline{x} \\ & \textit{has a path from node } 1 \textit{ to node } n. \\ 0 & \textit{otherwise} \end{cases}$$

Theorem 4.12 (Aleliunas, Karp, Lipton, Lovász, Rackoff, 1979) *UPATH is accepted by a PTM with error probability $\epsilon < \frac{1}{2}$ in space $O(\log n)$.*

Proof. The algorithm is based on the following idea: If a random walk starting from node 1 along the edges of the graph continues for sufficiently many steps (say s steps), it will with high probability pass through node n, provided nodes 1 and n are in the same connected component.

To implement such a random walk, we would need $\log s$ space to count the number of edges traversed so far and $\log n$ space to remember (the number of) the current node.

To compute the relevant probabilities, we use a Markov chain $\{X_0, X_1, \ldots\}$, where $X_i \in \{1, 2, \ldots, n\}$ denotes the number of the node being visited at time i. For technical simplicity, we will assume that the random walk starts at a random node (We return later to the case $X_0 = 1$):

$$\Pr(X_0 = u) = \frac{1}{n} \quad \text{for all } u$$

The transition probabilities are:

$$\Pr(X_{i+1} = u \mid X_i = v) = \begin{cases} \frac{1}{n} & \text{if } (u, v) \text{ is an edge in the graph.} \\ 1 - \frac{\deg(v)}{n} & \text{if } u = v. \\ 0 & \text{otherwise} \end{cases}$$

By induction, it follows that $Pr(X_i = u) = \frac{1}{n}$ for all u and i:

$$\Pr(X_{i+1} = u) = \sum_v \Pr(X_{i+1} = u \mid X_i = v) \cdot \Pr(X_i = v) = \frac{1}{n}$$

Hence, for every edge (u, v) in the graph:

$$\Pr(X_{i+1} = v, X_i = u) = \Pr(X_{i+1} = v \mid X_i = u) \cdot \Pr(X_i = u) = \frac{1}{n^2}$$

i.e. the probability that the random walk uses a given edge at a given time in a given direction is $\frac{1}{n^2}$. Hence, the expected time between two crossings of a given edge in a given direction is n^2. Hence, if we presently stand at one end of the

edge, the expected time until we stand at the other end of the edge is at most n^2:

$$\mathrm{E}(\min_j X_{i+j} = u \mid X_i = v, \text{“there is an edge } (u,v)\text{”}) \leq n^2$$

Since a shortest path from node 1 to node n has length $< n$, we get:

$$\mathrm{E}(\min_i X_i = n \mid X_0 = 1, \text{“there is a path from 1 to } n\text{”}) \leq n^3$$

If a random walk starts at node 1 and continues for $4n^3$ steps with transition probabilities as above then the following holds:

If there is a path from node 1 to node n, then

$$\Pr(\text{The random walk does not pass node } n) \leq \frac{1}{4}$$

If there is not a path from node 1 to node n, then

$$\Pr(\text{The random walk does pass node } n) = 0$$

□

Corollary 4.13 *UPATH* $\in$ L/poly

4.7 Nondeterministic Circuits

When relating space to depth, we got a quadratic slack that hides the difference between deterministic and nondeterministic space measures.

For time we suspect that there is much more than quadratic difference between deterministic and nondeterministic measures. We have so far only related deterministic time to circuit size. By introducing nondeterminism explicitly into the circuit model, we can formulate a correspondence to nondeterministic time:

Definition 4.6 *A nondeterministic circuit N that computes a function $f \in \mathrm{B}_n$ is a circuit with inputs $x_1, \ldots, x_n, y_1, \ldots, y_m$ (for some m) such that*

$$\forall \underline{x} \in \{0,1\}^n.[f(\underline{x}) = 1 \Leftrightarrow \exists \underline{y} \in \{0,1\}^m.N(\underline{x}, \underline{y}) = 1]$$

$y_1, \ldots, y_m$ *are called nondeterministic variables.*

pD *denotes the class of function families* $\{f_n\}$ *that are computed by nondeterministic circuit families of size* $n^{O(1)}$.

Theorem 4.14 NP/poly = pD

Proof. Exercise 4.10 □

Exercises

Exercise 4.1 *Show that the following problems are in* NL*: UPATH, UCON, CYC,* $\overline{2C}$*, where*

- *UPATH decides whether there is a path from node* 1 *to* n *in an undirected graph, when the incidence matrix is given as input.*
- *UCON decides whether an undirected graph is connected, when the incidence matrix is given as input.*
- *CYC decides whether a directed graph has a cycle, when the incidence matrix is given as input.*
- *2C decides whether an undirected graph can be 2-node-coloured, when the incidence matrix is given as input.*

Exercise 4.2 *Show that a deterministic Turing Machine using space* $s(n)$ *can be simulated by a nondeterministic Turing Machine using space* $O(s(n)^2)$. *Hint: Use divide and conquer.*

Exercise 4.3 *Show that* $modify_n$ *is computed by a circuit of size* $O(n)$ *and depth* $O(\log\log n)$, *and show that* $access_n$ *is computed by a circuit of size* $O(n)$ *and depth* $O(\log n)$.

Exercise 4.4 *Suggest another representation of Turing Machine configurations that allows the next-function to be computed by circuits of size* $O(n)$ *and depth* $O(1)$ *(compared to depth* $O(\log n)$*). Hint: Use locality.*

Exercise 4.5 *Fill in the missing details in the proof of theorem 4.3*

Exercise 4.6 *A RAM (random access machine) consists of*

- *A memory with s cells, each holding $\log s$ bits. s depends polynomially on the size of the input to the RAM.*
- *Two registers, named the address register and the data register, each holding $\log s$ bits.*
- *A finite program that consists of a sequence of instructions at the machine language level. Instructions are elementary arithmetic, indirect addressing in the memory though the address register, conditional jumps, etc. The program is placed in ROM (read only memory).*

1. *Describe how the RAM-model can be formalised so it makes sense to state and prove properties of it.*
2. *Argue that the time spent by an optimal RAM to solve a problem corresponds to the time taken by an optimal algorithm (in the more fuzzy notion of time measure introduced at first part courses).*
3. *Show that if a language L is recognised by a RAM in time $O(t)$, then $S(L_n) = O(ts \log t)$.*
4. *Show that if a language L is recognised by a RAM in time $O(t)$, then $S(L_n) = O(t^2 \log^2 t)$. Hint: When $s > t$, we can reduce memory usage by accepting a logarithmic slow down.*

Exercise 4.7 *A RAM is oblivious, if*

- *The line number (location in the program) of the instruction being executed at a given time depends only on the size of the input.*
- *If the RAM at a given time reads or writes to a cell a in the memory, then the address of a depends on the size of the input (and the given time) only.*

1. *Give natural examples of oblivious RAM's.*
2. *Show that if a language L is recognised by an oblivious RAM in time $O(t)$, then $S(L_n) = O(t \log t)$.*

Exercise 4.8 *A priority CRCW-PRAM is a RAM with polynomially many numbered processors that execute private programs synchronously. Several processors may read the same cell from memory simultaneously. If several processors write to the same cell simultaneously, the resulting contents of the cell will be the value written by the processor with the lowest number. It is not allowed for two processors to simultaneously read and write in the same cell.*

- *Show that if a CRCW-PRAM can recognise a language in constant time, then the language is in* AC^0.
- *Show that if a language is in* AC^0*, then it is recognised by a nonuniform CRCW-PRAM in constant time.*
- *Prove theorem 4.7.*

Exercise 4.9 *Show theorem 4.11.*

Exercise 4.10 *Show theorem 4.14*

Literature

1. Karp, R. M. and Lipton, R. J. (1982), Turing Machines that Take Advice. *L'Enseignement Mathematique* **28**, 191–209.

2. Goldschlager, L. M. (1982), A Universal Interconnection Pattern for Parallel Computers. *J. Assoc. Comput. Mach.* **30**, 1073–1086.

3. Pippenger, N. and Fischer, M. J. (1979), Relations Among Complexity Measures. *J. Assoc. Comput. Mach.* **26**, 361–381.

4. Savage, J. E. (1972), Computational Work and Time on Finite Machines. *J. Assoc. Comput. Mach.* **19**, 660–674.

5. Adleman, L. (1978), Two Theorems on Random Polynomial Time. *Proc. 19th Ann. IEEE FOCS*, 75–83.

6. Aleliunas, R., Karp, R. M., Lipton, R. J., Lovász, L and Rackoff, C. (1979), Random Walks, Universal Traversal Sequences and the Complexity of Maze Problems. *Proc. 20th Ann. IEEE FOCS*, 218–223.

7. Rabin, M. O. (1980), Probabilistic Algorithm for Testing Primality. *J. Number Theory* **12**, 128–138.

5 Lower Bounds I

We start by showing exponential lower bounds on the complexity of almost all Boolean functions using a simple counting argument phrased in terms of Kolmogorov complexity.

For specific functions in NP no such bounds are known. The best techniques give only linear and logarithmic lower bounds on size and depth respectively. We present Nečiporuk's technique and the substitution technique for the full binary basis B_2 and Krapchenko's technique for the basis $\{^-, \cdot, +\}$.

Since all efforts to prove super logarithmic lower bounds on depth for specific functions have failed, attention has been given to separate problems within NC^1. We know that $\mathrm{AC}^0 \subseteq \mathrm{NC}^1$, and a by now classic result states that in fact $\mathrm{AC}^0 \neq \mathrm{NC}^1$. Several proofs are known, and we present an algebraic one.

Much stronger lower bounds for monotone circuits are presented in section 7.

5.1 Lower bounds for almost all functions

In section 1, we stressed the fact that a *description method* may often be regarded as a *computation method* and conversely. For instance a formula can be regarded both as the definition of a function and as describing a method for computing the function. We are going to use this phenomenon for giving lower bounds on the complexity of almost all Boolean functions. The method dates back to Riordan and Shannon (1942) and Shannon (1949). Initially, we formalise our notion of a description method.

Definition 5.1 *A description method for the Boolean functions* B *is a mapping* $r : \{0,1\}^* \to \mathrm{B} \cup \{\perp\}$.

An assertion such as $r(0001010001) = OR$, means that the binary string 0001010001 encodes the function $OR \in \mathrm{B}_2$ with respect to the description method r. If $r(\underline{y}) = \perp$ then $\underline{y}$ is not a legal encoding of any function.

Example 5.1 Tabulation t *is a description method defined by*

$$\begin{aligned} t(y_0 y_1 \ldots y_{s-1}) &= \perp, \textit{ if } s \textit{ is not a power } 2. \\ t(y_0 y_1 \ldots y_{2^n-1}) &\in \mathrm{B}_n, \\ t(y_0 y_1 \ldots y_{2^n-1})(\underline{x}) &= y_{[\underline{x}]}, \end{aligned}$$

and e.g. $t(0111) = OR$.

Definition 5.2 *The Kolmogorov complexity measure* K_r *with respect to a description method* r *is defined by*

$$K_r(f) = \min\{|\underline{y}| \mid r(\underline{y}) = f\}$$

The Kolmogorov complexity of a function f with respect to a description method r is the length of the shortest description of the function. For tabulation t we have that $K_t(f) = 2^n$ for all functions $f \in \mathrm{B}_n$.

A simple theorem holds the key to all results in this subsection:

Theorem 5.1 *For every description method* r *there are at least* $2^{2^n} - \sqrt{2^{2^n}}$ *out of the* 2^{2^n} *functions* f *in* B_n *that have Kolmogorov complexity*

$$K_r(f) \geq 2^{n-1}$$

Proof. There are $2^{2^{n-1}} - 1 < \sqrt{2^{2^n}}$ strings of length strictly less than 2^{n-1}, and hence only this number of functions can have less complexity. □

From this point on, we will refer to $2^{2^n} - \sqrt{2^{2^n}}$ out of 2^{2^n} as "almost all", which it certainly is.

We may now prove lower bounds for almost all functions with respect to our usual complexity measures by regarding them as Kolmogorov complexity measures. If we want to perceive circuit size as a description method then we must provide a binary encoding of circuits. For this purpose we will assume the existence of some encoding for the alphabet $A = \{\#, 0, 1, v, x, +, \cdot, ^{-}, \leftarrow\}$ (this can be done with 4 bits per letter). There is a natural encoding of circuits over A, and e.g. the circuit of figure 3 should be encoded as

"$11\#v1 \leftarrow x1 \cdot x10\#v10 \leftarrow x1 \cdot x11\#v11 \leftarrow x10 \cdot x11\#v100 \leftarrow v1 + v10\#v101 \leftarrow v100 + v11$".

The number three at the beginning of the string specifies the arity of the function. This allows us to distinguish the described function from the function in B_4 that ignores its fourth argument, but is computed by the same circuit.

One may observe that the encoding of line i has length $O(\log i + \log n)$ so the encoding of an entire circuit N has length $O(S(N)(\log S(N) + \log n))$.

If we think of the circuit encoding as a description method s then for any function f we have:

$$K_s(f) = O(S(f)(\log S(f) + \log n))$$

and by theorem 5.1:

Corollary 5.2 *Almost all functions f in* B_n *has complexity*

$$S(f) = \Omega(\frac{2^n}{n})$$

In section 2 it was shown that all functions f in B_n have complexity $S(f) = O(2^n/n)$, so the two results combined determine the size complexity of almost all functions up to a multiplicative constant.

We can argue analogously for formula size. Postfix (reverse Polish) notation is a very compact description method for formulae. Using the alphabet $B = \{\#, x, 0, 1, \cdot, +, ^-\}$, the formula in figure 3 may be encoded as

"$11\#x1x10 \cdot x1x11 \cdot +x10x11 \cdot +$"

Such postfix notation for a formula N will have length $O(L(N)\log n)$ in a binary encoding of B, and theorem 5.1 leads to

Corollary 5.3 *Almost all functions f in* B_n *have complexity*

$$L(f) = \Omega(\frac{2^n}{\log n})$$

In section 2, we mentioned the existence of a corresponding upper bound (without proof), so the result is optimal, and we have

Corollary 5.4 *Almost all functions f have depth*

$$D(f) = n - \log\log n - O(1)$$

We may conclude that for almost all functions it holds that

$$D(f) = \Theta(\log S(f)).$$

As mentioned in section 2 it is unknown whether this statement is in fact true for all functions, and there is a widespread belief that this is not the case. The good parallelisation of almost all functions stems from the fact that the general method of computation from theorem 2.4 is parallel (and size optimal for almost all functions).

Though we have determined the complexity of almost all functions, it is a frustrating fact that we are unable to give non-linear lower bounds on size, non-logarithmic lower bounds on depth or non-polynomial lower bounds on formula size for any function family $\{f_n\} \in \mathrm{NP}$.

Our last application of theorem 5.1 will be to projection complexity (definition 2.10). In section 2.7, we saw that the universality of the DNF-family is useless from a practical point of view, since most functions are projections of only very large instances of DNF. This situation does not change when replacing DNF by some stronger family, say a pD-complete family. We shall see that most functions still have high projection complexity.

Given a universal family $\{g_i\}$ we encode a projection of g_m to $f \in \mathrm{B}_n$ given by $\sigma : \{y_1, y_2, y_3, \dots, y_m\} \to \{x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}, 0, 1\}$ as

- A binary encoding of n, followed by a separator #.
- For every j, a binary encoding of $\sigma(y_j)$ with separators between the $\sigma(y_j)$-encodings.

There is no reason to explicitly encode m, since the value of m may be found from the code by counting. The length of a binary version of the code is $O(m \log n)$.

Corollary 5.5 *Given a universal family $\{g_i\}$, for almost all functions $f \in \mathrm{B}_n$ it holds that*

$$C_g(f) = \Omega(\frac{2^n}{\log n})$$

5.2 Lower Bounds for Formula Size

We are going to present two techniques for proving quadratic or almost quadratic lower bounds on the formula size of many concrete problems.

5.2.1 Krapchenko's bound for the basis $\{+, \cdot, ^-\}$.

Definition 5.3 *Let A and B be subsets of $\{0,1\}^n$. We define the set*

$$A \otimes B = \{(x,y) \in A \times B | \text{ } x \text{ and } y \text{ differ in precisely one bit}\}$$

Let $f \in \mathrm{B}_n$ and $A \subseteq f^{-1}(0)$, $B \subseteq f^{-1}(1)$. If $A \otimes B$ is large then f will often change value, when a single bit is flipped. Intuitively, it is difficult to obtain many such changes with $\{+, \cdot, ^-\}$-formulae. More precisely

Theorem 5.6 (Krapchenko, 1972) *Let F be a $\{+, \cdot, ^-\}$-formula for a function $f \in \mathrm{B}_n$, and let $A \subseteq f^{-1}(0)$, $B \subseteq f^{-1}(1)$ with $A \neq \emptyset$ and $B \neq \emptyset$. It holds that*

$$L(F) \geq \frac{|A \otimes B|^2}{|A||B|} - 1.$$

Proof. The theorem is proved by induction in the formula size $L(F)$.

$L(F) = 0$: In this case F is a variable x_i. Therefore $x_i = 0$ for all $\underline{x} \in A$ and $x_i = 1$ for all $\underline{x} \in B$, which implies that there is at most a single $\underline{y} \in B$ with $(x,y) \in A \otimes B$ for each $\underline{x} \in A$. We deduce that $|A \otimes B| \leq |A|$, and similarly $|A \otimes B| \leq |B|$. This proves the theorem.

$L(F) > 0$: There are three cases:

- $F = F_1 \cdot F_2$. Let $A_1 = \{\underline{x} \in A | F_1(\underline{x}) = 0\}$ and $A_2 = \{\underline{x} \in A | F_1(\underline{x}) = 1\}$. Note that $F_2(\underline{x}) = 0$ for $\underline{x} \in A_2$. If $A_1 \neq \emptyset$ then the triple F_1, A_1, B satisfies the assumption of the theorem and by induction

  $$L(F_1) \geq \frac{|A_1 \otimes B|^2}{|A_1||B|} - 1,$$

  and similarly if $A_2 \neq \emptyset$ then the triple F_2, A_2, B satisfies the assumption of the theorem and by induction

  $$L(F_2) \geq \frac{|A_2 \otimes B|^2}{|A_2||B|} - 1.$$

  Using these inequalities, we calculate:

  $$L(F) - (\frac{|A \otimes B|^2}{|A| \cdot |B|} - 1)$$

$$= L(F_1) + L(F_2) + 1 - (\frac{|A \otimes B|^2}{|A| \cdot |B|} - 1)$$

$$\geq (\frac{|A_1 \otimes B|^2}{|A_1||B|} - 1) + (\frac{|A_2 \otimes B|^2}{|A_2||B|} - 1) + 1 - (\frac{|A \otimes B|^2}{|A| \cdot |B|} - 1)$$

$$= \frac{(|A_1 \otimes B||A_2| - |A_2 \otimes B||A_1|)^2}{|A_1||A_2|(|A_1| + |A_2|)|B|}$$

$$\geq 0$$

If $A_1 = \emptyset$ then $A = A_2$ and we have

$$L(F) > L(F_2) \geq \frac{|A_2 \otimes B|^2}{|A_2||B|} - 1 =$$

$$\frac{|A \otimes B|^2}{|A||B|} - 1.$$

If $A_2 = \emptyset$ then the argument is similar.

- $F = F_1 + F_2$. This case is handled in a similar manner (not A, but B is divided).
- $F = \bar{F_1}$. We use the induction hypothesis on F_1 with A and B interchanged.

□

Since $|A \otimes B| \leq n \min\{|A|, |B|\}$, the theorem can never lead to a better lower bound than $n^2 - 1$ for any function. Andrejev (1987) has proven a lower bound of $n^{2.5+\epsilon}$ on the formula size for a specific function over the basis $\{+, \cdot, \bar{}\}$ using a different technique. The parity function can make optimal use of Krapchenko's theorem:

$$parity_n(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } |\{i | x_i = 1\}| \text{ is even} \\ 1 & \text{if } |\{i | x_i = 1\}| \text{ is odd} \end{cases}$$

Note that $parity_n(\underline{x}) \neq parity_n(\underline{y})$ if $\underline{x}$ and $\underline{y}$ differ in precisely one bit. If we let $A = parity_n^{-1}(0)$ and $B = parity_n^{-1}(1)$ then $|A \otimes B| = n|A| = n|B|$ and the theorem tells us that

$$L(parity_n) \geq n^2 - 1$$

There is a matching upper bound (see exercise 5.2). Note that $parity_n$ has a formula of size $n-1$ over the basis $\{\oplus, \cdot\}$, i.e. the formula size is basis dependent.

5.2.2 Nečiporuk's bound for the full binary basis B_2.

By a different technique, we may prove almost quadratic lower bounds for formula size over the full binary basis B_2.

