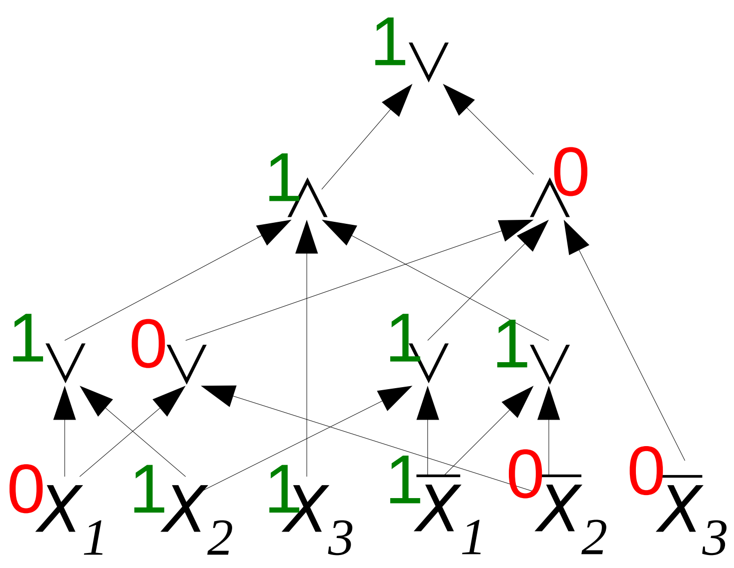
Computational complexity

lecture 4

Boolean circuits

Last lecture: boolean circuits

- directed acyclic graph
- OR gates, AND gates, input gates $X_1,\ldots,X_n$, negated input gates $\overline{X}_1,\ldots,\overline{X}_n$
- typical usage: a single output gate; result 1 when the input word belongs to a language
- a sequence of circuits – one circuit for every input length *n*



Simulating machines by circuits

Theorem

Every language recognizable in time $T(n)$ on a single-tape machine can be recognized by a sequence of circuits $(C_n)_{n\in\mathbb{N}}$ of depth $O(T(n))$ and number of gates $O((T(n))^2)$.

(actually, a stronger variant can be proven: depth $O(T(n))$ and $O(T(n)\cdot log(T(n)))$ gates, even for a multi-tape machine)

Additionally, the circuit C_n can be generated in logarithmic space (thus: in polynomial time) in n. (i.e., there exists a TM working in logarithmic space, which on input 1^n outputs a representation of the circuit C_n)

Simulating machines by circuits

<u>Theorem</u>

Every language recognizable in time $T(n)$ on a single-tape machine can be recognized by a sequence of circuits $(C_n)_{n\in\mathbb{N}}$ of depth $O(T(n))$ and number of gates $O((T(n))^2)$.

<u>Proof</u>

- Fix some M recognizing our language in time $T(n)$; fix also some n.
- We can assume that runs of M on words of length n have length precisely $T(n)$ (if M stops earlier, we repeat the last configuration).
- M uses at most $T(n)$ tape cells.
- A computation of M can be written in a square $T(n)\times T(n)$

Simulating machines by circuits

A computation of M can be written in a square $T(n) \times T(n)$:

- Every row consists of a tape contents in some step
- In the cell, over which the head is located, we additionally write the state.

▷$\underline{1}$	a	b	a	b	c	a	⊥	⊥
▷	a$\underline{5}$	b	a	b	c	a	⊥	⊥
▷	b$\underline{3}$	b	a	b	c	a	⊥	⊥
▷$\underline{4}$	b	b	a	b	c	a	⊥	⊥
▷	b$\underline{2}$	b	a	b	c	a	⊥	⊥
▷	b	b$\underline{5}$	a	b	c	a	⊥	⊥
▷	b	c	a$\underline{1}$	b	c	a	⊥	⊥
▷	b	c	a	b$\underline{4}$	c	a	⊥	⊥
▷	b	c	a	b$\underline{6}$	c	a	⊥	⊥

Simulating machines by circuits

A computation of M can be written in a square $T(n) \times T(n)$:

- Every row consists of a tape contents in some step
- In the cell, over which the head is located, we additionally write the state.
- The content of a cell depends only on the three cells located directly over it.

▷*1*	a	b	a	b	c	a	⊥	⊥
▷	a*5*	b	a	b	c	a	⊥	⊥
▷	b*3*	b	a	b	c	a	⊥	⊥
▷*4*	b	b	a	b	c	a	⊥	⊥
▷	b*2*	b	a	b	c	a	⊥	⊥
▷	b	b*5*	a	b	c	a	⊥	⊥
▷	b	c	a*1*	b	c	a	⊥	⊥
▷	b	c	a	b*4*	c	a	⊥	⊥
▷	b	c	a	b*6*	c	a	⊥	⊥

Simulating machines by circuits

A computation of M can be written in a square $T(n) \times T(n)$:

- Every row consists of a tape contents in some step
- In the cell, over which the head is located, we additionally write the state.
- The content of a cell depends only on the three cells located directly over it.
- Gate (i,j,z) – in the cell having coordinates i,j there is z
- The value of a gate (i,j,z) is a function of gates $(i-1,j-1,z')$, $(i-1,j,z')$, $(i-1,j+1,z')$ for all z' – it can be realized by a circuit of a constant size (the number of possible z,z' is fixed – independent on n)
- Output gate: in the last row there is an accepting state
- Details in notes of D.Niwiński

Simulating machines by circuits

Is it the case that every language recognizable by a sequence of circuits can be recognized by a Turing machine?

Simulating machines by circuits

Is it the case that every language recognizable by a sequence of circuits can be recognized by a Turing machine?

NO! – circuits need not to be <u>uniform</u>

(a sequence of circuits can recognize an arbitrary language, a Turing machine cannot)

Simulating machines by circuits

Is it the case that every language recognizable by a sequence of circuits can be recognized by a Turing machine?

NO! – circuits need not to be uniform

(a sequence of circuits can recognize an arbitrary language, a Turing machine cannot)

A theorem which is true:

There is a Turing machine (working in quadratic time), which inputs a representation of a circuit C_n and a word of w of length n, and computes the value of C_n on word w.

Turing machines with advice

A Turing machine with advice – a model that is non-uniform, but sequential.

Definition: A machine M together with a sequence of words $k_0, k_1, k_2, \ldots$ recognizes a language L iff

$$w \in L \Leftrightarrow k_{|w|}\$w \in L(M)$$

Turing machines with advice

A Turing machine with advice – a model that is non-uniform, but sequential.

Definition: A machine M together with a sequence of words $k_0, k_1, k_2, \ldots$ recognizes a language L iff

$$w \in L \Leftrightarrow k_{|w|}\$w \in L(M)$$

We consider the running time with respect to $|w|$, not with respect to the whole word.

E.g. an exponential advice enforces exponential running time (it is necessary to read it).

Turing machines with advice

A <u>Turing machine with advice</u> – a model that is non-uniform, but sequential.

Definition: A machine M together with a sequence of words $k_0, k_1, k_2, \ldots$ recognizes a language L iff

$$w \in L \Leftrightarrow k_{|w|}\$w \in L(M)$$

We consider the running time with respect to $|w|$, not with respect to the whole word.

E.g. an exponential advice enforces exponential running time (it is necessary to read it).

class **P/poly** – languages recognizable in polynomial time by a machine with advice (of polynomial size)

Turing machines with advice

class **P/poly** – languages recognizable in polynomial time by a machine with advice (of polynomial size)

Theorem

A language belongs to **P/poly** iff it is recognizable by a sequence of circuits of polynomial size.

Proof

Turing machines with advice

class **P/poly** – languages recognizable in polynomial time by a machine with advice (of polynomial size)

Theorem

A language belongs to **P/poly** iff it is recognizable by a sequence of circuits of polynomial size.

Proof

⇒ We convert the machine to a circuit.
The advice can be hard-coded in the circuit.

Turing machines with advice

class **P/poly** – languages recognizable in polynomial time by a machine with advice (of polynomial size)

<u>Theorem</u>

A language belongs to **P/poly** iff it is recognizable by a sequence of circuits of polynomial size.

<u>Proof</u>

⇒ We convert the machine to a circuit.
The advice can be hard-coded in the circuit.

⇐ k_n consists of a representation of C_n;
we evaluate C_n using a Turing machine

Turing machines with advice

The **P/poly** class is non-uniform – it contains undecidable languages.