Definition 5.4 *Let $f \in \mathrm{B}_n$ be a function on $X = \{x_1, \ldots, x_n\}$. Let $Y \subseteq \{x_1, \ldots, x_n\}$ satisfy $Y \neq \emptyset$. $g \in \mathrm{B}_{|Y|}$ is said to be a* sub-function *of f on Y, if g can be defined by substituting constants for the variables in $X - Y$, and leaving the variables in Y. We let $U_Y(f)$ denote the set of* non-constant *sub-functions on Y, and their negations.*

Consider the example of $parity_n$ that has only few distinct sub-functions:

$$U_Y(parity_n) = \{parity_{|Y|}, \overline{parity_{|Y|}}\}.$$

Intuitively, it is impossible to express all information about a function in a small formula, if it has many sub-functions.

Theorem 5.7 (Nečiporuk, 1966) *Let $f \in \mathrm{B}_n$ be a function on $x_1, \ldots, x_n$. Let $Y_1, \ldots, Y_k$ be disjoint subsets of $\{x_1, \ldots, x_n\}$. Then*

$$L_{\mathrm{B}_2}(f) \geq \sum_{i=1}^{k} \log_5(2|U_{Y_i}(f)| + 1) - 1$$

Proof. For every formula F on $x_1, \ldots, x_n$, and $Y \subseteq \{x_1, \ldots, x_n\}$, we let $|F|_Y$ denote the total number of occurrences in F of variables from Y. We are going to show by induction over $L(F)$ that

$$2|U_Y(F)| + 1 \leq 5^{|F|_Y}$$

For $L(F) = 0$ we have two cases

- F is a constant or a variable $x_i \notin Y$. In this case $|U_Y(F)| = 0$ and $|F|_Y = 0$.
- F is a variable $x_i \in Y$. In this case $|U_Y(F)| = 2$ and $|F|_Y = 1$.

If $F = F_1 * F_2$ for $* \in \mathrm{B}_2$ then a non-constant sub-function G of F can take one of the following forms (G being defined on Y):

- $G = G_1 * G_2$, where G_1 and G_2 are non-constant sub-functions of F_1 and F_2 on Y. Including negated functions we get a contribution of at most $2|U_Y(F_1)||U_Y(F_2)|$ to $|U_Y(F)|$.

- $G = G_1 * G_2$, where G_1 is a non-constant sub-function of F_1 and G_2 is a constant sub-function of F_2. If G is non-constant then $G = G_1$ or $G = \overline{G_1}$, and in any case $G, \bar{G} \in U_Y(F_1)$, so the contribution to $|U_Y(F)|$ is at most $|U_Y(F_1)|$.

- $G = G_1 * G_2$, where G_1 is a constant sub-function of F_1 and G_2 is a non-constant sub-function of F_2. In this case, the contribution to $|U_Y(F)|$ is similarly at most $|U_Y(F_2)|$.

In total we have

$$|U_Y(F)| \leq 2|U_Y(F_1)||U_Y(F_2)| + |U_Y(F_1)| + |U_Y(F_2)|$$

Using the induction hypothesis, we get

$$2|U_Y(F)| + 1 \leq 4|U_Y(F_1)||U_Y(F_2)| + 2|U_Y(F_1)| + 2|U_Y(F_2)| + 1 =$$

$$(2|U_Y(F_1)| + 1)(2|U_Y(F_2)| + 1) \leq 5^{|F_1|_Y} 5^{|F_2|_Y} = 5^{|F_1|_Y + |F_2|_Y} = 5^{|F|_Y}$$

Note that if $Y_1, Y_2, \ldots, Y_k$ are disjoint subsets of X, then $L(F) \geq \sum_{i=1}^{k} |F|_{Y_i} - 1$. This implies the theorem. □

For an application of Nečiporuk's technique, we consider the example of indirect store access. Let $p = 2^{2^l}$ for some l, let $k = \log p - \log\log p$ and let $n = 2p + k$. For ISA $\in \mathrm{B}_n$ the n input are divided in 3 groups.

- $\underline{x} = (x_0, x_1, \ldots, x_{p-1})$ is the actual store.

- $\underline{y} = (y_1, y_2, \ldots, y_p)$ contains addresses into the store. Each address requires $\log p$ bits, which leaves room for a total of $p/\log p$ addresses.

- $\underline{a} = (a_0, a_1, \ldots, a_{k-1})$ specifies which of the addresses in $\underline{y}$ that should be used. This is well-defined since $[\underline{a}]$ takes values in the interval from 0 to $p/\log p - 1]$.

Formally, we define

$$\mathrm{ISA}(\underline{x}, \underline{y}, \underline{a}) = x_{[\underline{d}]},$$

where

$$d = (y_{[\underline{a}]\log p+1}, y_{[\underline{a}]\log p+2}, \ldots, y_{([\underline{a}]+1)\log p}).$$

For the application of theorem 5.7 we choose

$$Y_i = \{y_{i\log p+1}, y_{i\log p+2}, \ldots, y_{(i+1)\log p}\} \quad \text{for} \quad 0 \le i \le \ p/\log p - 1.$$

The reader should convince herself that $U_{Y_i}(\mathrm{ISA}) = \mathrm{B}_{\log p}$ and therefore

$$L_{\mathrm{B}_2}(\mathrm{ISA}) \ge \sum_{i=1}^{p/\log p} \log_5(2|U_{Y_i}(\mathrm{ISA})| + 1) - 1 \ge$$

$$\frac{p}{\log p}\log_5(2 \cdot 2^{2^{\log p}} + 1) - 1 \ge$$

$$(\log_5 2)\frac{p^2}{\log p} - 1 = \Omega(\frac{n^2}{\log n})$$

This is in fact the best bound one can prove for any function using theorem 5.7 (see exercise 5.5).

The theorems of Nečiporuk and Krapchenko give as corollaries lower bounds of at most $2\log n$ on the depth of the function in question. No techniques are known for proving super logarithmic depth bounds for specific functions. This is somewhat unsatisfactory in the light of corollary 5.4, which states that almost all functions have depth $\Omega(n)$.

5.3 Lower bounds for circuit size

For circuit size the situation is similar. From corollary 5.2 we know that almost all functions requires size $\Omega(2^n/n)$, but (over the full binary basis B_2) the best known lower bound for any function in NP is $3n - o(n)$ proven by Blum (1984). We present a slightly weaker result.

5.3.1 The substitution technique

Theorem 5.8 $S_{\mathrm{B}_2}(Th^2_n) \geq 2n - 3$

Proof. The theorem is proven by induction in n. For $n = 2$ the theorem is trivially true. For $n \geq 3$, we are going to show that

$$S_{\mathrm{B}_2}(Th^2_{n-1}) \leq S_{\mathrm{B}_2}(Th^2_n) - 2,$$

which implies the induction step. Let N be a size optimal straight-line program for Th^2_n. We are going to construct a program N' for Th^2_{n-1}, such that N' is two lines shorter. Let

$$v_1 \leftarrow x_i * x_j$$

be the first line of N. Since N is optimal, we have $i \neq j$. We will argue that at least one of x_i and x_j occurs in some other line of the program. Assume, contrarily, that the two variables occur in the first line only. In that case all available information about x_i and x_j are concentrated in the variable v_1, and since v_1 can hold only two distinct values, there must be two of the following assignments to x_i and x_j that results in identical v_1 values.

- $x_i = 0$, $x_j = 0$.
- $x_i = 0$, $x_j = 1$.
- $x_i = 1$, $x_j = 1$.

However, for any two of the three cases it is possible to assign values to the remaining inputs such that the output from the function differs depending on the case. This is a contradiction, and we deduce that at least one of x_i and x_j occurs in another right hand side. Assume x_i occurs at least twice. We construct N' from N by a transformation:

- Replace all occurrences of x_i with 0.
- Some assignments contain constants now. These assignments are eliminated by either modifying the operation that uses the result (we are allowed to use all of B_2), or by modifying the operation that created the non-constant argument.

This transformation eliminates at least two lines. □

Boolean function	Equivalent polynomial	
	over GF(3)	over Q
0	0	0
1	1	1
$\overline{x}$	$1-x$	$1-x$
$x \cdot y$	xy	xy
$x+y$	$x+y-xy$	$x+y-xy$
$x \oplus y$	$x+y+xy$	$x+y-2xy$

Figure 15:

5.4 Lower bounds for unbounded fan-in circuits

It is unknown whether all functions in NP parallelise well, but we shall prove in this section that not all functions in NP parallelise extremely well. $parity_n \in \mathrm{NC}^1$ since it can be computed in depth $O(\log n)$, but $parity_n \notin \mathrm{AC}^0$, i.e. *parity* requires super polynomial circuit size over the basis $\mathcal{U} = \{^-, +_2, \cdot_2, +_3, \cdot_3, \ldots\}$, when the depth is bounded by a constant. This result was first proven by Furst, Saxe and Sipser (1984). We present a proof due to Beigel, Reingold and Spielman (1991).

The essence of the proof is a concept of *approximator* satisfying that every function in AC^0 has a good approximator but *parity* has no good approximator.

Approximators are polynomials. We will mostly use polynomials over the field of rational numbers, but occasionally we need polynomials over GF(3), the field with 3 elements. We represent GF(3) by the set $\{-1, 0, 1\}$ with modulo 3 arithmetic.

Notation 5.1 $0, 1$ *denote both arithmetic constants and Boolean constants. We do not distinguish the two cases.*

Definition 5.5 *A polynomial p with variables $x_1, x_2, \ldots, x_n$ is equivalent to a Boolean function $f \in \mathrm{B}_n$ if $p(\underline{x}) = f(\underline{x})$ for all $\underline{x} \in \{0,1\}^n$.*

Figure 15 shows a selection of Boolean functions with equivalent polynomials. In general we have

Lemma 5.9 *Every function in B_n has an equivalent polynomial of degree at most n.*

Proof. Exercise 5.6. □

We are interested in polynomials of degree less than those we can find using lemma 5.9. In return, we are willing to relax the notion of equivalence.

Definition 5.6 *A (d, ϵ)-L-approximator (or simply a (d, ϵ)-approximator) for a Boolean function $f \in \mathrm{B}_n$ is a polynomial g over the field L with variables $x_1, x_2, \ldots, x_n$ such that*

1. *degree$(g) \le d$.*
2. *$g(\underline{x}) = f(\underline{x})$ for all $\underline{x} \in \{0,1\}^n$ except for a fraction $\le \epsilon$.*

When a Boolean function is given by composition such as

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

it may happen that good approximators for $g, h_1, \ldots, h_n$ result in a bad approximator for f when composed (see exercise 5.8). To deal with this problem we need another notion of approximation.

Definition 5.7 *A probabilistic (d, ϵ)-approximator for a function $f \in \mathrm{B}_n$ is a polynomial g with variables $x_1, \ldots, x_n, w_1, \ldots, w_m$ for some m such that*

1. *degree$(g) \le d$.*
2. *$\Pr_{\underline{w} \in \{0,1\}^m}(g(\underline{x}, \underline{w}) \ne f(\underline{x})) \le \epsilon$ for all $\underline{x} \in \{0,1\}^n$.*

$w_1, \ldots, w_m$ are hidden *variables.*

Composition now preserves good approximation:

Lemma 5.10 *If $g \in \mathrm{B}_m$ and $h_1, \ldots, h_m \in \mathrm{B}_n$ all have probabilistic (d, ϵ)-approximators then $f \in \mathrm{B}_n$ defined by*

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

has a probabilistic $(d^2, (m+1)\epsilon)$-approximator.

Proof. Let p_F be a probabilistic (d, ϵ)-approximator for $F \in \{g, h_1, \ldots, h_m\}$ by the assumption of the lemma. We may assume that no two hidden variables from distinct p_F's have identical names (if necessary change names). This means that an approximator will answer correctly/wrongly independent of the other approximators, and we may form a probabilistic approximator p_f as required simply by substituting p_{h_i} for x_i in p_g. The degree of p_f is at most d^2 and the probability of error is bounded by $(m+1) \cdot \epsilon$ for input $\underline{x} \in \{0,1\}^n$. □

It is possible to remove randomisation from an approximator:

Lemma 5.11 *If $f \in$ B has a probabilistic (d, ϵ)-approximator then there also exists a (non-probabilistic) (d, ϵ)-approximator for f.*

Proof. Exercise 5.9 □

At this point, we are ready to describe in more detail our strategy for finding a lower bound on parity.

1. Show that $\mathrm{OR}(x_1, \ldots, x_n)$ has a good probabilistic approximator of low degree.
2. Show that every function in AC^0 has a good probabilistic approximator of low degree using lemma 5.10.
3. Show that parity does not have a GF(3)-approximator of low degree.
4. Combine the results using lemma 5.11.

For the construction of the probabilistic OR-approximator we need a lemma that regards $\{0,1\}^n$ as an n-dimensional vector space over GF(2), the field with 2 elements. We represent GF(2) by the set $\{0,1\}$ with arithmetic modulo 2.

Let $V = \{0,1\}^k$ be the k-dimensional vector space over GF(2). For $\underline{x}, \underline{y} \in V$ we let $\underline{x} \cdot \underline{y} = \sum_{i=1}^{k} x_i y_i \pmod 2$ denote the inner product on V.

Lemma 5.12 (Valiant and Vazirani, 1986) *For $S \subseteq V$ and $\underline{w}_1, \ldots, \underline{w}_k \in V$ we define the hyperplanes*

$$H_i = \{\underline{v} \in V \mid \underline{v} \cdot \underline{w}_i = 0\} \quad \textit{for } i = 1, \ldots, k$$

and the sets

$$S_i = S \cap H_1 \cap H_2 \cap \cdots \cap H_i \quad \textit{for } i = 0, \ldots, k.$$

It holds that

$$\Pr_{\underline{w}_1,\ldots,\underline{w}_k \in V}(\ |S_i| = 1 \ \ \textit{for some } i \in \{0, 1, \ldots, k\} \ ) \geq \frac{1}{4}$$

for all $S \neq \emptyset$.

Proof. We argue separately in the two cases $\underline{0} \in S$ and $\underline{0} \notin S$.

The case $\underline{0} \in S$: Clearly $|S_k| \geq 1$ since $\underline{0} \in H_i$ for all i. Let $\mathbf{W}$ denote the matrix with row vectors $\underline{w}_1, \ldots, \underline{w}_k$. $|S_k| = 1$ when the equation $\mathbf{W}\underline{x} = \underline{0}$ has only the trivial solution $\underline{x} = \underline{0}$, i.e. $\underline{w}_1, \ldots, \underline{w}_k$ are linearly independent. To calculate the probability of such event, we assume that the w_i's are selected one by one (randomly and independently).

$$\Pr_{\underline{w}_i \in V}(\underline{w}_i \notin \ \mathrm{span}\{\underline{w}_1, \ldots, \underline{w}_{i-1}\} \mid \underline{w}_1, \ldots, \underline{w}_{i-1} \text{ are lin. indep. }) =$$

$$\frac{2^k - 2^{i-1}}{2^k} = 1 - \frac{1}{2^{k-i+1}}.$$

In total for the case $\underline{0} \in S$ we have:

$$\Pr_{\underline{w}_1,\ldots,\underline{w}_k \in V}(\ |S_i| = 1 \ \ \text{for some } i \in \{0, 1, \ldots, k\} \ ) \geq$$

$$\Pr_{\underline{w}_1,\ldots,\underline{w}_k \in V}(\underline{w}_1, \ldots, \underline{w}_k \text{ are linearly independent}) =$$

$$\prod_{i=1}^{k}(1 - \frac{1}{2^i}) \geq \frac{1}{4}.$$

For the last inequality see exercise 5.10.

The case $\underline{0} \notin S$: Let

$$A(r) = \Pr_{\underline{w}_1,\ldots,\underline{w}_r \in V}(\ \exists i \in \{0,1,\ldots,r\}.\ |S_i| = 1 \ \ |\ \ \mathrm{rank}(S) = r).$$

The following inequality (combined with exercise 5.10) implies the lemma.

$$A(r) \geq \prod_{i=1}^{r-1}(1 - \frac{1}{2^i}). \tag{7}$$

Note that when $S \neq \emptyset$ and $\underline{0} \notin S$ then $\mathrm{rank}(S) \geq 1$. Hence it suffices to prove (7) for $r \geq 1$. We use induction

Basis of induction ($r = 1$): We have $|S| = 1$ since $\mathrm{rank}(S) = 1$ and $\underline{0} \notin S$.

Induction step (assume (7) for $1 \leq r < r'$): Let $\mathrm{rank}(S) = r'$ and let $\underline{e}_i = (0^{i-1}10^{k-i})$ be the i'th unit vector. Without loss of generality (if necessary apply a linear transformation) we may assume that $\{\underline{e}_1,\ldots,\underline{e}_{r'}\} \subseteq S \subseteq \ \mathrm{span}\{\underline{e}_1,\ldots,\underline{e}_{r'}\}$.

If $S_1 = \emptyset$ then $\underline{w}_1 \cdot \underline{e}_1 = \cdots = \underline{w}_1 \cdot \underline{e}_{r'} = 1$, i.e. $\underline{w}_1 \in 1^{r'}\{0,1\}^{k-r'}$. The probability of this event is $\leq \frac{1}{2^{r'}}$.

If $S_1 = S$ then $\underline{w}_1 \cdot \underline{e}_1 = \cdots = \underline{w}_1 \cdot \underline{e}_{r'} = 0$, i.e. $\underline{w}_1 \in 0^{r'}\{0,1\}^{k-r'}$. The probability of this event is $\leq \frac{1}{2^{r'}}$.

If $\emptyset \neq S_1 \neq S$ then $1 \leq \mathrm{rank}(S_1) \leq \mathrm{rank}(S) - 1$. We have

$$A(r') = \Pr_{\underline{w}_1,\ldots,\underline{w}_{r'} \in V}(\ \exists i \in \{0,1,\ldots,r'\}.\ |S_i| = 1 \ \ |\ \ \mathrm{rank}(S) = r') \geq$$

$$\Pr_{\underline{w}_1 \in V}(\ 1 \leq \mathrm{rank}(S_1) \leq r' - 1 \mid \mathrm{rank}(S) = r')\cdot$$

$$\min_{1 \leq r \leq r'-1} \Pr_{\underline{w}_1,\ldots,\underline{w}_r \in V}(\ \exists i \in \{0,1,\ldots,r\}.\ |S_i| = 1 \ \ |\ \ \mathrm{rank}(S) = r) \geq$$

$$(1 - \frac{1}{2^{r'}} - \frac{1}{2^{r'}}) \cdot \prod_{i=1}^{r'-2}(1 - \frac{1}{2^i}) = \prod_{i=1}^{r'-1}(1 - \frac{1}{2^i}).$$

□

Lemma 5.13 *For every $\epsilon > 0$, there exists a probabilistic $(O(\log(\epsilon^{-1})\log^3 n), \epsilon)$-approximator for* $\mathrm{OR}(x_1, \ldots, x_n)$.

Proof. Let $k = \log n$. We begin by constructing a $(O(\log^3 n), \frac{3}{4})$-approximator P that makes one-sided error only and has k^2 hidden variables, i.e. for all $\underline{x} \in \{0,1\}^n$ we want P to satisfy

1. If $\mathrm{OR}(\underline{x}) = 0$ then $\mathrm{Pr}_{\underline{w}\in\{0,1\}^{k^2}}(P(\underline{x}, \underline{w}) \neq 0) = 0$.
2. If $\mathrm{OR}(\underline{x}) = 1$ then $\mathrm{Pr}_{\underline{w}\in\{0,1\}^{k^2}}(P(\underline{x}, \underline{w}) \neq 1) \leq \frac{3}{4}$.

To prepare the application of lemma 5.12 we define for an input $\underline{x}$ the set $S = \{\underline{v} \in \{0,1\}^k \mid x_{[\underline{v}]} = 1\}$, and therefore $\mathrm{OR}(x_1, \ldots, x_n) \equiv (S \neq \emptyset)$. The k^2 hidden variables are structured in k vectors: $\underline{w} = (\underline{w}_1, \ldots, \underline{w}_k)$, where $\underline{w}_i = (w_{i1}, \ldots, w_{ik})$. Define *ortho* $\in \mathrm{B}_{2k}$ by

$$ortho(\underline{x}, \underline{y}) = \begin{cases} 1 & \text{if } \underline{x} \cdot \underline{y} = 0 \text{ (in the vector space V !)} \\ 0 & \text{otherwise} \end{cases}$$

Let p_{ortho} be some polynomial of degree $\leq 2k$ that is equivalent to *ortho* (using lemma 5.9), and let for $0 \leq i \leq k$, P_i be the polynomial

$$P_i = \sum_{\underline{v}\in\{0,1\}^k} x_{[\underline{v}]} \cdot p_{ortho}(\underline{v}, \underline{w}_1) \cdot \ldots \cdot p_{ortho}(\underline{v}, \underline{w}_i)$$

If we use the definition of S_i from lemma 5.12, then $|S_i| = P_i$. According to the same lemma

$$\Pr_{\underline{w}\in V^k}(\text{there exists } i \text{ such that } \ P_i(\underline{x}, \underline{w}) = 1) \geq \frac{1}{4}$$

when $\mathrm{OR}(\underline{x}) = 1$. In addition we have that $P_i(\underline{x}, \underline{w}) = 0$ for all i and all $\underline{w}$, when $\mathrm{OR}(\underline{x}) = 0$.