For example:

$L=\{1^n$: the n-th Turing machine halts on every input$\}$

Turing machines with advice

The **P/poly** class is non-uniform – it contains undecidable languages.

For example:

$L=\{1^n$ $:$ the n-th Turing machine halts on every input$\}$

The **P/poly** class is useful for modeling languages (problems), which can be solved quickly after a (probably very costly) preprocessing.
E.g., in cryptography one sometimes assumes that an intruder has computing power in **P/poly**.

Turing machines with advice

The **P/poly** class is non-uniform – it contains undecidable languages.

For example:

$L=\{1^n$: the n-th Turing machine halts on every input$\}$

The **P/poly** class is useful for modeling languages (problems), which can be solved quickly after a (probably very costly) preprocessing.
E.g., in cryptography one sometimes assumes that an intruder has computing power in **P/poly**.

Open problem: does **NP**$\not\subseteq$**P/poly**?

(this is a stronger statement than P≠NP, because obviously P⊆P/poly)

Uniform sequences of circuits

A sequence of circuits $C_0, C_1, C_2, \ldots$ is <u>uniform</u> if it is computable in logarithmic space, i.e., there exists a TM working in logarithmic space, which on input 1^n outputs the representation of circuit C_n

Uniform sequences of circuits

A sequence of circuits $C_0, C_1, C_2, \ldots$ is <u>uniform</u> if it is computable in logarithmic space, i.e., there exists a TM working in logarithmic space, which on input 1^n outputs the representation of circuit C_n

Let us recall the definition – functions computable in logarithmic space:

- a read-only input tape
- working tapes of logarithmic length
- an output tape, over which the head may only move right

Notice that in logarithmic space one can compute an output which is much longer than logarithmic (but necessarily is polynomial)

Corollary: such a procedure can only generate circuits C_n which are of size polynomial in n.

Uniform sequences of circuits

A sequence of circuits $C_0,C_1,C_2,...$ is <u>uniform</u> if it is computable in logarithmic space, i.e., there exists a TM working in logarithmic space, which on input 1^n outputs the representation of circuit C_n

Let us recall the definition – functions computable in logarithmic space:

- a read-only input tape
- working tapes of logarithmic length
- an output tape, over which the head may only move right

Notice that in logarithmic space one can compute an output which is much longer than logarithmic (but necessarily is polynomial)

Theorem
Functions computable in logarithmic space are closed under composition.

<u>Proof</u>
When the second TM wants to read the k-th bit of the output of the first machine, then we run the first TM, and we only check the value of the k-th bit of its output, ignoring the rest of the output.

Uniform sequences of circuits

Theorem

A language is recognizable by a uniform sequence of circuits iff it is in **P**.

Proof

⇒ obvious: having an input word of length n generate the n-th circuit, and compute its value

⇐ the algorithm given previously, which constructs a circuit basing on a Turing machine and on the input length n, works in logarithmic space (it only has to remember for which cell of the square it currently outputs gates; this fits in a logarithmic space)

Circuits of small depth

- class $\mathbf{AC}^k$ – languages recognizable by a sequence of circuits of depth $O((\log(n))^k)$, and of polynomial size
- most interesting cases: $\mathbf{AC}^0$ (constant depth), $\mathbf{AC}^1$ (logarithmic depth)
- $\mathbf{AC} = \bigcup_{k \in \mathbb{N}} \mathbf{AC}^k$

Circuits of small depth

- class **AC**^{k} – languages recognizable by a sequence of circuits of depth $O((\log(n))^k)$, and of polynomial size
- most interesting cases: **AC**^{0} (constant depth), **AC**^{1} (logarithmic depth)
- **AC**$=\bigcup_{k\in\mathbb{N}}$**AC**^{k}
- class **NC**^{k} – languages recognizable by a sequence of circuits of depth $O((\log(n))^k)$, of polynomial size, and <u>of fan-in 2</u> (i.e., every gate has at most 2 predecessors)
- class **NC**^{0} is not interesting (only a constant number of bits is checked)
- **NC**$=\bigcup_{k\in\mathbb{N}}$**NC**^{k}

Circuits of small depth

Uniform variant:

- class **u-AC**^{k} – languages recognizable by a <u>uniform</u> (i.e., computable in logarithmic space) sequence of circuits of depth $O((log(n))^k)$
- **u-AC**$=\cup_{k\in\mathbb{N}}$**u-AC**^{k}
- class **u-NC**^{k} – languages recognizable by a <u>uniform</u> sequence of circuits of depth $O((log(n))^k)$ and <u>of fan-in *2*</u>
- **u-NC**$=\cup_{k\in\mathbb{N}}$**u-NC**^{k}

implies polynomial size

Remark: Different names are used for these classes: **uniform-AC**^{k} or **u-AC**^{k} or $\mathbf{U_L}$**-AC**^{k} or **AC**^{k} (i.e., some authors already in the definition of **AC**^{k} assume that the sequence of circuits is uniform)

Circuits of small depth

Example:

Binary matrix multiplication is in **u-AC**^{0}

[more precisely: the language of tuples (M,N,i,j) such that $(M \cdot N)_{i,j} = 1$]

$$(M \cdot N)_{i,j} = \bigvee_k M_{i,k} \wedge N_{k,j}$$

- level 1: compute $M_{i,k} \wedge N_{k,j}$ for every (i,j,k)
- level 2: for every (i,j) compute a big disjunction
- additional two levels: select the cell (i,j) specified on input
- it is easy to generate this circuit in logarithmic space

Circuits of small depth

Example:

Binary matrix multiplication is in **u-AC**^{0}

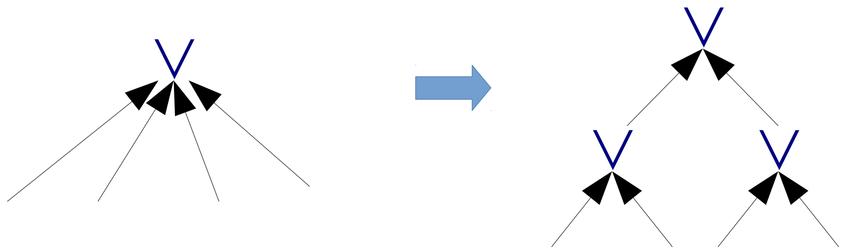
[more precisely: the language of tuples (M,N,i,j) such that $(M \cdot N)_{i,j} = 1$]

$$(M \cdot N)_{i,j} = \bigvee_k M_{i,k} \wedge N_{k,j}$$

- level 1: compute $M_{i,k} \wedge N_{k,j}$ for every (i,j,k)
- level 2: for every (i,j) compute a big disjunction
- additional two levels: select the cell (i,j) specified on input
- it is easy to generate this circuit in logarithmic space

Binary matrix multiplication is in **u-NC**^{1} as well

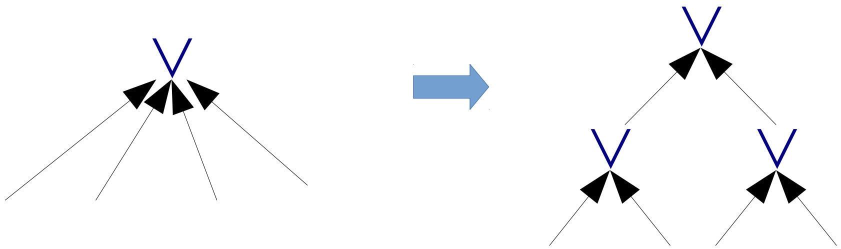
- a disjunction of n values (on level 2) can be realized as a tree of depth $\log(n)$ consisting of n-1 disjunctions of fan-in 2



Circuits of small depth

The same can be done in general:

every disjunction (conjunction) of m values can be replaced by a tree of depth $log(m) \leq c \cdot log(n)$ consisting of m-1 disjunctions (conjunctions) of fan-in 2



Thus we obtain that:

$\mathbf{AC}^k \subseteq \mathbf{NC}^{k+1}$ & $\mathbf{u\text{-}AC}^k \subseteq \mathbf{u\text{-}NC}^{k+1}$

By definition we also have that:

$\mathbf{NC}^k \subseteq \mathbf{AC}^k$ & $\mathbf{u\text{-}NC}^k \subseteq \mathbf{u\text{-}AC}^k$

Thus in particular:

AC=**NC** & **u-AC**=**u-NC**

Circuits of small depth

Intuition: **u-NC** contains problems, which can be quickly solved by parallel algorithm

An open problem: does **u-NC**≠**P**?