This one-sided error is used in the construction of the approximator:

$$P = 1 - \prod_{i=0}^{k}(1 - P_i).$$

P has degree at most $(k+1)(1+k\cdot 2k) = O(k^3)$ and P makes only one-sided error:

$$\Pr_{\underline{w}\in V^k}(P(\underline{x},\underline{w})=1) \geq \frac{1}{4} \quad \text{when } \mathrm{OR}(\underline{x})=1,$$

and $P(\underline{x},\underline{w})=0$ if $\mathrm{OR}(\underline{x})=0$ (independent of $\underline{w}$).

The error probability can be made arbitrarily small by using several (independent) versions of P.

Given $\epsilon > 0$, let d be minimal such that $(\frac{3}{4})^d \leq \epsilon$, and let $Q_1, Q_2, \ldots, Q_d$ be copies of P. We assume (if necessary change variable names) that no pair of Q_i's have hidden variables in common. Let

$$Q = 1 - \prod_{i=1}^{d}(1-Q_i)$$

The degree of Q is $O(d\cdot\log^3 n) = O(\log(\epsilon^{-1})\log^3 n)$ and

$$\Pr_{\underline{w}'\in V^{kd}}(Q(\underline{x},\underline{w}') \neq \mathrm{OR}(\underline{x})) \leq \epsilon$$

The approximator Q satisfies the assertion of the lemma. □

Lemma 5.14 *If $f \in \mathrm{B}_n$ is computed by a $\mathcal{U}$-circuit of size s ($\geq n$) and depth d then f has a probabilistic $(O((\log(s\cdot\epsilon^{-1})\log^3 s)^d),\epsilon)$-approximator for every $\epsilon > 0$.*

Proof. Without loss of generality we may assume that the circuit for f uses negation and OR only, and an OR-gate has at most s input. Lemma 5.13 supplies us with a probabilistic $(O(\log(s\cdot\epsilon^{-1})\log^3 s),\epsilon/s)$-approximator for $\mathrm{OR}(y_1,\ldots,y_s)$, and the polynomial $1-y$ is equivalent to $\overline{y}$. By repeated use of (the proof of) lemma 5.10 we get a probabilistic $(O((\log(s\cdot\epsilon^{-1})\log^3 s)^d),\epsilon)$-approximator for f. □

Lemma 5.15 *Every (d,ϵ)-GF(3)-approximator for parity$(x_1,\ldots,x_n)$ satisfies that $d \geq (\frac{1}{2}-\epsilon)\cdot\sqrt{n}-1$.*

Proof. In GF(3) we have

$$x - 1 = \begin{cases} 0 & \text{for } x = 1 \\ 1 & \text{for } x = -1 \end{cases}$$

This equation implies that

$$\prod_{i=1}^{n} x_i = parity_n(x_1 - 1, x_2 - 1, \ldots, x_n - 1) + 1$$

for $x_1, x_2, \ldots, x_n \in \{-1, 1\}$.

We see that an approximator of low degree for *parity* leads to an approximator of low degree for an arbitrary monomial of high degree. A polynomial is simply a sum of monomials, so we can easily form an approximator of low degree for an arbitrary polynomial. Unfortunately, the error from the single monomials will accumulate, if we use that approach blindly. We can get around the problem by using *parity* only once.

In the sequel assume that $x_1, x_2, \ldots, x_n \in \{-1, 1\}$. Let $I \subseteq \{1, 2, \ldots, n\}$ and let $m_I = \prod_{x \in I} x_i$. We have

$$m_I = m_{\overline{I}} \cdot \prod_{i=1}^{n} x_i = m_{\overline{I}} \cdot (parity_n(x_1 - 1, x_2 - 1, \ldots, x_n - 1) + 1).$$

Let p be a (d, ϵ)-GF(3)-approximator for *parity* and let $m'_I = m_{\overline{I}} \cdot (p(x_1 - 1, x_2 - 1, \ldots, x_n - 1) + 1)$. We have

1. For at least a fraction $(1-\epsilon)$ of all $\underline{x} \in \{-1, 1\}^n$ it holds that $m_I(\underline{x}) = m'_I(\underline{x})$ for all I.
2. One of m_I and m'_I has degree $\leq \frac{n}{2} + d$.

Let $f \in \{-1, 1\}^n \mapsto \{-1, 0, 1\}$ be represented by the polynomial p_f. Since $x^2 = 1$ for $x \in \{-1, 1\}$ we may assume that all monomials in p_f has the form m_I for some I. We construct $\tilde{p_f}$ from p_f by replacing all monomials m_I of degree larger than $\frac{n}{2}$ by m'_I. We have

1. For at least a fraction $(1 - \epsilon)$ of all $\underline{x} \in \{-1, 1\}^n$ it holds for all f that $\tilde{p_f}(\underline{x}) = p_f(\underline{x})$.

2. $\tilde{p_f}$ has degree at most $\frac{n}{2} + d$.

We may now deduce that

1. For every f it holds that $|\{g \mid \tilde{p_f} = \tilde{p_g}\}| \leq 3^{\epsilon \cdot 2^n}$.
2. There are at most $3^{\sum_{i=1}^{\frac{n}{2}+d} \binom{n}{i}} \leq 3^{(\frac{1}{2}+\frac{d+1}{\sqrt{n}}) \cdot 2^n}$ distinct $\tilde{p_f}$. (see exercise 5.11 for the inequality).

There are 3^{2^n} distinct f, meaning that $3^{(\frac{1}{2}+\frac{d+1}{\sqrt{n}}+\epsilon) \cdot 2^n} \geq 3^{2^n}$ or $d \geq (\frac{1}{2} - \epsilon) \cdot \sqrt{n} - 1$. □

Theorem 5.16 *If $parity_n$ is computed by a $\mathcal{U}$-circuit of size s and depth d, then*

$$d = \Omega(\frac{\log n}{\log \log s}).$$

Proof. Assume that $parity_n$ is computed by a $\mathcal{U}$-circuit of size s and depth d. Using lemma 5.14 combined with lemma 5.11, we see that *parity* has an $(O((\log(s \cdot \epsilon^{-1}) \log^3 s)^d), \epsilon)$-approximator. This approximator is in particular a GF(3)-approximator (if necessary use a modulo 3 operation on the coefficients). According to lemma 5.15 we have that

$$c \cdot (\log(s \cdot \epsilon^{-1}) \log^3 s)^d \geq (\frac{1}{2} - \epsilon)\sqrt{n} - 1$$

for some constant c. The theorem follows by substituting $\epsilon = \frac{1}{4}$ and taking the logarithm on both sides. □

Corollary 5.17 *parity* $\notin \mathrm{AC}^0$

Exercises

Exercise 5.1 *Let $\{g_n\}$ be a universal family of functions and let M be a complexity measure (say S or L). As usual $C_g(f)$ denotes the smallest i such that f is a projection of g_i. We define g to* characterise *M up to a constant, if*

$$C_g(f) = \Theta(M(f)).$$

Show that no *family characterises S up to a constant. Is L characterised up to a constant by any family ?*

Exercise 5.2 *Show that $L(parity_n) = O(n^2)$ over the basis $\{+, \cdot, ^-\}$.*

Exercise 5.3 *Show that over the basis $\{+, \cdot, ^-\}$ we have*

1. $L(Th_n^k) = \Omega(k(n-k))$.
2. $L(Eq_n^M) = \Omega(\max_{k \in M \Leftrightarrow k-1 \notin M} k(n-k))$.

Exercise 5.4 *Let p be a power of 2, let $n = 2p \log p$ and let $distinct_n \in \mathrm{B}_n$ be defined by*

$$distinct_n(x_1, \ldots, x_p) = \begin{cases} 1 & \text{if } x_i \neq x_j \text{ for all } i \neq j \\ 0 & \text{otherwise} \end{cases},$$

where $|x_i| = 2 \log p$.

Show that $L_{\mathrm{B}_2}(distinct_n) = \Omega(n^2/\log n)$. Find an upper bound for comparison.

Exercise 5.5 *Show that Nečiporuk's theorem can never give a lower bound better than $\Omega(n^2/\log n)$.*

Exercise 5.6

1. *Prove lemma 5.9*
2. *Find a polynomial of degree 3 equivalent to Th_3^2.*

Exercise 5.7 *How well do the following polynomials approximate $\mathrm{OR}(x_1, \ldots, x_n)$:*

1. $1 - \prod_{i=1}^n (1 - x_i)$
2. $\sum_{i=1}^n x_i$

Exercise 5.8 *Why is the proof of lemma 5.10 not valid for (non-probabilistic) approximators ?*

Exercise 5.9 *Prove lemma 5.11*

Exercise 5.10 *1. Show that* $\log_4 x \geq x - 1$ *for* $\frac{1}{2} \leq x \leq 1$.

2. Show that $\prod_{i=1}^{\infty}(1 - \frac{1}{2^i}) \geq \frac{1}{4}$.

Exercise 5.11 *Prove that for n sufficiently large, we have*

$$\sum_{i=1}^{\frac{n}{2}+d} \binom{n}{i} \leq (\frac{1}{2} + \frac{d+1}{\sqrt{n}}) \cdot 2^n$$

for all $0 \leq d \leq \frac{n}{2}$.

Hint: By Stirling approximation we have that $n! = \sqrt{2\pi n} \cdot (\frac{n}{e})^n \cdot (1 + O(\frac{1}{n}))$

Exercise 5.12 *Show that the depth* $\Theta(\frac{\log n}{\log \log n})$ *is necessary and sufficient for a polynomial size family of* $\mathcal{U}$*-circuits to compute parity.*

What does this mean for the complexity of parity on a CRCW-PRAM ?

Exercise 5.13 *For a given constant d construct a family of* $\mathcal{U}$*-circuits that computes parity in depth d and has the smallest possible size.*

Exercise 5.14 *Show that binary sorting* $\{SORT_n\}$ *is not in* AC^0.

Show that binary merging $\{MERGE_{n,n}^{2n}\}$ *is in* AC^0.

Exercise 5.15 *Let MAJ denote the function family* $\{Th_{2n-1}^{n} \mid n \geq 2\}$. *Show that every function in* SYM_n *is computed by a circuit over the basis* $\{^-\} \cup MAJ$ *of size* $O(n)$ *and depth* 4.

Show that $MAJ \notin \mathrm{AC}^0$.

Exercise 5.16 *Show that binary multiplication is in* $\mathrm{TC}^0 - \mathrm{AC}^0$.

Exercise 5.17 *Let MOD-*3 *be the family of Boolean functions defined by*

$$MOD\text{-}3_n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \textit{if } |\{x_i \mid x_i = 1\}| = 0 \pmod 3 \\ 0 & \textit{otherwise} \end{cases}$$

1. *Find a* GF(3)*-polynomial* p *of degree at most* 2 *satisfying that*

$$p(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1, -1 \end{cases}$$

2. *Show that* $MOD\text{-}3_n$ *has an equivalent* GF(3)*-polynomial of degree at most* 2.

3. *Show that any polynomial size family of circuits over the basis* $\mathcal{U} \cup MOD\text{-}3$ *computing parity must have depth* $\Omega(\frac{\log n}{\log \log n})$

Exercise 5.18 *Let* $p \neq 2$ *be a prime number.* GF(p) *is the finite field with* p *elements. We represent* GF(p) *by the set* $\{-\frac{p-1}{2}, \dots, \frac{p-1}{2}\}$ *with arithmetic modulo* p.

Formulate and prove a variant of lemma 5.15, where the constant 3 *is replaced by* p.

Exercise 5.19 1. *Given a prime number* $p \neq 2$, *show that any polynomial size family of circuits over the basis* $\mathcal{U} \cup MOD\text{-}p$ *computing parity, must have depth* $\Omega(\frac{\log n}{\log \log n})$.

Hint: use the result of exercise 5.18 and redo exercise 5.17 in a generalised version.

2. *What happens if you try to do the first part of this exercise with a number* p *that is not prime, e.g.* $p = 15$ *?*

Literature

1. Andrejev, A. E. (1987) On a Method for Obtaining more than Quadratic Effective Lower Bounds for the Complexity of π-schemes. *Vestnik Moskov. Univ. Mat.* **42**, 70–73 (in Russian); English translation in Moscow Univ. Math. Bull. **42**, 63-66.

2. Beigel, R., Reingold, N. and Spielman, D. (1991) The Perceptron Strikes Back. *Proc. 6th Ann. IEEE Conf. on Structure in Complexity Theory.* 286–291.

3. Blum, N. (1984), A Boolean Function requiring $3n$ Network Size. *Theor. Comp. Sc.* **28**, 337–345.

4. Furst, M., Saxe, J. and Sipser, M. (1984), Parity, Circuits and the Polynomial Time Hierarchy. *Math. Systems Theory* **17**, 13–27.

5. Krapchenko, V. M. (1972), The Complexity of the Realization of Symmetrical Functions by Formulae. *Math. Notes Acad. Sci. USSR*, 70–76.

6. Nečiporuk, È. I. (1966), A Boolean Function. *Sov. Math. Dokl.* **7**, 999-1000.

7. Riordan, J. and Shannon, C. E. (1942), The number of Two-Terminal Series-Parallel Networks. *J. Math. Phys.* **21**, 83–93.

8. Shannon, C. E. (1949), The Synthesis of Two-Terminal Switching Circuits. *Bell Syst. Techn. J.* **28**, 59–98.

9. Valiant, L. G. and Vazirani, V. V. (1986), NP is as Easy as Detecting Unique Solutions. *Theoretical Computer Science* **47**, 85–93.

6 Constructions II

We start by switching attention from Boolean circuits to arithmetic circuits, i.e. circuits where the domain is a semiring, ring or field and the circuit computes a polynomial or rational function using arithmetic gates.

We present the power of subtraction in the form of Strassen's technique for efficient matrix multiplication and we show an upper bound on the additional power that division can provide.

We present a technique for constructing small size and low depth circuits for any polynomial of low degree that is computed by small size circuits. This implies the existence of circuits for efficient parallel computation of determinants.

The above technique works over any semiring, and in particular it can be applied for the parallelisation of Boolean circuits. This leads to nontrivial parallelisation results for specific problems including context free language recognition.

When regarding the Booleans as a semiring, one might expect different results depending on whether AND or OR is identified with the semiring multiplication (both are possible). We prove that in a certain sense it does not matter: The classes SAC^i are closed under complement for $i \geq 1$.

6.1 Arithmetic circuits

An arithmetic circuit is defined analogously to a Boolean circuit, but the operators are $\{+, -, \cdot, /\}$, and the Boolean values $\{0, 1\}$ are replaced by the values of the underlying domain, which is at least a semiring (e.g. $(\mathbf{N}, +, \cdot, 0, 1)$ or $\{0, 1\}$ with AND, OR). If the circuit uses subtraction then the domain must be at least a ring (e.g. $(\mathbf{Z}, +, -, \cdot, 0, 1)$), and if the circuit uses division then the domain must be a field (e.g. $(\mathbf{Q}, +, -, \cdot, /, 0, 1)$). Unless stated otherwise, we assume that rings (semirings) are commutative. An arithmetic circuit over the domain Q with input $x_1, x_2, \ldots, x_n$ computes a polynomial in $Q[x_1, x_2, \ldots, x_n]$ (or a rational function in $Q(x_1, x_2, \ldots, x_n)$, if the circuit uses $/$).

6.2 The power of subtraction

We begin our study of arithmetic circuits by presenting a perhaps surprising result that uses subtraction and therefore the domain must be at least a ring. It is a simple way of multiplying $n \times n$ matrices, which is faster than the obvious

direct method requiring n^3 multiplications and $(n-1) \cdot n^2$ additions.

Note that the multiplication of $n \times n$ matrices consists in the computation of n^2 polynomials of degree 2, i.e. for $n = 2$:

$$\left\{ \begin{array}{cc} c_{11} & c_{12} \\ c_{21} & c_{22} \end{array} \right\} = \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right\} \cdot \left\{ \begin{array}{cc} b_{11} & b_{12} \\ b_{21} & b_{22} \end{array} \right\}$$

where

$$\begin{array}{ll} c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}, & c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21}, & c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{array}$$

Strassen (1969) discovered a computational method using only 7 multiplications but 18 additions, namely

$$\begin{array}{ll} m_1 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22}) & m_2 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22}) \\ m_3 = (a_{11} - a_{21}) \cdot (b_{11} + b_{12}) & \\ m_4 = (a_{11} + a_{12}) \cdot b_{22} & m_5 = a_{11} \cdot (b_{12} - b_{22}) \\ m_6 = a_{22} \cdot (b_{21} - b_{11}) & m_7 = (a_{21} + a_{22}) \cdot b_{11} \end{array}$$

and

$$\begin{array}{ll} c_{11} = m_1 + m_2 - m_4 + m_6 & c_{12} = m_4 + m_5 \\ c_{21} = m_6 + m_7 & c_{22} = m_2 - m_3 + m_5 - m_7 \end{array}$$

Since $n \times n$ matrices may be perceived as 2×2 matrices, where the elements are $n/2 \times n/2$ matrices, and $k \times k$ matrices with multiplication and addition is a ring, the method can be used recursively (the underlying ring need not be commutative)

$$\left\{ \begin{array}{cc} C_{11} & C_{12} \\ C_{21} & C_{22} \end{array} \right\} = \left\{ \begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array} \right\} \cdot \left\{ \begin{array}{cc} B_{11} & B_{12} \\ B_{21} & B_{22} \end{array} \right\}$$

This leads to

Theorem 6.1 *Two $n \times n$ matrices can be multiplied by $O(n^{\log 7}) = O(n^{2,81})$ operations*

Proof. The method sketched above leads to a recurrence for the total number of operations used:

$$A(n) = 7 \cdot A(n/2) + 18 \cdot n^2/4 \quad \text{for } n \geq 2$$

with the solution $A(n) = O(n^{\log 7})$. □

A method for multiplying $n \times n$ matrices using only $O(n^{\alpha})$ operations, where $\alpha < 2,38$ is known (Coppersmith and Winograd, 1990). It is of theoretical interest only, since the involved constants are very large, but Strassen's technic is practical (see exercise 6.1).

6.3 Elimination of division

If some techniques require the domain to be a ring (subtraction is available), one might imagine that even smarter constructions are possible in a field (using the division operation). Though the use of Gaussian elimination for determinant computation provides an example of the latter, it is possible to eliminate all division gates with only moderate increase in circuit size, when computing a polynomial. This result is also due to Strassen (1973).

We will assume that the underlying field Q is infinite.

Definition 6.1 *For N being a $(+,-,\cdot,/)$-circuit, let fun(v) denote the polynomial or rational function computed in the node v.*

Equality (or equivalence) between rational functions $p_1/q_1 = p_2/q_2$ $(q_1, q_2 \neq 0)$ means that $p_1(\underline{x}) \cdot q_2(\underline{x}) = q_1(\underline{x}) \cdot p_2(\underline{x})$ for all $\underline{x}$, where $q_1(\underline{x})q_2(\underline{x}) \neq 0$.

Theorem 6.2 *If $f : Q^n \to Q$ is a rational function computed by a $\{+,-,\cdot,/\}$-circuit N then f is computed by an equivalent circuit N' using only a single division (in the root node r) and $S(N') \leq 4S(N)$.*

Proof. N' is constructed inductively. For each node v in N, N' has two nodes v_1 and v_2 such that $fun(v_1)$ and $fun(v_2)$ are polynomials and $fun(v) = fun(v_1)/fun(v_2)$. For describing the construction, we let v range over constants and variables in addition to gates:

Construction:

v is a constant or variable.	$v_1 \sim v$ and $v_2 \sim 1$
$v \leftarrow u + w.$	$v_1 \leftarrow u_1 \cdot w_2 + u_2 \cdot w_1$ and $v_2 \leftarrow u_2 \cdot w_2$
$v \leftarrow u - w.$	$v_1 \leftarrow u_1 \cdot w_2 - u_2 \cdot w_1$ and $v_2 \leftarrow u_2 \cdot w_2$
$v \leftarrow u \cdot w.$	$v_1 \leftarrow u_1 \cdot w_1$ and $v_2 \leftarrow u_2 \cdot w_2$
$v \leftarrow u/w.$	$v_1 \leftarrow u_1 \cdot w_2$ and $v_2 \leftarrow u_2 \cdot w_1$

The root of N' is a new node r' computing r_1/r_2. For each node v with label $+$ or $-$ we need two extra nodes (in addition to v_1 and v_2) for computing v_1. Since

the construction is only needed, when N contains at least one division gate, we have proven the theorem. □

Definition 6.2 *A $\{+,-,\cdot\}$-circuit N is* homogeneous*, if we for all nodes v in N have that fun(v) is a homogeneous polynomial, i.e. all terms in fun(v) have the same degree, which we denote* $\deg(v)$.