Circuits of small depth

Intuition: **u-NC** contains problems, which can be quickly solved by parallel algorithm

An open problem: does **u-NC≠P**?

We have a sequence of inclusions:

u-AC^0⊆**u-NC**^{1}⊆**u-AC**^{1}⊆**u-NC**^{2}⊆...⊆**u-AC=u-NC**⊆**P**⊆**NP**⊆**PSPACE**

It is conjectured that all of them are strict, but it is only known that:

- **u-AC**^{0}≠**u-NC**^{1}
- **u-NC≠PSPACE**

Circuits of small depth

Intuition: **u-NC** contains problems, which can be quickly solved by parallel algorithm

An open problem: does **u-NC≠P**?

We have a sequence of inclusions:

$\mathbf{u\text{-}AC}^0 \subseteq \mathbf{u\text{-}NC}^1 \subseteq \mathbf{u\text{-}AC}^1 \subseteq \mathbf{u\text{-}NC}^2 \subseteq \ldots \subseteq \mathbf{u\text{-}AC} = \mathbf{u\text{-}NC} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$

It is <u>conjectured</u> that all of them are strict, but it is only known that:

- $\mathbf{u\text{-}AC}^0 \neq \mathbf{u\text{-}NC}^1$
- **u-NC≠PSPACE**

Why **u-NC≠PSPACE**?

Follows from the hierarchy theorem, because **u-NC⊆polyL**
(on tutorials you will prove that $\mathbf{u\text{-}NC}^1 \subseteq \mathbf{L}$)

Why $\mathbf{u\text{-}AC}^0 \neq \mathbf{u\text{-}NC}^1$?

<u>Following slides</u>

The parity language

PARITY – the language of those words $\{0,1\}$ in which the number of ones is even

Fact: PARITY$\in$**u-NC**^{1}

We count ones modulo 2 – circuit of tree-like shape.

Theorem (1986): PARITY$\notin$**AC**^{0}

Proof – the following part of the lecture

- It is one of quite rare nontrivial proofs saying that some problem cannot be solved in some complexity class.
- (Mostly hardness theorems are relative – is a problem A is hard, then a problem B is hard, e.g. NP-completeness)

PARITY$\notin$**AC**^{0}

- We are going to consider multi-variable polynomials over the field $\mathbb{Z}_3=\{0,1,2\}$ (we will use them to approximate the behavior of a circuit)
- A polynomial p (of n variables) is called <u>*proper*</u> if for arguments in $\{0,1\}^n$ it gives results in $\{0,1\}$ (we are interested only in such polynomials - they define a boolean function of n variables, like circuits)

PARITY$\notin$**AC**^{0}

- We are going to consider multi-variable polynomials over the field $\mathbb{Z}_3=\{0,1,2\}$ (we will use them to approximate the behavior of a circuit)
- A polynomial p (of n variables) is called *proper* if for arguments in $\{0,1\}^n$ it gives results in $\{0,1\}$ (we are interested only in such polynomials - they define a boolean function of n variables, like circuits)
- The *total degree* of a polynomial p is defined as the sum of exponents in a monomial in p, e.g., $x^4y^1+x^1y^2z^3$ has degree 6

PARITY$\notin$**AC**^{0}

- We are going to consider multi-variable polynomials over the field $\mathbb{Z}_3=\{0,1,2\}$ (we will use them to approximate the behavior of a circuit)
- A polynomial p (of n variables) is called *proper* if for arguments in $\{0,1\}^n$ it gives results in $\{0,1\}$ (we are interested only in such polynomials - they define a boolean function of n variables, like circuits)
- The *total degree* of a polynomial p is defined as the sum of exponents in a monomial in p, e.g., $x^4y^1+x^1y^2z^3$ has degree 6

Fix a depth d. We will prove that PARITY cannot be recognized by a sequence (even not necessarily uniform) of circuits of depth d and polynomial size.