Theorem 6.3 *If f is a polynomial of degree d, computed by a $\{+,-,\cdot\}$-circuit N then there is a homogeneous circuit N' computing $f_0, f_1, \cdots, f_d$ such that f_i is a (homogeneous) polynomial of degree i and $f = \sum_{i=0}^{d} f_i$ and $S(N') \leq (d+1)^2 S(N)$.*

Proof. N' is constructed by induction. For each node v in N there will be $d+1$ nodes $v_0, v_1, \cdots, v_d$ in N' such that *fun*(v_i) is a homogeneous polynomial of degree i and *fun*$(v) = q_v + \sum_{i=0}^{d}$ *fun*(v_i), where q_v is a polynomial with all monomials of degree at least $d+1$. The resulting circuit will have the stated properties. For describing the construction, we let v range over constants and variables in addition to gates:

Construction:

v is a constant.	$v_0 \sim v$ and $v_i \sim 0$ for $i = 1, 2, \cdots, d$.
v is a variable.	$v_1 \sim v, v_i \sim 0$ for $i = 0, 2, 3, \cdots, d$.
$v \leftarrow u + w$.	$v_i \leftarrow u_i + w_i$ for $i = 0, 1, \cdots, d$.
$v \leftarrow u - w$.	$v_i \leftarrow u_i - w_i$ for $i = 0, 1, \cdots, d$.
$v \leftarrow u \cdot w$.	$v_i \leftarrow \sum_{j=0}^{i} u_j \cdot w_{i-j}$ for $i = 0, 1, \cdots, d$.

A multiplication node v needs extra nodes in addition to the nodes $v_0, v_1, \cdots, v_d$ for computing the v_i's. each v_i is computed by a homogeneous circuit of size $(i+1)+i = 2i+1$, i.e. for each node in N we need at most $d+1+\sum_{i=0}^{d} 2i = (d+1)^2$ nodes in N'. □

Lemma 6.4 *Let $p = \sum_{i=0}^{d_p} p_i$ and $q = \sum_{i=0}^{d_q} q_i$ be polynomials over Q, where p_i, q_i are homogeneous polynomials of degree i. Assume that $q_0 \neq 0$ and $p/q = f = \sum_{i=0}^{d} f_i$ $(d = d_p - d_q)$. There is a $\{+,-,\cdot\}$-circuit of size $(d+1)^2$ that computes f from $\{p_0, \cdots, p_{d_p}\}$ and $\{q_0, \cdots, q_{d_q}\}$ alone.*

Proof. Let $q_0' = 1/q_0$. This is a constant in Q. Since $p = q \cdot f$ we have for $i = 0, 1, \cdots, d$:

$$p_i = \sum_{j=0}^{i} q_j \cdot f_{i-j} \text{ and } f_i = q_0'(p_i - \sum_{j=1}^{i} q_j \cdot f_{i-j})$$

using $q_i = 0$ for $i > d_q$. The lemma follows. □

Theorem 6.5 *If f is a polynomial of degree d computed by a $\{+, -, \cdot, /\}$-circuit N then there is $\{+, -, \cdot\}$-circuit N' computing f such that $S(N') = \mathrm{O}(d^2 S(N))$.*

Proof. Let N_1 be the circuit originating from N by the construction in the proof of theorem 6.2. Removing the root node from N_1 leaves a $\{+, -, \cdot\}$-circuit of size at most $4S(N)$ computing p and q such that $f = p/q$.

Since q is not the 0-polynomial there exists an $\underline{\alpha} \in Q^n$ such that $q(\alpha_1, \alpha_2, \cdots, \alpha_n) \neq 0$. Use this $\underline{\alpha}$ to define $p^{(\underline{\alpha})}(\underline{x}) = p(x_1 + \alpha_1, x_2 + \alpha_2, \cdots, x_n + \alpha_n)$ and $q^{(\underline{\alpha})}(\underline{x}) = q(x_1 + \alpha_1, x_2 + \alpha_2, \cdots, x_n + \alpha_n)$. It must hold that $q^{(\underline{\alpha})}(\underline{0}) = q(\alpha_1, \alpha_2, \cdots, \alpha_n) \neq 0$. Let $p^{(\underline{\alpha})} = \sum_{i=0}^{d_p} p_i$ and $q^{(\underline{\alpha})} = \sum_{i=0}^{d_q} q_i$, where p_i and q_i are homogeneous polynomials of degree i. $p^{(\underline{\alpha})}$ and $q^{(\underline{\alpha})}$ is computed by N_1 when the input x_i is replaced by $u_i \leftarrow x_i + \alpha_i$ for $i = 1, 2, \cdots, n$. We homogenise this circuit using the construction from the proof of theorem 6.3. The resulting homogeneous circuit N_2 satisfies

- $S(N_2) \leq (d+1)^2(S(N_1) + n)$
- N_2 computes p_i and q_i for $i = 0, 1, \cdots, d$

Using lemma 6.4 we see that $f^{(\underline{\alpha})}(\underline{x}) = f(x_1 + \alpha_1, x_2 + \alpha_2, \cdots, x_n + \alpha_n) = \sum_{i=0}^{d} f_i$ is computed by a circuit N_3 using N_2 plus $d^2 + 3d + 1$ extra nodes. Since $f(\underline{x}) = f^{(\underline{\alpha})}(x_1 - \alpha_1, x_2 - \alpha_2, \cdots, x_n - \alpha_n)$, we may compute f by a $\{+, -, \cdot\}$-circuit N_4 of size $S(N_3) + n$ when replacing an input x_i of N_3 by a new node $w_i \leftarrow x_i - \alpha_i$. In total, we have that

$$S(N_4) \leq S(N_3) + n \leq S(N_2) + n + (d+1)^2 \leq$$

$$(d+1)^2 S(N_1) + n + (d+1)^2 \leq (d+1)^2 4S(N) + n + (d+1)^2 = \mathrm{O}(d^2 S(N))$$

□

6.4 Parallel computation of low degree functions

When the circuit computing a function is a formula, we have a result similar to the Boolean case:

Theorem 6.6 *If a rational function f is computed by an arithmetic* formula *of size s then f is also computed by an arithmetic circuit of depth* $\mathrm{O}(\log s)$ *and size* $\mathrm{O}(s)$.

Proof. Omitted. □

We are going to prove a theorem that relies on a property of the computed function (low degree polynomial) rather than the computing circuit (formula or not):

Theorem 6.7 *If p is a polynomial of degree d computed by a circuit over $(+, -, \cdot, /)$ of size s and arbitrary depth then p is also computed by a circuit over $(+, -, \cdot)$ of depth $\mathrm{O}((\log s + \log d) \cdot \log d)$ and size $\mathrm{O}((d^2 s)^3)$.*

Note the strength of the theorem, e.g. let $det_{n^2}(\underline{x})$ denote the determinant of a $n \times n$ matrix. det_{n^2} is a polynomial of degree n that can be computed by an arithmetic circuit over $\{+, -, \cdot, /\}$ of size n^3 (Gaussian elimination). The theorem implies that det_{n^2} is also computed by an arithmetic circuit over $\{+, -, \cdot\}$ of polynomial size and depth $\log^2 n$. The result is due to Valiant and Skyum (1981), but we present a simpler proof by Valiant, Skyum, Berkowitz and Rackoff (1983).

Theorem 6.7 is a corollary to theorems 6.3, 6.5, and the following lemma.

Lemma 6.8 *If p is a polynomial of degree d computed by a homogeneous circuit over $(+, \cdot)$ of size s then there exists a circuit of depth $\mathrm{O}(\log s \cdot \log d)$ and size $\mathrm{O}(s^3)$ computing p.*

The proof of this lemma is extensive and we divide it into a number of lemmas. The construction is based on a nonstandard way of evaluating an expression. We begin by introducing some underlying concepts through an example.

Without loss of generality, we may assume that each input occurs once only. Therefore, in the sequel we assume that N is a minimal homogeneous $(+, \cdot)$-circuit where each input occurs only once, and if $v \leftarrow v_1 \cdot v_2$ then $\deg(v_1) \geq \deg(v_2)$.

The circuit E in figure 16 will be our example in the sequel.

If $(u_1, u_2, \cdots, u_k)$ is a path in N directed towards the root then $prod((u_1, u_2, \cdots, u_k))$ denotes the product of the arguments "hanging" at the $\cdot$-nodes of the path, e.g. in E,

$$prod((y, v_2, v_5)) = fun(v_1) \cdot fun(v_4) = (x + y)(x \cdot z + z^2)$$

and

$$prod((v_3, v_4)) = z$$

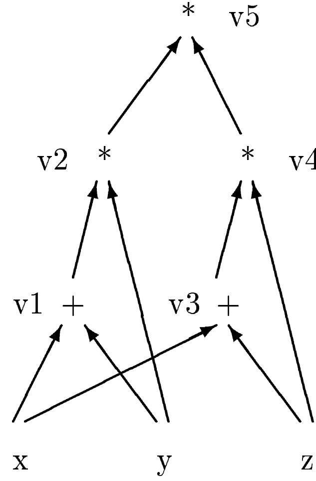

Figure 16: Homogeneous circuit E

Let a $\cdot -$*left path* (towards the root) be a path that does not enter any $\cdot-$node from the right. In E, (v_3, v_4) is such a path, but (y, v_2, v_5) is not.

$con(u, v)$ (short for context of u seen from v) denotes the sum of $prod(s)$ for all $\cdot-$left paths s from u up to v. In E we have

$$con(x, v_5) = \\ prod(x, v_1, v_2, v_5) = y \cdot fun(v_4) = y(x \cdot z + z^2) = con(y, v_5)$$

We make a formal definition of *con* by induction for use in later proofs:

Definition 6.3 *Let u and v be two nodes (or inputs). $con(u, v)$ is defined by*

If $u = v$ then $con(u, v) = 1$

If $v \neq u$ and v is a constant or input then $con(u, v) = 0$

If $v \neq u$ and $v \leftarrow v_1 + v_2$ then $con(u, v) = con(u, v_1) + con(u, v_2)$

If $v \neq u$ and $v \leftarrow v_1 \cdot v_2$ then $con(u, v) = con(u, v_1) \cdot fun(v_2)$

Lemma 6.9 *If $con(u, v) \neq 0$ then it is a homogeneous polynomial of degree* $\deg(v) - \deg(u)$.

Proof. Structural induction in v:

If $u = v$ then $\deg(con(u,v)) = \deg(1) = \deg(v) - \deg(u) = 0$

If $v \leftarrow v_1 + v_2$ then $\deg(con(u,v)) = \deg(con(u,v_1)) = \deg(con(u,v_2)) = \deg(v_1) - \deg(u) = \deg(v_2) - \deg(u) = \deg(v) - \deg(u)$, since $\deg(v) = \deg(v_1) = \deg(v_2)$.

If $v \leftarrow v_1 \cdot v_2$ then $\deg(con(u,v)) = \deg(con(u,v_1)) + \deg(v_2) = \deg(v_1) - \deg(u) + \deg(v_2) = \deg(v) - \deg(u)$. □

Definition 6.4 *If t is a positive integer then V_t denotes the set of nodes*

$$\mathrm{V}_t = \{v \mid \deg(v) > t,\ v \leftarrow v_1 \cdot v_2 \text{ and } \deg(v_1) \leq t\}$$

Lemma 6.10 *Let $0 < t$. If $\deg(u) \leq t < \deg(v)$ then it holds that*

$$fun(v) = \sum_{w \in \mathrm{V}_t} fun(w) \cdot con(w,v)$$

and

$$con(u,v) = \sum_{w \in \mathrm{V}_t} con(u,w) \cdot con(w,v)$$

Proof. Again we use structural induction in v. We consider only $con(u,v)$. The identity for $fun(v)$ is shown in a similar manner.

Since $0 < t < \deg(v)$, v is not a constant or an input variable.

If $v \leftarrow v_1 + v_2$ then

$$\begin{aligned}
&con(u,v) = \\
&\quad con(u,v_1) + con(u,v_2) = \\
&\quad \sum_{w \in \mathrm{V}_t} con(u,w) \cdot con(w,v_1) + \sum_{w \in \mathrm{V}_t} con(u,w) \cdot con(w,v_2) = \\
&\quad \sum_{w \in \mathrm{V}_t} con(u,w) \cdot (con(w,v_1) + con(w,v_2)) = \\
&\quad \sum_{w \in \mathrm{V}_t} con(u,w) \cdot con(w,v)
\end{aligned}$$

Assume that $v \leftarrow v_1 \cdot v_2$ and $v \in \mathrm{V}_t$. If $w \neq v$ and $w \in \mathrm{V}_t$ then $con(w, v) = con(w, v_1) \cdot fun(v_2) = 0$, since $\deg(w) > t \geq \deg(v_1)$ and therefore $con(w, v_1) = 0$. I.e.

$$
\begin{aligned}
& con(u, v) = \\
& \quad con(u, v) \cdot con(v, v) = \\
& \quad \sum_{w \in \mathrm{V}_t} con(u, w) \cdot con(w, v)
\end{aligned}
$$

If $v \leftarrow v_1 \cdot v_2$ and $v \notin \mathrm{V}_t$ then $t < d(v_1)$ and

$$
\begin{aligned}
& con(u, v) = \\
& \quad con(u, v_1) \cdot fun(v_2) = \\
& \quad \left[\sum_{w \in \mathrm{V}_t} con(u, w) \cdot con(w, v_1) \right] fun(v_2) = \\
& \quad \sum_{w \in \mathrm{V}_t} con(u, w) \cdot (con(w, v_1) \cdot fun(v_2)) = \\
& \quad \sum_{w \in \mathrm{V}_t} con(u, w) \cdot con(w, v)
\end{aligned}
$$

□

Proof of lemma 6.8. Let N of size s be a minimal homogeneous circuit computing the polynomial p of degree d. We will use the notation introduced before the proof. Using lemma 6.10 one may construct a low depth circuit computing $fun(v)$ and $con(u, v)$ for all nodes u and v in N.

We construct a circuit divided in $\lceil \log d \rceil$ layers. Each layer i is composed of two sub-layers, namely a *fun*-layer computing $fun(v)$ for the v satisfying $2^{i-1} < \deg(v) \leq 2^i$ and a *con*-layer computing $con(u, v)$ for the nodes u, v satisfying $2^{i-1} < \deg(v) - \deg(u) \leq 2^i$. The *fun*-sub-layer is below (computed before) the *con*-sub-layer.

In the first layer, we compute all $fun(v)$ and $con(u, v)$ of degree at most $1 = 2^0$. Since these polynomials are homogeneous linear forms in the input variables, this can be done in depth $\lceil \log n \rceil + 1$, with the upper $\lceil \log n \rceil$ levels using $+$ only. The size is at most $\mathrm{O}(n \cdot s^2)$.

The $i+1$'st *fun*-sub-layer computes $fun(v)$ as follows:
Let v satisfy $2^i < \deg(v) \leq 2^{i+1}$. For $t = 2^i$ we have by lemma 6.10 that

$$fun(v) = \sum_{w \in \mathrm{V}_t} fun(w) \cdot con(w,v) = \sum_{w \in \mathrm{V}_t} fun(w_1) \cdot fun(w_2) \cdot con(w,v)$$

For $w \in \mathrm{V}_t$ it holds that $\deg(w_2) \leq \deg(w_1) \leq 2^i$ and $\deg(con(w,v)) = \deg(v) - \deg(w) < 2^{i+1} - 2^i$. Therefore $fun(v)$ is a sum of no more than s terms each of which is a product of three factors, all computed in lower layers.
Hence $fun(v)$ may be computed in size $\mathrm{O}(s)$ and depth $\lceil \log s + 2 \rceil$ by a circuit using only $+$ in the upper $\lceil \log s \rceil$ levels.

The $i+1$'st *con*-sub-layer is built on top of *fun*-sub-layer:
Let u, v satisfy $2^i < \deg(v) - \deg(u) = \deg(con(u,v)) \leq 2^{i+1}$. For $t = \deg(u) + 2^i$ we have by lemma 6.10 that

$$con(u,v) = \sum_{w \in \mathrm{V}_t} con(u,w) \cdot con(w,v) = \sum_{w \in \mathrm{V}_t} con(u,w_1) \cdot fun(w_2) \cdot con(w,v)$$

For $w \in \mathrm{V}_t$ we have $\deg(con(u,w_1)) = \deg(w_1) - \deg(u) \leq \deg(u) + 2^i - \deg(u) = 2^i$ and $\deg(con(w,v)) = \deg(v) - \deg(w) < \deg(v) - \deg(u) - 2^i < 2^i$, implying that $con(u,w_1)$ and $con(w,v)$ are both computed in lower layers.
We have no similar upper limit for $\deg(w_2)$. If $\deg(w_2) \leq 2^i$ then $fun(w_2)$ is computed in a lower layer, and if $2^i < \deg(w_2) \leq 2^{i+1}$ then it is computed in the *fun*-sub-layer.
Consider the remaining case of $\deg(w_2) > 2^{i+1}$. In fact we do not need the value of $fun(w_2)$ in this case, because $con(u,w_1) \cdot con(w,v) = 0$. Why does the latter condition hold ? Assume hypothetically that $con(u,w_1) \cdot con(w,v) \neq 0$. Then $\deg(w_1) \geq \deg(u)$ and $\deg(v) \geq \deg(w)$ and we have $\deg(v) \geq \deg(w) = \deg(w_1) + \deg(w_2) > \deg(u) + 2^{i+1}$ in conflict with $\deg(v) - \deg(u) = \deg(con(u,v)) \leq 2^{i+1}$.
Thus $con(u,v)$ is computed in size $\mathrm{O}(s)$ and depth $\lceil \log s + 2 \rceil$ by a circuit using only $+$'s in the upper $\lceil \log s \rceil$ levels.

The total depth of the constructed circuit is $\mathrm{O}(\log d \cdot \log s)$ and the total size is $\mathrm{O}((s+n) \cdot s^2) = \mathrm{O}(s^3)$. □

6.5 Degree bounded Boolean circuits

The proof of theorem 6.7 works over any semiring, and since $\{0,1\}$ with OR and AND is a semiring, the parallelisation result is also true for Boolean circuits, provided all negations are applied directly to inputs.

Assumption 6.1 *In the sequel, we assume that negation gates are applied to inputs only.*

When phrasing the result, we must use the term *degree* with care. In the Boolean algebra where $x \cdot x = x$, polynomials of distinct degrees may be functionally identical, so when we speak about the polynomial computed by a circuit we really mean the (unique) *formal* polynomial computed by that circuit.

Definition 6.5 *The degree of a node in a* $\{+, \cdot, ^{-}\}$*-circuit (or straight line program) is defined inductively by*

- $\deg(0) = \deg(1) = 0$
- *For a variable* x*:* $\deg(x) = \deg(\overline{x}) = 1$
- *If* $v_i \leftarrow v_j + v_k$ *then* $\deg(v_i) = \max\{\deg(v_j), \deg(v_k)\}$.
- *If* $v_i \leftarrow v_j \cdot v_k$ *then* $\deg(v_i) = \deg(v_j) + \deg(v_k)$.

The degree of a circuit, is the degree of the output node.

Definition 6.6 pdC *is the class of Boolean function families* $F = \{F_n\}$*, computed by circuits of polynomial size and polynomial degree.*

We can make a stronger Boolean version of theorem 6.7 when using unbounded fan-in gates

Definition 6.7 *An* $\mathcal{SU}$*-circuit is a circuit over the basis* $\{^{-}, \cdot, +_2, +_3, \ldots\}$*, where negation is used on inputs only. Negations are not counted when calculating the size and depth of an* $\mathcal{SU}$*-circuit.*

SAC^i *is the class of Boolean function families computed by* $\mathcal{SU}$*-circuits of polynomial size and depth* $\mathrm{O}(\log^i n)$

Note that $\mathrm{SAC}^i \subseteq \mathrm{NC}^{i+1}$ (exercise 6.7).

Theorem 6.11 $\mathrm{pdC} = \mathrm{SAC}^1$.

Proof. pdC $\subseteq$ SAC^1 by the construction in the proof of theorem 6.7. SAC^1 $\subseteq$ pdC by exercise 6.8. □

When regarding the Booleans as a semiring, we let AND take the roll of multiplication and we let OR take the roll of addition. This is supported by the notation we are using, but what happens if we reverse the rolls of AND and OR? – Essentially Nothing: the class pdC is invariant under interchange of $+$ and $\cdot$ in the definition of degree. We prove a more general result for which we need the concept of co-classes.

Definition 6.8 *For a class $\mathcal{C}$ of Boolean function families the class co-$\mathcal{C}$ consists of the families $F = \{F_n\}$ satisfying that $\overline{F} = \{\overline{F}_n\}$ is in $\mathcal{C}$.*

The definitions of pdC and SAC^i do not indicate that these classes are closed under complement, but Borodin, Cook, Dymond, Ruzzo and Tompa (1989) showed that SAC^i = *co*-SAC^i (and therefore also pdC = *co*-pdC). The key to this result is the proper use of a *counter*, and the authors state in the original paper that they were inspired by the proof of Immerman (1988) that nondeterministic space is closed under complement. The proof has also some similarity to the construction in exercise 3.30.

Theorem 6.12 *If $f \in \mathrm{B}_n$ is computed by an $\mathcal{SU}$-circuit of size $s \geq n$ and depth d then $\overline{f}$ is computed by an $\mathcal{SU}$-circuit of size $O(s^3 d \log s)$ and depth $O(d + \log s)$.*

Proof. We are going to modify N in 4 steps, ending up with a circuit N_4 that computes the negated value of N's output in addition to computing all the subresults computed by N.

We start by constructing a layered circuit N_1, where a gate in layer t takes input from layer $t-1$ only.

The circuit N_2 uses the division into layers for computing a conditional negation of every value computed in N_1. If v is a $\sum$-gate in layer t and we know the total number of computed 1-values in layer $t-1$ (denote this number by P_{t-1}) then the negation of v is computed by the function $Th^{P_{t-1}}$ applied to those gates in layer $t-1$ that are *not* inputs to v. Therefore the correct computation of the negations is conditional to knowing the number of 1's computed in each layer.

The P_t's are computed inductively by the circuits N_3, and N_4 computes the negated output.

Let us describe the construction of the circuit N_1.

Let $J = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$, let G be the set of $\cdot$-gates and $\sum$-gates in N and let $V = J \cup G$, i.e. $|V| = 2n + s = O(s)$. For $v \in G$ let $I(v) \subseteq V$ denote the set of inputs to v.

N_1 has d layers of gates ($|V|$ gates in each layer), and in addition a bottom layer with $|V|$ input/neg-input/constants.

For each $v \in V$ and each layer $t = 1, 2, \ldots, d$, N_1 has a gate v_t, and for each $v \in V$, N_1 has an input/neg-input/constant v_0.

If $v \leftarrow u \cdot w$ then $v_0 = 0$ and $v_t \leftarrow u_{t-1} \cdot w_{t-1}$ for $t \geq 1$.

If $v \leftarrow \sum_{u \in I(v)} u$ then $v_0 = 0$ and $v_t \leftarrow \sum_{u \in I(v)} u_{t-1}$ for $t \geq 1$.

If $v \in J$ then $v_0 = v$ and for $t \geq 1$, v_t computes the same value as v_{t-1}, which could be realised by letting $v_t \leftarrow v_{t-1} \cdot v_{t-1}$.

Let $V_t = \{v_t \mid v \in V\}$. N_1 satisfies the following:

1. *layering:* A gate in V_t takes input from V_{t-1} only, and $|V_t| = |V|$.
2. *equivalence to N:* If a gate $v \in G$ has depth l then $val(v_t) = val(v)$ for $t \geq l$. (Use induction in l for a proof).
3. *depth:* $D(N_1) = d$
4. *size* $S(N_1) = |V| \cdot d = O(sd)$

The circuit N_2 will contain N_1 as a sub-circuit and in addition N_2 has gates (inputs/constants) $v_{t,m}$ where $m = 0, 1, 2, \ldots, |V|$ for each gate (input/constant) v_t in N_1. We intend $v_{t,m}$ to compute v_t negated when m is the precise number of 1-values computed in the prior layer (V_{t-1}), i.e.

$$val(v_{t,m}) = \begin{cases} \overline{val(v_t)} & \text{if } m = P_{t-1} \\ \text{arbitrary value} & \text{otherwise} \end{cases}$$

where

$$P_{t-1} = |\{u_{t-1} \mid u_{t-1} \in V_{t-1} \ \text{ and } \ val(u_{t-1}) = 1\}|.$$

The $v_{t,m}$'s are constructed:

$v_{0,m} = \overline{v_0}$ for all m.

If $v_t \leftarrow u_{t-1} \cdot w_{t-1}$ then $v_{t,m} \leftarrow Th^{m-1}(V_{t-1} - \{u_{t-1}, w_{t-1}\})$ for all m.

If $v_t \leftarrow \sum_{u \in I(v)} u_{t-1}$ then $v_{t,m} \leftarrow Th^{m}(V_{t-1} - \{u_{t-1} \mid u \in I(v)\})$ for all m.

Each Th-function is computed by an $\mathcal{SU}$-circuit of size $O(s \log s)$ and depth $O(\log s)$ (A monotone circuit over the basis $\{+, \cdot\}$ is also an $\mathcal{SU}$-circuit, and monotone circuits are constructed from sorting networks).

The circuit N_2 contains the circuit N_1 and satisfies

1. *conditional negation:* $val(v_{t,m}) = \overline{val(v_t)}$ for $m = P_{t-1}$.
2. *depth:* $D(N_2) = O(d + \log s)$. (All inputs to a Th-function are gates/inputs/ constants from N_1).
3. *size:* $S(N_2) = O(S(N_1) \cdot |V| \cdot s \log s) = O(s^3 d \log s)$.

The circuit N_3 computes the P_t-values in a positional representation, i.e. N_3 contains gates/constants $c_{t,m}$ for $t = 0, 1, \ldots, d$ and $m = 0, 1, \ldots, |V|$ such that

$$val(c_{t,m}) = \begin{cases} 1 & \text{if } m = P_t \\ 0 & \text{otherwise} \end{cases}$$

We have that $P_0 = n$, since precisely n of $\{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ have value 1. We therefore let $c_{0,n} = 1$ and $c_{0,m} = 0$ for $m \neq n$.

For $t \geq 1$ we may express $P_t = m$ in the following way:

i) $P_t \geq m$, i.e. at least m of the v_t's compute the value 1, and

ii) $P_t \leq m$, i.e. if $P_{t-1} = r$ then at least $|V| - m$ of the $v_{t,r}$'s compute the value 1.

This implies the correctness of the following computation of $c_{t,m}$ (by induction in t):

$$c_{t,m} \leftarrow Th^{m}(V_t) \cdot \sum_{r=0}^{|V|} c_{t-1,r} \cdot Th^{|V|-m}(\{v_{t,r} \mid v \in V\})$$

The Th-functions are computed as in N_2.

The circuit N_3 contains N_2 and satisfies

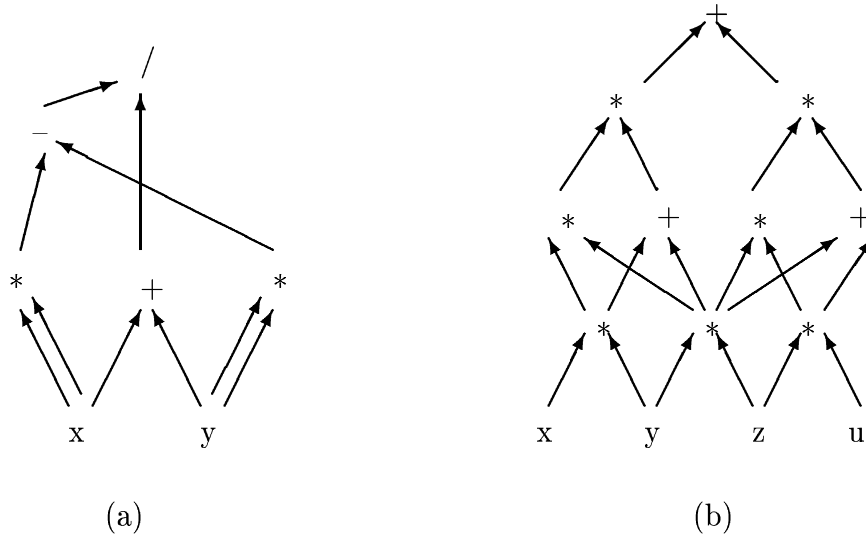

Figure 17:

1. *correct counting:* $c_{t,m} = 1$ if and only if the v_t's have m 1-values.
2. *depth:* $D(N_3) = O(D(N_2) + d + \log s) = O(d + \log s)$ (All inputs to the new *Th*-functions are gates/inputs/constants from N_2).
3. *size:* $S(N_3) = O(S(N_2) + d|V|^2 s \log s) = O(s^3 d \log s)$

N_4 contains a special output gate v'. If v is the output gate in N then $v' \leftarrow \sum_{m=0}^{|V|} c_{d-1,m} \cdot v_{d,m}$ satisfies that $val(v') = \overline{val(v)}$. □

Corollary 6.13 $\mathrm{SAC}^i = co\text{-}\mathrm{SAC}^i$ *for* $i \geq 1$.

Exercises

Exercise 6.1 *1) Decide for which* m, $2^m \times 2^m$ *matrix multiplication based on Strassen's technique uses fewer operations than the standard technique.*

2) What method should be used in practice, if $+$ *and* $\cdot$ *are equally fast ?*

Exercise 6.2 *Assume that we base matrix multiplication on a method for multiplication of* 3×3 *matrices. How many multiplications is allowed if we want to beat Strassen based matrix multiplication (asymptotically)?*

Exercise 6.3 *Use the method of theorem 6.5 to find the polynomial (of degree 1) computed by the circuit in figure 17 (a)*

Exercise 6.4 *State and prove an arithmetic version of Theorem 2.5.*

Exercise 6.5 *Compute $con(u, v)$ for all nodes in the circuit of figure 17 (b), where* $\deg(v) = 6$ *and* $u \in \mathrm{V}_3$

Exercise 6.6 *Show that* $\mathcal{C} \subseteq \mathcal{D} \Rightarrow co\text{-}\mathcal{C} \subseteq co\text{-}\mathcal{D}$

Exercise 6.7 *Show that* $\mathrm{SAC}^i \subseteq \mathrm{NC}^{i+1}$.

Exercise 6.8 *Show that* $\mathrm{SAC}^1 \subseteq \mathrm{pdC}$.

Exercise 6.9 *1) Show that PATH (Example 1.3) is in* pdC.

2) Find some graph problems in pdC.

Exercise 6.10 *Show that recognition of a context free language is in* pdC.

Hint: use the following algorithm (based on dynamic programming).

Let $G = (N, \Sigma, P, S)$ be a context free grammar in Chomsky normal form.
N is the set of nonterminals, Σ is the set of input symbols, S is the start symbol and P is the set of production rules being in the form $A \to BC$ or $A \to a$ ($A, B, C \in N$, $a \in \Sigma$).

For a string $w = a_1 a_2 \cdots a_n$ the following algorithm decides whether $w \in L(G)$. The M_{ij}'s are sets of nonterminals satisfying $(A \in M_{ij}) \Leftrightarrow (A \Rightarrow^* a_i a_{i+1} \cdots a_j)$.

```
w ∈ L(G)
  for i := 1 to n do
     M_ii ← {A ∈ N | A → a_i}
  od
  for i := 1 to n do
     for j := i + 1 to n do
        M_ij ← ∅
     od
  od
```

```
for k := 1 to n − 1 do
   for i := 1 to n − k do
      for j := i to i + k − 1 do
         M_{i,i+k} ← {A ∈ N | A → BC, B ∈ M_{ij}, C ∈ M_{j+1,i+k}} ∪ M_{i,i+k}
      od
   od
od
output S ∈ M_{1n}
```

Exercise 6.11 *Show that the problem of computing the determinant of an integer matrix is in* NC^2. *Improve your upperbound by showing that the problem is in* TC^1.

Exercise 6.12 *Show that the size of the circuit constructed in the proof of theorem 6.12 can be reduced to* $O(s^2 d \log s)$.

Literature

1. Bürgisser, P., Clausen, M. and Shokrollahi, M.A. (1997), *Algebraic Complexity Theory.* Volume 315 of Grundlehren der Mathematischen Wissenschaften, Springer.

2. Borodin, A., Cook, S. A., Dymond, P. W., Ruzzo, W. L. and Tompa, M. L. (1989), Two Applications of inductive Counting for Complementation Problems. *SIAM J. Comput.* **18**, 559–578.

3. Coppersmith, D. and Winograd, S. (1990) Matrix Multiplication via Arithmetic Progressions. *J. Symbolic Computation* **9**, 251–280.

4. Immerman, N. (1988), Nondeterministic Space is Closed Under Complementation. *SIAM J. Comput.* **17**, 935–938.

5. Strassen, V. (1969), Guassian Elimination is not Optimal. *Numer. Math.* **13**, 354–356.

6. Strassen, V. (1973), Vermeidung von Divisionen. *J. Reine Angew. Math.* **264**, 184–202.

7. Valiant L. G. and Skyum, S. (1981), Fast Parallel Computation of Polynomials using few Processors. *Proc. from 10th MFCS.*

8. Valiant L. G., Skyum, S., Berkowitz, S. and Rackoff, C. (1983), Fast Parallel Computation of Polynomials using few Processors. *SIAM J. Comput.* **12**, 641–644.

7 Lower Bounds II

We start this section by introducing communication complexity, which is a rich subject on its own in addition to providing a technique for proving lower bounds on circuit depth. We prove a lower bound on the probabilistic communication complexity of deciding disjointness and apply this result to prove a linear depth lower bound for monotone solutions to a matching problem (which have low depth general solutions).

Finally, we present Razborov's original super-polynomial size lower bound for monotone solutions to the NP-complete k-clique problem.

7.1 Communication Complexity

Just as time, space and hardware size are fundamental resources, when computing a function, one may consider *communication* a resource, i.e. if the input is distributed between several agents, how much communication is needed for the agents to agree on a function value?

Results from communication complexity theory applies directly to distributed systems. There are also less obvious applications, e.g. trade off results in VLSI theory (see exercise 7.2), lower bounds on depth (see section 7.3), and most recently lower bounds on data type implementation.

The study of communication complexity was initiated by Yao (1979).

Definition 7.1 *Let X, Y, Z be finite sets. A deterministic communication protocol A over X, Y, Z specifies the exchange of information between two players I and II who initially receive inputs $x \in X$ and $y \in Y$ respectively, and finally agree on a value $A(x, y) \in Z$. Let $C_A(x, y)$ denote the number of bits exchanged between the two players using protocol A on input x, y.*

A computes the function $f : X \times Y \mapsto Z$ defined by $f(x, y) = A(x, y)$ for all $(x, y) \in X \times Y$, and we let $C_A(f) = \max_{(x,y)\in X\times Y} C_A(x, y)$. The communication complexity of f is

$$C(f) = \min\{C_A(f) \mid A \text{ computes } f\}$$

More generally, A is said to compute a (non-unique) relation $R \subseteq X \times Y \times Z$, if $(x, y, A(x, y)) \in R$ for all $(x, y) \in X \times Y$. $C_A(R)$ and $C(R)$ is defined in the obvious way.

Definition 7.2 *For a Boolean function $f \in \mathrm{B}_{2n}$, we let $C_{X \leftrightarrow Y}(f)$ denote the communication complexity of f with respect to a specific division $X \subseteq \{x_1, \ldots, x_{2n}\}$ and $Y = \{x_1, \ldots, x_{2n}\} - X$.*

We have a default division: $C(f) = C_{\{x_1,\ldots,x_n\} \leftrightarrow \{x_{n+1},\ldots,x_{2n}\}}(f)$.

We also need $C_{best}(f) = \min\{ C_{X \leftrightarrow Y}(f) \mid |X| = |Y| = n,$ and $X \cup Y = \{x_1, \ldots, x_{2n}\} \}$

Proposition 7.1 *For all $f \in \mathrm{B}_{2n}$, $C(f) \leq n + 1$.*

Proof. We may assume that player I and II initially have inputs $\underline{x} \in \{0,1\}^n$ and $\underline{y} \in \{0,1\}^n$ respectively. Player I sends all of $\underline{x}$ to player II that computes $f(\underline{x}, \underline{y})$ and returns the result bit to player I. □

Example 7.1 *Examples of problems having nontrivial deterministic communication complexity – in class only.*

Definition 7.3 *The communication matrix M_f of $f \in \mathrm{B}_{2n}$ is the $2^n \times 2^n$ 0,1-matrix $\{m_{ij}\}$ defined by*

$$m_{[\underline{x}],[\underline{y}]} = f(\underline{x}, \underline{y}).$$

Proposition 7.2 $C(f) \geq \log \operatorname{rank}(M_f)$.

Proof. A communication protocol A for computing f specifies who sends the first bit, say player I does. This bit is determined by the input $\underline{x}$ to player I. In other words the protocol divides the rows of M_f in two classes and the bit sent tells player II in which of the two classes $\underline{x}$ belongs. The rows in this class forms a submatrix M_1 of M_f. The ith bit exchanged between player I and player II chooses either a subset of the rows or the columns in the previous matrix M_{i-1} resulting in a submatrix M_i of M_{i-1}.

When the protocol ends, say after the exchange of k bits, the rows of M_k correspond to the set of $\underline{x}$'s consistent with player II's knowledge and similarly the columns of M_k correspond to the set of $\underline{y}$'s consistent with player I's knowledge. Since player I knows that player II knows the value of $f(\underline{x}, \underline{y})$, each column of M_k must be all 1's or all 0's. By a symmetric argument, each row of M_k must also be all 1's or all 0's, i.e. M_k is monochromatic (all 1's or all 0's), and therefore $\operatorname{rank}(M_k) \leq 1$.

There exists an input such that $\text{rank}(M_i) \geq \frac{1}{2}\text{rank}(M_{i-1})$ for all i, and therefore A uses at least $\log \text{rank}(M_f)$ bits of communication in the worst case. □

Definition 7.4 *Define* $EQUAL_{2n}, DISJ_{2n} \in \mathrm{B}_{2n}$ *by*

$$EQUAL(\underline{x}, \underline{y}) = \begin{cases} 1, & \text{if } \underline{x} = \underline{y} \\ 0, & \text{otherwise} \end{cases}$$

$$DISJ(\underline{x}, \underline{y}) = \begin{cases} 1, & \text{if } x_i y_i = 0 \text{ for all } i \\ 0, & \text{otherwise} \end{cases}$$

Proposition 7.3 $n \leq C(EQUAL), C(DISJ) \leq n + 1$

Proof. The communication matrices for both problems have rank 2^n (exercise 7.3). Thus the lower bound follows from proposition 7.2. The upper bound follows from the trivial communication protocol, proposition 7.1. □

Less communication may suffice when coin tossing is allowed:

Definition 7.5 *A probabilistic communication protocol A is a probability distribution over deterministic protocols. $C_A(x, y)$ is the expected number of bits exchanged by the two players on an input x, y.*

A probabilistic protocol ϵ-computes a function f, if for all (x, y) in the domain of f, $\Pr(A(x,y) \neq f(x,y)) \leq \epsilon$ and we let $C_A(f) = \max_{x,y} C_A(x,y)$. The ϵ-error probabilistic communication complexity of f is

$$C_\epsilon(f) = \min\{C_A(f) \mid \text{ protocol } A \ \epsilon\text{-computes } f\}.$$

For a relation R, $C_\epsilon(R)$ is defined similarly.

Proposition 7.4 $C_\epsilon(EQUAL_{2n}) \leq \log n + \log(\epsilon^{-1}) + O(1)$

Proof. For each prime p, we have a deterministic protocol A_p that works as follows: player I sends the value $[\underline{x}] \bmod p$ ($\log p$ bits) to player II. If $[\underline{x}] \bmod p = [\underline{y}] \bmod p$ then the result of the protocol is 1 and otherwise 0. Player II notifies player I of the result, using a single bit.

If $\underline{x} = \underline{y}$ then $A_p(\underline{x}, \underline{y}) = EQUAL_{2n}(\underline{x}, \underline{y})$ for all p. If $\underline{x} \neq \underline{y}$ then $A_p(\underline{x}, \underline{y}) = EQUAL_{2n}(\underline{x}, \underline{y})$ except when p divides $|[\underline{x}] - [\underline{y}]| \leq 2^n$. Define Chebyshev's ϑ-function by

$$\vartheta(x) = \sum_{p \leq x} \ln p$$

A version of the prime number theorem states that $\lim_{x\to\infty} \frac{\vartheta(x)}{x} = 1$ (see Apostol, *Introduction to Analytic Number Theory*, Springer-Verlag, 1976). In particular this means that there is a constant c such that $\vartheta(cx) \geq x$ for all $x \geq 2$. Let $m = \frac{c \ln 2}{\epsilon} n$. The probabilistic protocol A, will use the protocol A_p with probability $\frac{\ln p}{\vartheta(m)}$ for $p \leq m$ and

$$\Pr(A(\underline{x}, \underline{y}) \neq EQUAL(\underline{x}, \underline{y})) \leq$$

$$\Pr(p \text{ divides } [\underline{x}] - [\underline{y}] \;\mid\; \underline{x} \neq \underline{y}) \leq$$

$$\frac{\ln(2^n)}{\vartheta(m)} \leq \epsilon$$

The number of bits transfered in the protocol is

$$\log p + 1 \leq \log m + 1 = \log n + \log(\epsilon^{-1}) + O(1)$$

□

7.2 Lower bound on probabilistic communication complexity

The last result has no analogue for the disjointness problem. In fact it holds that

Proposition 7.5 (Kalyanasundaram and Schnitger, 1987)

$$C_{\frac{1}{3}}(DISJ) = \Omega(n)$$

Proof. Omitted. A relatively short proof is given by Razborov (1992). □

We give the details of a weaker result with a simpler proof, namely $C_{\frac{1}{3}}(DISJ) = \Omega(\sqrt{n})$.