PARITY$\notin$**AC**^{0}

- We are going to consider multi-variable polynomials over the field $\mathbb{Z}_3=\{0,1,2\}$ (we will use them to approximate the behavior of a circuit)
- A polynomial p (of n variables) is called *proper* if for arguments in $\{0,1\}^n$ it gives results in $\{0,1\}$ (we are interested only in such polynomials - they define a boolean function of n variables, like circuits)
- The *total degree* of a polynomial p is defined as the sum of exponents in a monomial in p, e.g., $x^4y^1+x^1y^2z^3$ has degree 6

Fix a depth d. We will prove that PARITY cannot be recognized by a sequence (even not necessarily uniform) of circuits of depth d and polynomial size.

General idea:

- Every circuit of small depth can be approximated by a proper polynomial of low degree (Lemma 1)
- The parity function cannot be approximated by a polynomial of low degree (Lemma 2)

PARITY$\notin$**AC**^{0}

Lemma 1. For every $t>0$ and n, for every circuit C with n input gates and depth d there exists a proper polynomial of n variables and total degree $\leq(2t)^d$, which differs from C on at most $\frac{|C|}{2^t}2^n$ inputs (where $|C|$ denotes the number of gates in C)

PARITY$\notin$**AC**^{0}

Lemma 1. For every $t>0$ and n, for every circuit C with n input gates and depth d there exists a proper polynomial of n variables and total degree $\leq (2t)^d$, which differs from C on at most $\frac{|C|}{2^t} 2^n$ inputs (where $|C|$ denotes the number of gates in C)

We will use this lemma with $2t=n^{1/(2d)}$

Then we obtain polynomials of degree $\leq\sqrt{n}$, while the fraction $|C|/2^t$ tends to 0 when $|C|$ is polynomial in n, and d is constant.

PARITY$\notin$**AC**^{0}

Lemma 1. For every $t>0$ and n, for every circuit C with n input gates and depth d there exists a proper polynomial of n variables and total degree $\leq(2t)^d$, which differs from C on at most $\frac{|C|}{2^t}2^n$ inputs (where $|C|$ denotes the number of gates in C)

We will use this lemma with $2t=n^{1/(2d)}$

Then we obtain polynomials of degree $\leq\sqrt{n}$, while the fraction $|C|/2^t$ tends to 0 when $|C|$ is polynomial in n, and d is constant.

Lemma 2. For large enough n every polynomial of n variables and total degree $\leq\sqrt{n}$ differs from the parity function on at least $\frac{1}{100}2^n$ inputs.

Lemma 1 + Lemma 2 → polynomial circuits of constant depth cannot recognize PARITY

Proof of Lemma 1 (⋆)

<u>Lemma 1.</u> For every $t>0$ and n, for every circuit C with n input gates and depth d there exists a proper polynomial of n variables and total degree $\leq(2t)^d$, which differs from C on at most $\frac{|C|}{2^t}2^n$ inputs (where $|C|$ denotes the number of gates in C)

<u>Proof.</u>

- Fix n, t and a circuit C of depth d.
- Assume w.l.o.g. that C uses only OR and NOT gates.
- To every gate of C we will assign a proper polynomial of n variables $x_1,\ldots,x_n$, by induction on the depth of the gate, so that it will compute the value of this gate C for relatively many inputs

Proof of Lemma 1 (∗)

To every gate of C we will assign a proper polynomial of n variables $x_1,\ldots,x_n$, by induction on the depth of the gate, so that it will compute the value of this gate C for relatively many inputs:

- i-th input gate – take the polynomial x_i, which always computes a correct value
- NOT gate. If we have assigned a polynomial p to its predecessor, we take polynomial 1-p, which computes a correct value precisely when p computed a correct value
- it remains to handle OR gates – the only nontrivial case

Proof of Lemma 1 (⋆)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- we could take the polynomial: $1-(1-p_1)\cdot\ldots\cdot(1-p_k)$
- it works well whenever $p_1,\ldots,p_k$ worked well
- but its degree is too large: if $p_1,\ldots,p_k$ have degrees at most s, then its degree is ks – we rather need to obtain $\leq 2ts$, as then on the output gate we will have degree $(2t)^d$
- we thus have to proceed in a more clever way

Proof of Lemma 1 (∗)

Consider an <u>OR gate</u> of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- we could take the polynomial: $1-(1-p_1)\cdot\ldots\cdot(1-p_k)$
- it works well whenever $p_1,\ldots,p_k$ worked well
- but its degree is too large: if $p_1,\ldots,p_k$ have degrees at most s, then its degree is ks – we rather need to obtain $\leq 2ts$, as then on the output gate we will have degree $(2t)^d$
- we thus have to proceed in a more clever way
- in a moment, we will appropriately choose sets $S_1,\ldots,S_t \subseteq \{1,\ldots,k\}$
- we will take the polynomial:

$$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=\left(\sum_{j\in S_i} p_j\right)^2$$