We need some definitions and lemmas.

Lemma 7.6 *Let $f \in \mathrm{B}$ and let ϵ, δ be fixed constants such that $0 < \delta \leq \epsilon < \frac{1}{2}$. It holds that $C_\delta(f) = O(C_\epsilon(f))$.*

Proof. see exercise 7.6. □

Notation 7.1 *We will when convenient regard $\underline{x}$ as the subset of $\{1, 2, \ldots, n\}$ defined by $\underline{x} = \{i \mid x_i = 1\}$, and accordingly we use set notation such as*

$$\underline{x} \cap \underline{y} = \underline{z} \quad \textit{where} \quad z_i = x_i y_i$$

$$\#\underline{x} = |\{i \mid x_i = 1\}|$$

Definition 7.6 *Let $Q = \{\underline{x} \in \{0,1\}^n \mid \#\underline{x} = \sqrt{n}\}$ and define DISJ-Q = $DISJ|_{Q \times Q}$.*

Lemma 7.7 *$C_\epsilon(DISJ) \geq C_\epsilon(DISJ\text{-}Q)$ for all ϵ.*

Proof. Any probabilistic protocol for *DISJ* will also work for *DISJ-Q*. □

Definition 7.7 *For $f : X \times Y \mapsto Z$ and $\epsilon > 0$ let $d_\epsilon(f)$ be the minimum of $C_A(f)$, where the minimum is taken over all possible deterministic protocols A that compute f correctly on all inputs except for a fraction ϵ.*

Lemma 7.8 *$2C_\epsilon(f) \geq d_{2\epsilon}(f)$.*

Proof. See exercise 7.7. □

Definition 7.8 *Let M denote the $|Q| \times |Q|$ communication matrix for DISJ-Q. A submatrix $R = F \times G$ of M is an ϵ-error 1-rectangle if $\underline{x} \cap \underline{y} = \emptyset$ except for a fraction ϵ of all $(\underline{x}, \underline{y}) \in F \times G$.*

Lemma 7.9 *There exists $\epsilon_0 > 0$ and $c_0 > 0$ such that for all $0 < \epsilon < \epsilon_0$ and $0 < c < c_0$ the following holds.*

Let $R = F \times G$ be an ϵ-error 1-rectangle.

Let $F_1 = \{x \in F \mid \#\{y \in G | \underline{x} \cap \underline{y} \neq \emptyset\} \leq 2\epsilon|G| \}$.

(i) If $|F_1| \geq \frac{1}{2}|Q|2^{-c\sqrt{n}}$ then we may choose $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_{\sqrt{n}/3} \in F_1$ such that

$$|\bigcup_{i=1}^{\sqrt{n}/3} \underline{x}_i| \geq \frac{n}{6}$$

provided n is sufficiently large.

(ii) $|R| \leq |Q|^2 2^{-c\sqrt{n}}$ provided n is sufficiently large.

Proof. part (i): It suffices to choose $\underline{x}_l$ such that $|\underline{x}_l - \cup_{i=1}^{l-1} \underline{x}_i| \geq \frac{\sqrt{n}}{2}$. Since $|\cup_{i=1}^{l-1} \underline{x}_i| < \frac{\sqrt{n}}{3}\sqrt{n} = \frac{n}{3}$, the number of choices for $\underline{x}_l$ is at least

$$|F_1| - \sum_{i=\frac{\sqrt{n}}{2}+1}^{\sqrt{n}} \binom{\frac{n}{3}}{i} \binom{\frac{2n}{3}}{\sqrt{n}-i} \geq$$

$$\frac{1}{2}|Q|2^{-c\sqrt{n}} - \frac{\sqrt{n}}{2} \binom{\frac{n}{3}}{\frac{\sqrt{n}}{2}} \binom{\frac{2n}{3}}{\frac{\sqrt{n}}{2}} \geq$$

$$\frac{1}{2}\binom{n}{\sqrt{n}} 2^{-c\sqrt{n}} - \frac{n\sqrt{n}}{2} (\frac{1}{3})^{\frac{\sqrt{n}}{2}} (\frac{2}{3})^{\frac{\sqrt{n}}{2}} \binom{\sqrt{n}}{\frac{\sqrt{n}}{2}} \binom{n}{\sqrt{n}} \geq$$

$$\frac{1}{2}\binom{n}{\sqrt{n}} 2^{-c\sqrt{n}} - \frac{n\sqrt{n}}{2} (\frac{2\sqrt{2}}{3})^{\sqrt{n}} \binom{n}{\sqrt{n}}$$

which is strictly positive for n sufficiently large provided that $2^{-c} > \frac{2\sqrt{2}}{3}$, i.e. $c < \log \frac{3}{2\sqrt{2}} \approx 0.09$.

part (ii): If $|F| \leq |Q|2^{-c\sqrt{n}}$ then $|R| = |F \times G| \leq |Q|^2 2^{-c\sqrt{n}}$, and therefore we will assume that $|F| \geq |Q|2^{-c\sqrt{n}}$. By a counting argument, we also have that

$|F_1| \geq |F|/2 \geq \frac{1}{2}|Q|2^{-c\sqrt{n}}$, and therefore we can use part (i) of the lemma. In the following let $\underline{x}_1, \ldots, \underline{x}_{\sqrt{n}/3}$ be given (and fixed) such that $|\cup_{i=1}^{\sqrt{n}/3} \underline{x}_i| \geq \frac{n}{6}$. Let

$$G_1 = \{\underline{y} \in G \ \mid \ \#\{i | \underline{x}_i \cap \underline{y} \neq \emptyset, 1 \leq i \leq \frac{\sqrt{n}}{3}\} \leq 4\epsilon \frac{\sqrt{n}}{3} \}.$$

By a counting argument, we also have that $|G_1| \geq |G|/2$. For S being some subset of the x_i's with $|S| = 4\epsilon\frac{\sqrt{n}}{3}$, let G_1^S be those $y \in G_1$ that intersect no x_i not contained in S. For a fixed S there must be at least $\frac{n}{6} - 4\epsilon\frac{\sqrt{n}}{3}\sqrt{n} \geq \frac{n}{9}$ (for $\epsilon \leq \frac{1}{24}$) elements in $\{1, \ldots, n\}$ that are not included in any $y \in G_1^S$, and therefore $|G_1^S| \leq \binom{\frac{8n}{9}}{\sqrt{n}}$. Combined we get

$$|G| \leq 2|G_1| \leq 2|\bigcup_S G_1^S| \leq 2\binom{\frac{\sqrt{n}}{3}}{4\epsilon\frac{\sqrt{n}}{3}}\binom{\frac{8n}{9}}{\sqrt{n}}$$

By exercise 7.8 this expression is $\leq 2^{-c\sqrt{n}}|Q|$ for ϵ, c sufficiently small and n sufficiently large. □

Theorem 7.10 (Babai, Frankl and Simon, 1986)
$C_{\frac{1}{3}}(DISJ_{2n}) = \Omega(\sqrt{n})$

Proof. By lemmas 7.6, 7.7 and 7.8 it suffices to prove that $d_\epsilon(DISJ\text{-}Q_{2n}) = \Omega(\sqrt{n})$ for some $\epsilon < \frac{1}{2}$.

Remember that in the communication matrix M for $DISJ\text{-}Q_{2n}$ a 1 corresponds to an input pair $\underline{x}, \underline{y}$ with $\underline{x} \cap \underline{y} = \emptyset$, and a 0 corresponds to an input pair $\underline{x}, \underline{y}$ with $\underline{x} \cap \underline{y} \neq \emptyset$

Given a deterministic communication protocol A, all inputs $\underline{x}, \underline{y}$ that lead to the same communication (i.e. same bit sequence) between the two players as a given input $(\underline{x}_0, \underline{y}_0)$ corresponds to a submatrix $M_{(\underline{x}_0, \underline{y}_0)}$ of M. $M_{(\underline{x}_0, \underline{y}_0)}$ may contain both 0's and 1's, but the protocol will give the same answer to all the corresponding inputs, either always 0 or always 1.

The fraction of 1's in the communication matrix M for $DISJ\text{-}Q_{2n}$ is

$$\binom{n - \sqrt{n}}{\sqrt{n}} / \binom{n}{\sqrt{n}} \geq (\frac{n - 2\sqrt{n} + 1}{n - \sqrt{n} + 1})^{\sqrt{n}} \geq (1 - \frac{1}{\sqrt{n} - 1})^{\sqrt{n}}$$

which is greater than $\frac{1}{3}$ for n sufficiently large.

If A is a deterministic protocol erring on no more than a fraction ϵ of inputs then at least $(\frac{1}{3}-\epsilon)|Q|^2$ of all the 1's in M must also lead to the answer 1 by A.

If more than $\frac{1}{2}(\frac{1}{3}-\epsilon)|Q|^2$ of the 1's in M lead to 1-answers based on submatrices with more than a fraction 6ϵ of 0's, then the total number of 0's leading to a wrong 1-answer is at least $\frac{1}{2}(\frac{1}{3}-\epsilon)\frac{6\epsilon}{1-6\epsilon}|Q|^2 > \epsilon|Q|^2$. This is a contradiction, so at least $\frac{1}{2}(\frac{1}{3}-\epsilon)|Q|^2$ of the 1's in M lead to a 1-answer based on submatrices that are 6ϵ-error 1-rectangles. Each such submatrix has size $\leq |Q|^2 2^{-c\sqrt{n}}$ for some $c > 0$ (if ϵ is sufficiently small) according to lemma 7.9(ii). Hence there are at least

$$\frac{\frac{1}{2}(\frac{1}{3}-\epsilon)|Q|^2}{|Q|^2 2^{-c\sqrt{n}}} = 2^{c\sqrt{n}}\frac{1}{2}(\frac{1}{3}-\epsilon)$$

distinct submatrices $M_{(\underline{x},\underline{y})}$. Therefore the protocol must take at least $\log(2^{c\sqrt{n}}\frac{1}{2}(\frac{1}{3}-\epsilon)) = \Omega(\sqrt{n})$ steps. □

7.3 Lower bound on monotone depth

Definition 7.9 *$MATCH_{\binom{n}{2}} \in \mathrm{B}_{\binom{n}{2}}$ is defined by*

$$MATCH_{\binom{n}{2}}(\underline{x}) = \begin{cases} 1, & \text{if } G[\underline{x}] \text{ has a matching of size } \frac{n}{3} \\ 0, & \text{otherwise} \end{cases}$$

Mulmuley, Vazirani and Vazirani (1987) constructs a randomised parallel algorithm for *MATCH* using $n^{O(1)}$ processors and $\log^{O(1)} n$ time with error probability $\leq \frac{1}{4}$. By the techniques of section 4.6 this means that $MATCH \in \mathrm{NC}^k$. See also exercise 7.14.

MATCH is a monotone problem, and one might expect that a good parallel *and* monotone solution also exists. However, Raz and Wigderson (1992) proved that $D_m(MATCH) = \Omega(n)$.

Definition 7.10 *Let $f \in \mathrm{MON}_n$.*

$\underline{x}$ is a minterm for f if $f(\underline{y}) = 1$ for all $\underline{y} \geq \underline{x}$ and $f(y) = 0$ for all $\underline{y} < \underline{x}$.

$\underline{x}$ is a maxterm for f if $f(\underline{y}) = 1$ for all $\underline{y} > \underline{x}$ and $f(y) = 0$ for all $\underline{y} \leq \underline{x}$.

Let $MIN(f), MAX(f) \subseteq \{0,1\}^n$ be the set of minterms and maxterms, respectively, for f. Define the relation $R(f) \subseteq MIN(f) \times MAX(f) \times \{1,2,\ldots,n\}$ by

$$(\underline{x}, \underline{y}, i) \in R(f) \quad \text{iff} \quad x_i = 1 \quad \text{and} \quad y_i = 0$$

Lemma 7.11 *$D_m(f) \geq C(R(f))$ for all $f \in$ MON.*

Proof. A $(+,\cdot)$-formula F of depth d is known to both players. Player I has an input $\underline{x}$ such that $F(\underline{x}) = 1$ and player II has an input $\underline{y}$ such that $F(\underline{y}) = 0$. If $F = F_0 \cdot F_1$ then one of $F_0(\underline{y})$ and $F_1(\underline{y})$ must be 0. Assume that $F_i(\underline{y}) = 0$ then player II sends the bit i to player I and since $F_i(\underline{x}) = 1$, the protocol may continue recursively on F_i that has depth $\leq d-1$. In case $F = F_0 + F_1$ the roles of the two players are reversed. After at most d steps the formula is reduced to a single variable, the index of which is known to both players. □

Definition 7.11 *Let P_n be the set of all $\frac{n}{3}$-matchings over n vertices (Each member of P_n is a graph with $\frac{n}{3}$ disjoint edges). Let Q_n be the set of all $(\frac{n}{3}-1)$-subsets of n vertices (Each member of Q_n is $\frac{n}{3}-1$ vertices). Let E_n be all edges in the complete graph on n vertices. Define $M\hat{A}TCH_n \subset P_n \times Q_n \times E_n$ by*

$$(p, q, e) \in M\hat{A}TCH_n \quad \text{iff} \quad e \in p \quad \text{and} \quad e \cap q = \emptyset.$$

Lemma 7.12 *$D_m(MATCH_{\binom{n}{2}}) \geq C(M\hat{A}TCH_n)$*

Proof. This follows from lemma 7.11, since $p \in P_n$ is a minterm for $MATCH$ and $q \in Q_n$ induces a maxterm r (all the edges in $q \times \{1,2,\ldots,n\}$). For any edge e, we have that $e \in p$ and $e \cap q = \emptyset$ if and only if $e \in p - r$. □

Definition 7.12 *Define $DIST_{\frac{2n}{3}} : \{a,b,c\}^{\frac{n}{3}} \times \{a,b,c\}^{\frac{n}{3}} \mapsto \{0,1\}$ by*

$$DIST_{\frac{2n}{3}}(\underline{x}, \underline{y}) = 1 \quad \text{iff} \quad x_i \neq y_i \quad \text{for all} \quad 1 \leq i \leq \frac{n}{3}$$

Proposition 7.13 *$C(M\hat{A}TCH_n) \geq C_{\frac{1}{3}}(DIST_{\frac{2n}{3}})$*

Proof. We first design a deterministic protocol for every triple (S, ρ, k), where

$S = (S_1, S_2, \ldots, S_{\frac{n}{3}})$ is a partition of the vertex set $\{1, 2, \ldots, n\}$ into triples, i.e. $|S_i| = 3$, and $|S_i \cap S_j| = \emptyset$ for $i \neq j$, and

$\rho = (\rho_1, \rho_2, \ldots, \rho_{\frac{n}{3}})$ is a sequence of bijections, $\rho_i : \{a, b, c\} \mapsto S_i$, and

$k \in \{1, 2, \ldots, \frac{n}{3}\}$.

For each ρ_i, we define a complementary mapping, by $\overline{\rho}_i(\gamma) = S_i - \{\rho_i(\gamma)\}$.

To a triple (S, ρ, k), we associate two mappings $\sigma_I : \{a, b, c\}^{\frac{n}{3}} \mapsto P_n$ and $\sigma_{II} : \{a, b, c\}^{\frac{n}{3}} \mapsto Q_n$ defined by

$$\sigma_I(\underline{x}) = \{\overline{\rho}_1(x_1), \overline{\rho}_2(x_2), \ldots, \overline{\rho}_{\frac{n}{3}}(x_{\frac{n}{3}})\}$$

$$\sigma_{II}(\underline{y}) = \{\rho_1(y_1), \rho_2(y_2), \ldots, \rho_{\frac{n}{3}}(y_{\frac{n}{3}})\} - \{\rho_k(y_k)\}$$

The deterministic protocol $B_{(S,\rho,k)}$ works as follows. On input $(\underline{x}, \underline{y})$, player I computes $p = \sigma_I(\underline{x})$ and player II computes $q = \sigma_{II}(\underline{y})$. Next they apply a deterministic protocol A for $\hat{MATCH}$ to find an e such that $(p, q, e) \in \hat{MATCH}$. The result of the protocol is 1 if $\rho_k(y_k) \in e$ and otherwise it is 0.

The probabilistic protocol B for $DIST$ is defined as follows. With probability $\frac{1}{3}$ the result is 0 (without any communication) and with probability $\frac{2}{3}$ it chooses some protocol $B_{(S,\rho,k)}$ uniformly at random, and uses the result of this protocol.

If $DIST(\underline{x}, \underline{y}) = 1$, then $\rho_i(y_i) \in \overline{\rho}_i(x_i)$ for all $1 \leq i \leq \frac{n}{3}$, and therefore the value returned by protocol A must be $e = \overline{\rho}_k(x_k)$ satisfying that $\rho_k(y_k) \in e$, i.e. we have

$$\Pr(B(\underline{x}, \underline{y}) = 1 \mid DIST(\underline{x}, \underline{y}) = 1) = \frac{1}{3} \cdot 0 + \frac{2}{3} \cdot 1 = \frac{2}{3}.$$

If $DIST(\underline{x}, \underline{y}) = 0$, then for some j, $\rho_j(y_j) \notin \overline{\rho}_j(x_j)$. If k happens to be equal to such a j, then we necessarily have that $\rho_k(y_k) \notin e$, resulting in a 0 from the protocol. If in stead $\rho_k(y_k) \in \overline{\rho}_k(x_k)$ then there is probability $\geq \frac{1}{2}$ that protocol A returns an $e = \overline{\rho}_j(x_j)$ with $j \neq k$, resulting in a 0, i.e. we have:

$$\Pr(B(\underline{x}, \underline{y}) = 0 \mid DIST(\underline{x}, \underline{y}) = 0) \geq \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \frac{1}{2} = \frac{2}{3}.$$

□

Proposition 7.14 $C_\epsilon(DIST_{2n}) \geq C_\epsilon(DISJ_{2n})$

Proof. Define two mappings $\tau_I, \tau_{II} : \{0,1\} \mapsto \{a,b,c\}$ by $\tau_I(0) = a$, $\tau_{II}(0) = b$ and $\tau_I(1) = \tau_{II}(1) = c$. Both maps are extended to $\{0,1\}^n$ in the natural way. Clearly, $(\underline{x}, \underline{y}) \in DISJ_{2n}$ iff $(\tau_I(\underline{x}), \tau_{II}(\underline{y})) \in DIST_{2n}$. A probabilistic protocol D for *DISJ* may be constructed by letting player I and player II compute $\tau_I(\underline{x})$ and $\tau_{II}(\underline{y})$, respectively, and use a probabilistic protocol for *DIST*. □

Theorem 7.15 $D_m(MATCH_{\binom{n}{2}}) = \Omega(n)$

Proof. Combine the above propositions and proposition 7.5. □

7.4 Lower bound on Monotone Size

We have so far encountered a number of monotone problems, e.g. threshold functions, binary sorting and Boolean matrix multiplication. We know that the monotone basis $\{+, \cdot\}$ is complete for MON (exercise 1.6). Obviously, $S_m(f) = \Omega(S(f))$ for all $f \in$ MON, but is it also true that $S_m(f) = O(S(f))$? The question came into focus, when Razborov (1985) gave a super-polynomial lower bound on the size of a monotone circuit for a specific (monotone) NP-complete function (This result is shown in the following subsection).

Already Lamagna and Savage (1974) had proven the following result:

Theorem 7.16 $S_m(SORT_n) = \Omega(n \log n)$

Proof. We will use the substitution technique to show that $S_m(SORT_n) \geq \lceil \log n \rceil + S_m(SORT_{n-1})$, which implies the theorem.

Let C be an optimal circuit over $\{+, \cdot\}$ for computing $SORT_n$. Since $SORT_n(\underline{x}) = (Th_n^n(\underline{x}), \ldots, Th_n^1(\underline{x}))$, we denote the n outputs from C by $Th_n^1, \ldots, Th_n^n$. Since Th_n^n depends on all inputs, C must contain a path from each input to Th_n^n. Each gate has fan-in 2, and we can therefore find an input x_i, satisfying that a shortest path from x_i to Th_n^n passes at least $\lceil \log n \rceil$ gates. Let C' be the circuit arising from C when substituting 0 for x_i and eliminating all gates that have one or more constant inputs. Since the Th_n^n-output is now constantly 0, we must have eliminated all gates on a path from x_i to Th_n^n, implying that

$$S_m(C) \geq S_m(C') + \lceil \log n \rceil.$$

Since C' computes $SORT_{n-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ on the last $n-1$ outputs, we have that

$$S_m(SORT_n) \geq S_m(SORT_{n-1}) + \lceil \log n \rceil,$$

which implies the theorem. □

Since binary sorting is computed by a circuit of size $O(n)$, when allowing negation (exercise 3.19), there is at least a factor $\log n$ slack between S_m and S.

The following result makes the situation even worse:

Theorem 7.17 *Let $MM \in \mathrm{MON}_{2n^2,n^2}$ denote Boolean matrix multiplication, i.e. $MM_{2n^2,n^2}(\underline{x}, \underline{y})_{ij} = \sum_{k=1}^{n} x_{i,k} y_{k,j}$.*
Then $S_m(MM_{2n^2,n^2}) = \Omega(n^3)$ and $S(MM_{2n^2,n^2}) = O(n^{2.81})$.

Proof. The upper bound is shown in exercise 7.18. We omit a proof of the lower bound. □

Any hope of a polynomial relation between S_m and S was eventually terminated by Razborov (1985) who proved a super-polynomial lower bound for the size of any monotone circuit deciding the existence of a perfect matching in a bipartite graph, a problem in P and even in NC^2 (exercise 7.14).

7.4.1 The k-clique problem

We are now going to prove Razborovs first lower bound.

Let k be a fixed integer. k-CLIQUE$\in \mathrm{MON}_{\binom{n}{2}}$ is the function that when input an undirected graph with n nodes (represented by a bit for each possible edge) returns the value 1, if the graph contains a complete subgraph (clique) on k nodes.

We are going to prove that any family of monotone circuits computing k-CLIQUE has size $n^{\Omega(\sqrt{k})}$. The proof follows Boppana and Sipser (1990).

The basic idea of the proof is the following. For every monotone circuit C we describe an approximator circuit $\tilde{C}$ disagreeing with C on few inputs only *provided* C is small. However, all approximators will disagree with the k-CLIQUE function on many inputs, implying that C must be large in order to compute k-CLIQUE correctly.

When judging how well $\tilde{C}$ approximates C, we will not directly count the inputs where the two circuits disagree, rather we count the number of positive and negative test graphs giving rise to different output from the circuits.

A *positive* test graph has k distinguished nodes forming a clique. There are no edges apart from the clique. In total, there are $\binom{n}{k} = \Theta(n^k)$ distinct positive test graphs.

In a *negative* test graph each node has a single colour out of $(k-1)$ possible colours. Two nodes are connected by an edge, precisely when they have distinct colours. In total there are $(k-1)^n$ distinct negative test graphs, since graphs with permuted colours are not regarded as identical.

Note that

$$k\text{-CLIQUE}(G) = \begin{cases} 1 & \text{if } G \text{ is a positive test graph} \\ 0 & \text{if } G \text{ is a negative test graph} \end{cases}$$

We need some notation. If $X \neq \emptyset$ is a set of nodes, then let $\lceil X \rceil$ denote a clique indicator, i.e. a circuit detecting whether X forms a clique:

$$\lceil X \rceil(G) = \begin{cases} 1 & \text{if } X \text{ spans a clique in the graph } G \\ 0 & \text{otherwise} \end{cases}$$

E.g. $\sum_{|X|=k} \lceil X \rceil$ computes k-CLIQUE.

An *approximator* is a circuit computing a sum of clique indicators: $\sum_{i=1}^{s} \lceil X_i \rceil$, where $s \leq m$ and $|X_i| \leq l$. If $m \ll \binom{n}{k}$ and $l \ll k$, then it seems intuitively reasonable that any approximator will deviate from k-CLIQUE on many test graphs. The following values turn out to be appropriate:

$$l = \lfloor \sqrt{k} \rfloor \tag{8}$$

$$m = (p-1)^l \cdot l! \tag{9}$$

where

$$p = \sqrt{k} \log n \tag{10}$$

We are going to motivate these choices in connection with the upcoming proofs, of which the first one formalises our intuition that an approximator is really poor at computing k-CLIQUE.

Lemma 7.18 *An approximator is either identically* 0*, or it returns* 1 *on at least* $\frac{1}{2}(k-1)^n$ *of all negative test graphs.*

Proof. Let the approximator $\tilde{C}$ compute $\sum_{i=1}^{s} \lceil X_i \rceil$, where $s \leq m$ and $|X_i| \leq l$. If $\tilde{C}$ is not identically 0, then we must have $s \geq 1$. We are going to argue that $\lceil X_1 \rceil$ alone accepts sufficiently many negative test graphs. For this we use a probabilistic technique:

$$\Pr_{G \text{ is negat. test graph}} (\lceil X_1 \rceil (G) = 1) =$$

$$1 - \Pr_{G \text{ is negat. test graph}} (\text{2 nodes from } X_1 \text{ have same colours in } G) \geq$$

$$1 - \binom{|X_1|}{2} \frac{1}{k-1} \geq$$

$$\frac{1}{2} \quad \text{if} \quad |X_1| \leq \sqrt{k}.$$

(8) is chosen carefully to satisfy this condition. Since we have $(k-1)^n$ negative test graphs in total, the lemma follows. □

We need to describe for every monotone circuit C the construction of an approximator $\tilde{C}$ such that C and $\tilde{C}$ deviates on a number of test graphs proportional to the size of C only.

Without loss of generality, we can assume that C does not use constants. $\tilde{C}$ is constructed from C in the following way:

1. Every input x_i is preserved. (If x_i represents an edge between the nodes u and v, then x_i is identical to the clique indicator $\lceil \{u, v\} \rceil$).
2. Every $+$-operation is replaced by a new operator $\sqcup$.
3. Every $\cdot$-operation is replaced by a new operator $\sqcap$.

The remaining problem consists in defining the binary operators $\sqcup$ and $\sqcap$ on approximators such that $\tilde{a} \sqcup \tilde{b}$ (respectively $\tilde{a} \sqcap \tilde{b}$) deviates very little from $\tilde{a} + \tilde{b}$ (respectively $\tilde{a} \cdot \tilde{b}$). We start by $\sqcup$:

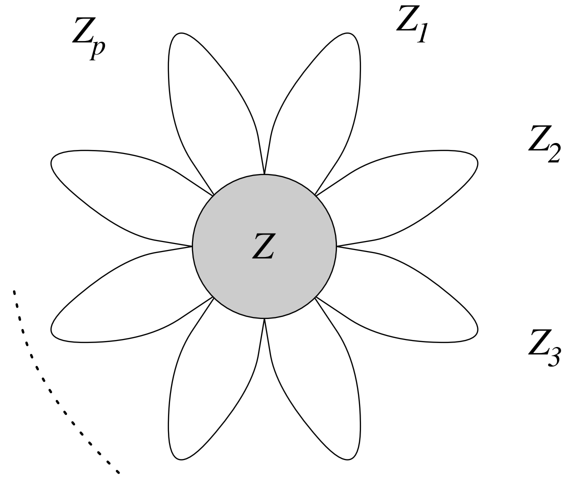

Figure 18: Sunflower

Lemma 7.19 *Given two approximators $\tilde{a}$ and $\tilde{b}$, we can find an approximator $\tilde{a} \sqcup \tilde{b}$ such that*

1. $\tilde{a} \sqcup \tilde{b} \geq \tilde{a} + \tilde{b}$ *for all positive test graphs.*

2. $\tilde{a} \sqcup \tilde{b} \leq \tilde{a} + \tilde{b}$ *for all (apart from $m2^{-p}(k-1)^n$) negative test graphs.*

Proof. If $\tilde{a} = \sum_{i=1}^{r} \lceil X_i \rceil$ and $\tilde{b} = \sum_{j=1}^{s} \lceil Y_j \rceil$ then $\tilde{a} + \tilde{b}$ is not necessarily an approximator, since it may hold up to $2m$ clique indicators.

We are going to form $\tilde{a} \sqcup \tilde{b}$ from $\tilde{a} + \tilde{b}$ by decreasing the number of clique indicators while only changing the output on test graphs within the limits stated in the lemma. We need a definition:

A class of distinct sets $\mathcal{Z} = \{Z_1, \ldots, Z_p\}$ is called a *sunflower* if for all $i \neq j$ it holds that $Z_i \bigcap Z_j = Z$, where $Z = \cap_{i=1}^{p} Z_i$. Figure 18 shows a sunflower. Z is the *centre* of the sunflower, while the Z_i's are the *petals*. If a class of sets $\mathcal{W}$ contains the sunflower $\mathcal{Z}$, we say that we *pick* $\mathcal{Z}$ from $\mathcal{W}$ when we in $\mathcal{W}$ replace the petals of $\mathcal{Z}$ with the centre of $\mathcal{Z}$. The following result guarantees that if $|\mathcal{W}| > m$ then it is possible to pick a sunflower with p petals from $\mathcal{W}$ and this result is the reason for fixing m as in (9).

Lemma 7.20 (Erdös and Rado, 1960) *If $\mathcal{W}$ is a class of* distinct *sets, where each set has at most l elements and $|\mathcal{W}| > (p-1)^l \cdot l!$ then there is a subclass $\mathcal{Z} \subseteq \mathcal{W}$ satisfying that $\mathcal{Z}$ is a sunflower with p petals.*

Proof. We use induction on l. If $l = 1$ then $\mathcal{W}$ is a sunflower with centre $\emptyset$ and $|\mathcal{W}| > p - 1$ petals.

If $l \geq 2$ we let $\mathcal{M} \subseteq \mathcal{W}$ be a maximal class of disjoint sets. If $|\mathcal{M}| \geq p$ then $\mathcal{M}$ is a sufficiently large sunflower.

Otherwise we form the set $S = \bigcup_{M \in \mathcal{M}} M$ that satisfies

$$|S| \leq l(p-1)$$

$$S \cap W \neq \emptyset \text{ for every } W \in \mathcal{W} \tag{11}$$

(11) means that we can find an $s \in S$ such that $s \in W$ for many (at least $\frac{|\mathcal{W}|}{|S|}$) $W \in \mathcal{W}$. We choose such an s and define

$$\mathcal{W}' = \{W - \{s\} \mid s \in W \quad \text{and} \quad W \in \mathcal{W}\}$$

It holds that

$$|\mathcal{W}'| \geq \frac{|\mathcal{W}|}{|S|} > \frac{(p-1)^l \cdot l!}{l(p-1)} = (p-1)^{l-1} \cdot (l-1)!$$

Since every set in $\mathcal{W}'$ has at most $l - 1$ elements, we can (by induction) find a subclass $\mathcal{Z}' \subseteq \mathcal{W}'$ that forms a sunflower with p petals. If this sunflower has centre C then

$$\mathcal{Z} = \{Z' \cup \{s\} \mid Z' \in \mathcal{Z}'\}$$

is a sunflower with centre $C \cup \{s\}$ also having p petals. □

Proof of lemma 7.19 continued. We can now define $\tilde{a} \sqcup \tilde{b}$. Let $\mathcal{W} := \{X_1, \ldots, X_r, Y_1, \ldots, Y_s\}$. If $|\mathcal{W}| > m$ then pick sunflowers (with p petals) from $\mathcal{W}$ until $|\mathcal{W}| \leq m$. Then $\tilde{a} \sqcup \tilde{b} = \sum_{W \in \mathcal{W}} \lceil W \rceil$ is an approximator.

We still need to analyse the deviation of $\tilde{a} \sqcup \tilde{b}$ from $\tilde{a} + \tilde{b}$.

The deviation for positive test graphs is analysed in exercise 7.20. For negative test graphs we use a probability theoretic argument similar to what we did in the

proof of lemma 7.18. If $\mathcal{Z} = \{Z_1, \ldots, Z_p\}$ is a sunflower with centre Z then we have:

$$\Pr_{G \text{ is negat. test graph}}(\sum_{i=1}^{p} \lceil Z_i \rceil(G) = 0 \text{ and } \lceil Z \rceil(G) = 1) \leq$$

$$\Pr_{G \text{ is negat. test graph}}(\sum_{i=1}^{p} \lceil Z_i \rceil(G) = 0 \mid \lceil Z \rceil(G) = 1) =^{1}$$

$$\prod_{i=1}^{p} \Pr_{G \text{ is negat. test graph}}(\lceil Z_i \rceil(G) = 0 \mid \lceil Z \rceil(G) = 1) \leq^{2}$$

$$\prod_{i=1}^{p} \Pr_{G \text{ is negat. test graph}}(\lceil Z_i \rceil(G) = 0) \leq^{3}$$

$$\prod_{i=1}^{p} \frac{1}{2} = 2^{-p}$$

We make no more than $2m - m = m$ pickings. Therefore

$$\Pr_{G \text{ is negat. test graph}}(\tilde{a} \sqcup \tilde{b}(G) = 1 \text{ and } \tilde{a} + \tilde{b}(G) = 0) \leq$$

$$m2^{-p}$$

Since we have $(k-1)^n$ negative test graphs in total, there at most $m2^{-p}(k-1)^n$ negative test graphs with $\tilde{a} \sqcup \tilde{b} > \tilde{a} + \tilde{b}$. □

Lemma 7.21 *Given two approximators $\tilde{a}$ and $\tilde{b}$, there exists an approximator $\tilde{a} \sqcap \tilde{b}$ such that*

[1] This identity uses that the events $\langle \lceil Z_i \rceil(G) = 0 \mid \lceil Z \rceil(G) = 1 \rangle$ $(i = 1, \ldots, p)$ are independent, because the Z_i's form a sunflower with centre Z. The identity is the reason for introducing sunflowers.

[2] Since $\lceil Z \rceil(G) = 0$ implies that $\lceil Z_i \rceil(G) = 0$

[3] This inequality is implied by the reasoning in the proof of lemma 7.18

1. *$\tilde{a} \sqcap \tilde{b} \geq \tilde{a} \cdot \tilde{b}$ for all (apart from at most $m^2\binom{n-l-1}{k-l-1}$) positive test graphs.*

2. *$\tilde{a} \sqcap \tilde{b} \leq \tilde{a} \cdot \tilde{b}$ for all (apart from at most $m^2 2^{-p}(k-1)^n$) negative test graphs.*

Proof. The $\cdot$-case is apparently worse than the $+$-case, since $\tilde{a} = \sum_{i=1}^{r}\lceil X_i \rceil$ and $\tilde{b} = \sum_{j=1}^{s}\lceil Y_j \rceil$, implies $\tilde{a} \cdot \tilde{b} = \sum_{i=1}^{r}\sum_{j=1}^{s}\lceil X_i \rceil \cdot \lceil Y_j \rceil$ and the latter expression does not look like an approximator.

We start by describing, how $\tilde{a} \cdot \tilde{b}$ is transformed into an approximator, and later we analyse the error made by the approximator.

We start by replacing $\lceil X \rceil \cdot \lceil Y \rceil$ with $\lceil X \cup Y \rceil$ obtaining $c_1 = \sum_{i=1}^{r}\sum_{j=1}^{s}\lceil X_i \cup Y_j \rceil$. It is possible that $|X_i \cup Y_j| > l$, but all clique indicators that are too large are thrown away from c_1 resulting in $c_2 = \sum_{k=1}^{t}\lceil Z_k \rceil$, where $|Z_k| \leq l$. There could still be up to m^2 clique indicators, since $t \leq rs \leq m^2$. To decrease this number we pick sunflowers as in the proof of lemma 7.19 resulting in the approximator $\tilde{a} \sqcap \tilde{b} = \sum_{k=1}^{u}\lceil W_k \rceil$.

We first analyse the error of the approximator on positive test graphs. If G is a positive test graph then $\tilde{a} \cdot \tilde{b}(G) = c_1(G) \geq c_2(G) \leq \tilde{a} \sqcap \tilde{b}(G)$ (see exercise 7.21), so the error has the proper bias except when transforming c_1 into c_2. If $c_1(G) = 1$ and $c_2(G) = 0$, then during the transition from c_1 to c_2 we have deleted a clique indicator $\lceil Z \rceil$, where $|Z| \geq l + 1$ and the nodes in Z form a clique in G. Since G is a clique on k nodes, there are at most $\binom{n-l-1}{k-l-1}$ distinct such G's for a given Z. Since c_1 contains at most m^2 clique indicators, we see that $\tilde{a} \sqcap \tilde{b} \geq \tilde{a} \cdot \tilde{b}$ for all apart from at most $m^2\binom{n-l-1}{k-l-1}$ positive test graphs.

Finally, we consider the case of G being a negative test graph. It holds that $\tilde{a} \cdot \tilde{b}(G) \geq c_1(G) \geq c_2(G) \leq \tilde{a} \sqcap \tilde{b}(G)$ (see exercise 7.21), i.e. the error has the proper bias except in the transition from c_2 to $\tilde{a} \sqcap \tilde{b}$, where we pick up to m^2 sunflowers. By an argument similar to the one in the proof of lemma 7.19, we have that $\tilde{a} \sqcap \tilde{b} \leq \tilde{a} \cdot \tilde{b}$ for all apart from at most $m^2 2^{-p}(k-1)^n$ negative test graphs. □

Theorem 7.22 *For a fixed k it holds that*

$$S_m(k\text{-}CLIQUE_{\binom{n}{2}}) = n^{\Omega(\sqrt{k})}.$$

Proof. Let C be a monotone circuit computing k-CLIQUE$_{\binom{n}{2}}$, i.e. $C = 1$ for all $\binom{n}{k}$ positive test graphs, and $C = 0$ for all $(k-1)^n$ negative test graphs.

We have so far described the construction of an approximator $\tilde{C}$ over the basis $\{\sqcup, \sqcap\}$ for a given C. The lemmas 7.19 and 7.21 combined bounds the error of the approximator:

$\tilde{C} \geq C$ for all positive test graphs, apart from at most $S(C) \cdot m^2 \cdot \binom{n-l-1}{k-l-1}$, and

$\tilde{C} \leq C$ for all negative test graphs, apart from at most $S(C) \cdot m^2 \cdot 2^{-p} \cdot (k-1)^n$.

We also know, by lemma 7.18 that the error is large:

Either $\tilde{C} = 0$ on all positive test graphs or $\tilde{C} = 1$ for at least $\frac{1}{2}(k-1)^n$ negative test graphs.

When combining this information, we get:

1. Either

$$S(C) \cdot m^2 \cdot \binom{n-l-1}{k-l-1} \geq \binom{n}{k}$$

   and with substitutions using (8) and (9) we get

$$S(C) = \Omega(p^{-2\sqrt{k}} \cdot n^{\sqrt{k}})$$

   since k is a constant!

2. Or

$$S(C) \cdot m^2 \cdot 2^{-p} \cdot (k-1)^n \geq \frac{1}{2}(k-1)^n$$

   and similarly with substitutions using (8) and (9) we get

$$S(C) = \Omega(p^{-2\sqrt{k}} \cdot 2^p)$$

The two lower bounds are identical, when $n^{\sqrt{k}} = 2^p$, which happens to hold according to (10). By substitution, using (10) we get

$$S(C) = \Omega((\frac{n}{\log^2 n})^{\sqrt{k}}) = n^{\Omega(\sqrt{k})}.$$

□

Exercises

Exercise 7.1 *Let $k = \log n$, and define SHIFT-EQUAL $\in \mathrm{B}_{2n+k}$ by*

$$SHIFT\text{-}EQUAL(x_1, \dots, x_n, y_1, \dots, y_n, z_1, \dots, z_k) = \begin{cases} 1, & \text{if } x_i = y_{i+[\underline{z}] \bmod n} \text{ for all } 1 \le i \le n \\ 0, & \text{otherwise} \end{cases}$$

Show that $C_{best}(SHIFT\text{-}EQUAL_{2n+k}) = \Omega(n)$.

Hint: Show that $C_{best}(SHIFT\text{-}EQUAL_{2n+k}) \ge C(EQUAL_{2m})$ for $m = \frac{n}{4} - \log n + O(1)$.

Exercise 7.2 *Area-Time trade-off in VLSI.*

We consider a chip to be a rectangular grid with processors at selected nodes. Processors are connected by wires along grid edges. Wires may cross, but each grid edge carry at most one wire. In one step (systolic: synchronised globally), a processor can read the bits sent to it in the last step, make some local computation and sent at most one bit out on each connecting wire.

The chip has n special input wires and an output wire. The chip computes a function $f \in \mathrm{B}_n$ in time T if the chip when presented with $x_1, \dots, x_n$ on the input wires renders $f(x_1, \dots, x_n)$ on the output wire after T steps. The area A of the chip is the number of grid squares (or the total length of wires).

Assume that a chip of area A computes a function $f \in \mathrm{B}$ in time T. Prove that

$$AT^2 = \Omega((C_{best}(f))^2)$$

Exercise 7.3 *Show that the communication matrices for $EQUAL_{2n}$ and $DISJ_{2n}$ both have full rank 2^n.*

Exercise 7.4 *Find $O(\log n)$ bounds for the following communication problems.*

(i) The two players have initially bit vectors $\underline{x}$ and $\underline{y}$, respectively, with the property that the parity of the number of 1's in $\underline{x}$ is different from the parity of the number of 1's in $\underline{y}$. The two players must agree on an index "i" such that $x_i \neq y_i$.

(ii) The two players have initially bit vectors $\underline{x}$ and $\underline{y}$, respectively, with the property that the number of 1's in $\underline{x}$ is different from the number of 1's in $\underline{y}$. The two players must agree on an index "i" such that $x_i \neq y_i$.

Exercise 7.5 *Prove a linear lower bound on the communication complexity for the problem of computing the inner product (modulo 2) of two binary vectors.*

Exercise 7.6 *Prove lemma 7.6.*

Exercise 7.7 *Prove lemma 7.8.*

Exercise 7.8 *Show that*

$$2\binom{\frac{\sqrt{n}}{3}}{4\epsilon\frac{\sqrt{n}}{3}}\binom{\frac{8n}{9}}{\sqrt{n}} \leq 2^{-c\sqrt{n}}\binom{n}{\sqrt{n}}$$

for $\epsilon, c > 0$ sufficiently small and n sufficiently large.

Hint: Use Stirling approximation to show that $\binom{m}{m/k} \leq (ke)^{m/k}$.

Exercise 7.9 *Show that the reverse inequality of the one in lemma 7.11 also holds, i.e.*

$$D_m(f) = C(R(f)) \quad \textit{for all} \quad f \in \mathrm{MON}$$

Exercise 7.10 *State and prove a non-monotone version of lemma 7.11 (and definition 7.10)*

Exercise 7.11 *Define $PM \in \mathrm{MON}$ by*

$$PM_{\binom{n}{2}}(\underline{x}) = \begin{cases} 1, & \textit{if the graph } G[\underline{x}] \textit{ has a perfect matching} \\ 0, & \textit{otherwise} \end{cases}$$

Show that $D_m(PM) = \Omega(n)$.

Exercise 7.12 *Define $BPM \in \mathrm{MON}$ by*

$$BPM_{n^2}(\underline{x}) = \begin{cases} 1, & \textit{if the bipartite graph } BPG[\underline{x}] \textit{ has a perfect matching} \\ 0, & \textit{otherwise} \end{cases}$$

Show that $D_m(BPM) = \Omega(n)$.

Exercise 7.13 *Define $CL(k) \in \mathrm{MON}$ by*

$$CL(k)_{\binom{n}{2}}(\underline{x}) = \begin{cases} 1, & \text{if the graph } G[\underline{x}] \text{ has a clique of size } k \\ 0, & \text{otherwise} \end{cases}$$

Show that $D_m(CL(k)) = \Omega(k)$.

Hint: Show that $D_m(CL(\frac{2n}{3}+1)) = \Omega(n)$.

Exercise 7.14 *Define $BPM \in \mathrm{MON}$ by*

$$BPM_{n^2}(\underline{x}) = \begin{cases} 1, & \text{if the bipartite graph } BPG[\underline{x}] \text{ has a perfect matching} \\ 0, & \text{otherwise} \end{cases}$$

For a bipartite graph G on vertices $\{1, 2, \ldots, n\} \cup \{\bar{1}, \bar{2}, \ldots, \bar{n}\}$ define a matrix $T^G(\underline{z})$ with indeterminates $\underline{z} = \{z_{11}, \ldots, z_{nn}\}$ at the entries by

$$[T^G(\underline{z})]_{ij} = \begin{cases} z_{ij}, & \text{if } (i, \bar{j}) \text{ is an edge in } G \\ 0, & \text{otherwise} \end{cases}$$

1. *Show that if G has a perfect matching then $\det(T^G(\underline{z}))$ is a nontrivial polynomial of degree n, and if G has no perfect matching then $\det(T^G(\underline{z}))$ is the 0-polynomial.*

2. *Choose some prime $p > 2n$. Show that*

$$\Pr_{\underline{a} \in \{0,1,\ldots,2n\}^{n^2}} (\det(T^G(\underline{a})) \neq 0 \bmod p \mid \det(T^G(\underline{z})) \not\equiv 0) > \frac{1}{2}$$

   Hint: observe that $\det(T^G(\underline{z})) = p_{0ij}(\underline{z}) + z_{ij}p_{1ij}(\underline{z})$ where p_{kij} does not contain z_{ij} and use the inequality $(1 - \frac{1}{2n+1})^n > \frac{1}{2}$.

3. *Show that there exists $\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_{n^2} \in \{0, 1, \ldots, 2n\}^{n^2}$ such that for all bipartite graphs G on $n + n$ nodes,*

$$\det(T^G(\underline{z})) \quad \text{is the } 0\text{-polynomial}$$

   if and only if

$$\det(T^G(\underline{a}_i)) = 0 \quad \text{for all } i = 1, \ldots, n^2$$

4. *Show that* $D(BPM) = O(\log^2 n)$
   Hint: See exercise 6.11

Exercise 7.15 *Given any two functions* g, h *with* $g(n) \leq \sqrt{n}$ *and* $h(n) = o(g(n))$, *construct a monotone function* f *such that* $D_m(f) \notin O(h(n))$ *and* $D_m(f) = O(g(n))$.

Exercise 7.16 *Given any function* g *with* $g(n) \leq \sqrt{n}$, *construct a monotone function* f *such that* $D_m(f) = \Omega(g(n))$ *and* $D(f) = O(log^2 g(n))$.

Hint: Use exercises 7.12 and 7.14.

Exercise 7.17 *Let* $S_m^{\wedge}(f)$ *denote the minimum number of* $\cdot$*-gates a correct monotone circuit for* f *can possibly have. Similarly, let* $S_m^{\vee}(f)$ *denote the minimum number of* $+$*-gates in any correct monotone circuit for* f.

1. *Show that* $S_m^{\vee}(Th_n^2) \geq 2n - 4$. *(Hint: modify the proof of theorem 5.8)*
2. *Show that* $S_m^{\vee}(Th_n^2) = S_m^{\wedge}(Th_n^{n-1})$. *(Hint: duality)*
3. *Show that* $S_m(f) \geq 4n - 12$ *for the function*

$$f(x_1, \ldots, x_n) = \\ Th_{n-1}^{n-2}(x_1, \ldots, x_{n-1}) + (x_n \cdot Th_{n-1}^2(x_1, \ldots, x_{n-1}))$$

Exercise 7.18 *Show that* $S(MM_{2n^2,n^2}) = O(n^{\log_2 7} \log^2 n)$.

Exercise 7.19 *Show the following modified version of lemma 7.20, where the assumption of the sets being distinct is removed:*

If $W_1, W_2, \ldots, W_n$ *is a collection of sets (not necessarily distinct) with at most* l *elements in a single set and* $n > (p-1)^{(l+1)} \cdot l!$ *then there exists a sub-collection* $W_{k_1}, W_{k_2}, \ldots, W_{k_p}$ *such that* $\cap_{i=1}^p W_{k_i} = W_{k_i} \cap W_{k_j}$ *for all* $i \neq j$.

Exercise 7.20 *Show that the construction of* $\tilde{a} \sqcup \tilde{b}$ *in lemma 7.19 satisfies that* $\tilde{a} \sqcup \tilde{b}(G) \geq \tilde{a} + \tilde{b}(G)$ *for all positive test graphs* G.

Exercise 7.21 *In the proof of lemma 7.21 we named some sub-results* c_1 *and* c_2 *when constructing the approximator* $\tilde{a} \sqcap \tilde{b}$ *for* $\tilde{a} \cdot \tilde{b}$. *Show that*

1. $\tilde{a} \cdot \tilde{b}(G) = c_1(G) \geq c_2(G) \leq \tilde{a} \sqcap \tilde{b}(G)$ *for every positive test graph* G.

2. $\tilde{a} \cdot \tilde{b}(G) \geq c_1(G) \geq c_2(G) \leq \tilde{a} \sqcap \tilde{b}(G)$ *for every negative test graph* G.

Exercise 7.22 *What fails, if you try to generalise the proof of theorem 7.22 to obtain a lower bound for* $S(k\text{-}CLIQUE_{\binom{n}{2}})$*?*

Literature

1. Babai, L., Frankl, P. and Simon, J. (1986), Complexity Classes in Communication Complexity Theory. *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, 337-347.

2. Boppana, R. B. and Sipser, M. (1990), The Complexity of Finite Functions. *Handbook of Theoretical Computer Science* **A**, 757–804.

3. Erdös, P., and Rado, R. (1960), Intersection Theorems for Systems of Sets. *J. London Math. Soc.* **35**, 85–90.

4. Kalyanasundaram, B. and Schnitger, G. (1987), The Probabilistic Communication Complexity of Set Intersection. *Proc. 2nd Ann. Conf. Structure in Complexity Theory*, 41–49.

5. Lamagna, E. A. and Savage, J. E. (1974), Combinational Complexity of some Monotone Functions. *Proc. 15th Ann. IEEE Symp. on Switching and Automata Theory*, 140–144.

6. Mulmuley, K., Vazirani, U. V. and Vazirani, V. V. (1987), Matching is as easy as Matrix Inversion. *Combinatorica* **7**, 105-113.

7. Pratt, V. R. (1974), The Power of Negative Thinking in Multiplying Boolean Matrices. *SIAM J. Comput.* **4**, 326–330.

8. Raz, R. and Wigderson, A. (1992), Monotone Circuits for Matching Require Linear Depth. *J. of the Assoc. Comput. Mach.* **39**, 736-744.

9. Razborov, A. A. (1985), Lower Bounds on the monotone Complexity of Some Boolean Functions. *Soviet Math. Dokl.* **31**, 354–357.

10. Razborov, A. A. (1985), A Lower Bound on the Monotone Network Complexity of the Logical Permanent. *Math. Notes* **37**, 485–493.

11. Razborov, A. A. (1992), On the Distributional Complexity of Disjointness. *Theoretical Computer Science* **106**, 385-390.

12. Yao, A. C.-C. (1979), Some Complexity Questions Related to Distributive Computing. *Proc. 11th ACM Symp. on Theory of Computing,* 209–213.

Class	Exists	Example (if exists)
AC^0	no	
TC^0	?	
NC^1	yes	a specific regular language
NL/poly	yes	*PATH*
SAC^1	yes	a specific context free language
P/poly	yes	linear programming
NP/poly	yes	*SAT*
B	no	

Table 1: The existence (and examples) of p-complete problems for selected classes

8 Overview

8.1 Class Overview

In figure 19 we present known inclusion relations between some of the non-uniform classes that we have seen. We know that $AC^0 \neq TC^0$, but it would be consistent with our present knowledge if $TC^0 = NP/poly$.

In figure 20 we, similarly, present known inclusion relations between some uniform classes. The space hierarchy theorem (presented in dAlg) implies that NL $\neq$ PSPACE. It is, however, consistent with our present knowledge if L = PH. En route we have seen some relations between uniform and nonuniform complexity classes. In particular, we know that BPP $\subseteq$ P/poly in addition to C $\subseteq$ C/poly.

In table 1 we have indicated the existence of p-complete problems for selected classes. Completeness in the class NP is a property indicating that it is hard to find an efficient solution at all to the problem in question, and similarly completeness in the class P is a property indicating that it is hard to find an efficient *parallel* solution to the problem in question. It is perhaps surprising that such a large number of classes (as it happens) do possess p-complete problems. The existence of a p-complete problem for a class means that all the difficult problems in the class are quite similar. They can all be solved by p-projection to a single family of circuits.

Johnson (1990) presents a comprehensive class overview and Skyum and Valiant (1985) show the existence of p-complete problems for pF, pdC, pC and pD.

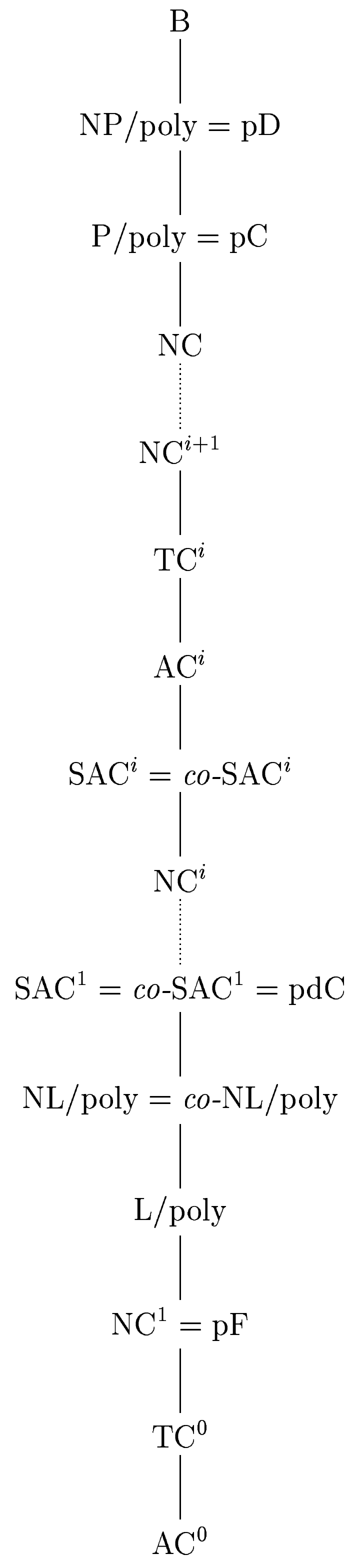

Figure 19: Non-uniform Classes

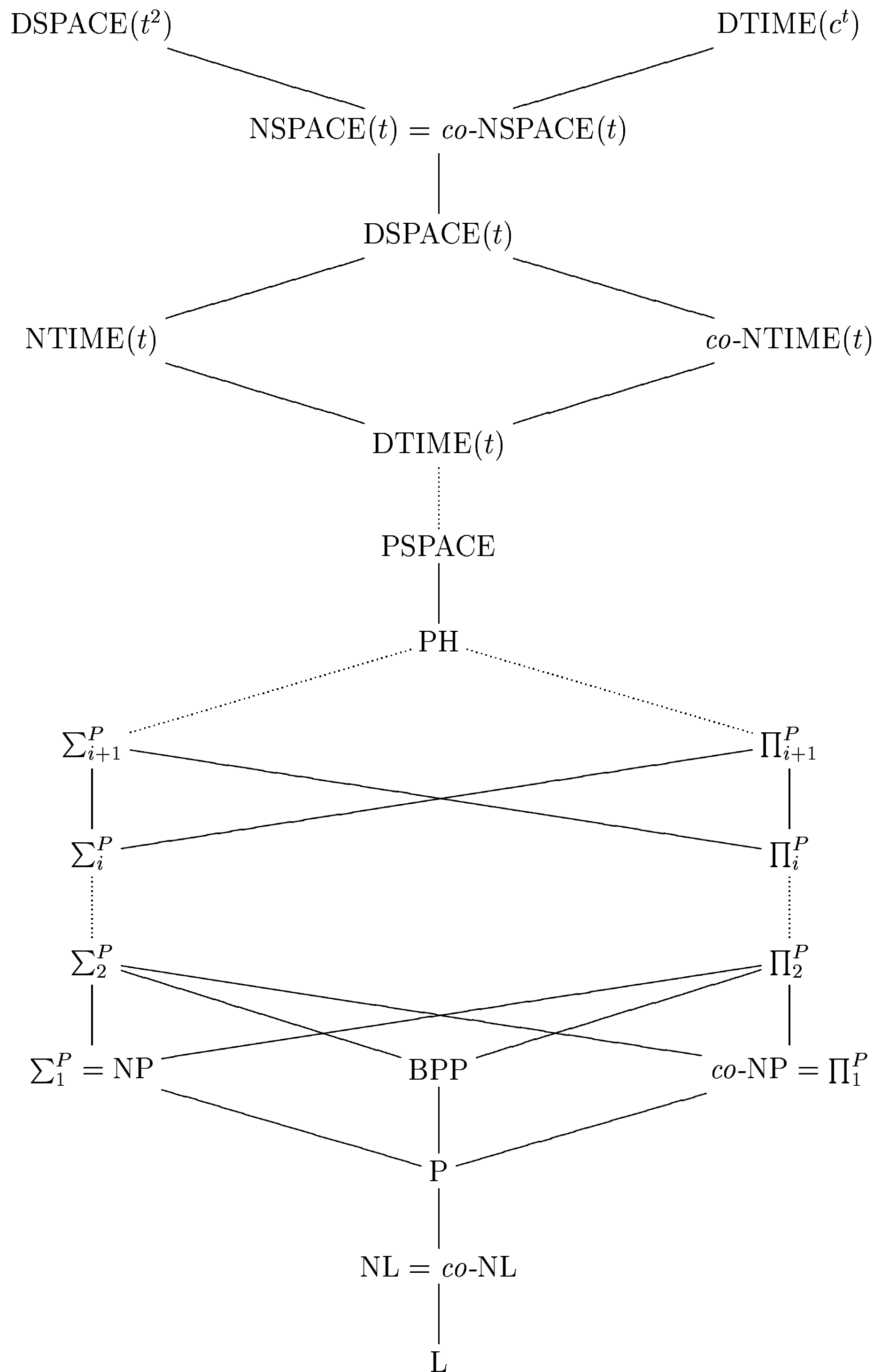

Figure 20: Uniform Classes

8.2 General Literature

There exist several encyclopedic treatises covering most of the existing knowledge (with extensive bibliographies) within a restricted subject area at the time of publishing:

1. *Handbook of Theoretical Computer Science, Vol* **A** *Algorithms and Complexity* (ed. van Leeuwen). Elsevier, 1990.

   Each chapter of the book presents a thorough overview within a selected (narrow) subject. Among the chapters are: Machine Models and Simulation, A Catalog of Complexity Classes, Machine-Independent Complexity Theory, Kolmogorov Complexity and its Applications, Algebraic Complexity Theory, The Complexity of Finite Functions, Communication Networks, VLSI Theory.

2. Wagner, K. and Wechsung, G., *Computational Complexity.* Reidel, 1986.

   The book is concerned with classic (Turing Machine based) complexity theory. A nonstandard notation impedes browsing.

3. Wegener, I., *The Complexity of Boolean Functions.* Wiley, 1987.

4. Kushilevitz, E. and Nisan, N., *Communication Complexity.* Cambridge University Press, 1996.

5. Bürgisser, P., Clausen, M. and Shokrollahi, M.A., *Algebraic Complexity Theory.* Volume 315 of Grundlehren der Mathematischen Wissenschaften. Springer, 1997.

Compared to encyclopaedia the text books present selected bits of a broader subject. Usually, problems and exercises are provided to support the students work with the subject.

1. Balcázar, J. L., Díaz, J. and Gabarro, J., Structural Complexity. Springer Verlag, 1988 (Vol. 1), 1990 (Vol. 2).

   These volumes have a machine based perspective on complexity theory, but they do also describe nonuniform complexity measures.

2. Papadimitriou, C. H., Computational Complexity. Addison Wesley, 1994.

New results are published at conferences and/or in journals. The following journals are particularly rich with results from complexity theory:

1. *Computational Complexity*
2. *J. of the Assoc. Comput. Mach.*
3. *SIAM J. Comput.*
4. *Mathematical Systems Theory*
5. *Information and Computation*
6. *J. of Complexity*
7. *SIAM J. Discr. Math.*
8. *Theoretical Computer Science*

One should keep an eye on proceedings from the following conferences (among others):

1. "STOC": ACM Symposium on Theory of Computing.
2. "FOCS": IEEE Symposium on Foundations of Computer Science.
3. "Structures": IEEE Structure in Complexity Theory.
4. "STACS": Symposium on Theoretical Aspects of Computer Science.

Recently, electronic publications have become an increasingly important source of both new and old results.

1. *Electronic Colloquium on Computational Complexity.*

   http://www.eccc.uni-trier.de/eccc/

   This www-page contains access to tech reports, lecture notes, text books, email addresses of complexity theoreticians, conference lists and much more.

Literature

1. Johnson, D. S. (1990), A Catalogue of Complexity Classes. *Handbook of Theoretical Computer Science* **A**, 67–161.
2. Skyum, S and Valiant, L. G. (1985), A Complexity Theory Based on Boolean Algebra. *J. Assoc. Comput. Mach.* **32**, 484–502.