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- in a moment, we will appropriately choose sets $S_1,\ldots,S_t \subseteq \{1,\ldots,k\}$
- we will take the polynomial:

$$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=\left(\textstyle\sum_{j\in S_i} p_j\right)^2$$

- p is proper, since $\{0^2,1^2,2^2\}=\{0,1\}$

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- in a moment, we will appropriately choose sets $S_1,\ldots,S_t \subseteq \{1,\ldots,k\}$
- we will take the polynomial:

$$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=\left(\textstyle\sum_{j\in S_i} p_j\right)^2$$

- p is proper, since $\{0^2,1^2,2^2\}=\{0,1\}$
- if degrees of $p_1,\ldots,p_k$ are $\leq s$, then the degree of p is $\leq 2ts$; then for the output gate of C we obtain degree $\leq (2t)^d$ – as required in the lemma
- it remains to see that p approximates well the value of the gate (for an appropriate choice of the sets $S_1,\ldots,S_t$)

Proof of Lemma 1 (⋆)

Consider an <u>OR gate</u> of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- we will take the polynomial:

$$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=(\textstyle\sum_{j\in S_i} p_j)^2$$

Fix some input (of the whole circuit C) on which all $p_1,\ldots,p_k$ give correct values. Let us randomly choose sets $S_1,\ldots,S_t\subseteq\{1,\ldots,k\}$ (every list of sets has the same probability)

Proof of Lemma 1 (∗)

Consider an <u>OR gate</u> of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

$$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=(\textstyle\sum_{j\in S_i}p_j)^2$$

Fix some input (of the whole circuit C) on which all $p_1,\dots,p_k$ give correct values. Let us randomly choose sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ (every list of sets has the same probability)

- If all p_j give value 0, then p also gives value 0 – correctly

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

  $p=1-(1-q_1)\cdot\dots\cdot(1-q_t)$ where $q_i=(\sum_{j\in S_i} p_j)^2$

- Fix some input (of the whole circuit C) on which all $p_1,\dots,p_k$ give correct values. Let us randomly choose sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ (every list of sets has the same probability)
- If all p_j give value 0, then p also gives value 0 – correctly
- If some p_j gives value 1, then for a chosen set S_i the polynomial q_i gives value 1 if in this set S_i the number of polynomials with value 1 is not divisible by 3. This is the case for at least half of choices of S_i. Thus the probability that for a random S_i the polyno-mial q_i gives value 1 is ≥ 0.5 (then the whole p also gives value 1).

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

  $p=1-(1-q_1)\cdot\ldots\cdot(1-q_t)$ where $q_i=(\sum_{j\in S_i} p_j)^2$

- Fix some input (of the whole circuit C) on which all $p_1,\dots,p_k$ give correct values. Let us randomly choose sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ (every list of sets has the same probability)
- If all p_j give value 0, then p also gives value 0 – correctly
- If some p_j gives value 1, then for a chosen set S_i the polynomial q_i gives value 1 if in this set S_i the number of polynomials with value 1 is not divisible by 3. This is the case for at least half of choices of S_i. Thus the probability that for a random S_i the polynomial q_i gives value 1 is ≥ 0.5 (then the whole p also gives value 1).
- Thus, if the sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ are chosen randomly, the probability that p will give an incorrect value is at most $1/2^t$

Proof of Lemma 1 (✶)

Consider an <u>OR gate</u> of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

  $p=1-(1-q_1)\cdot\ldots\cdot(1-q_t)$ where $q_i=(\sum_{j\in S_i} p_j)^2$

- For a <u>fixed</u> input, for which all $p_1,\dots,p_k$ give correct values, and for sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ <u>chosen randomly</u>, the probability that p gives an incorrect value is at most $1/2^t$

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\ldots,p_k$.

- we will take the polynomial:

  $p=1-(1-q_1)\cdot\ldots\cdot(1-q_t)$ where $q_i=(\sum_{j\in S_i}p_j)^2$
- For a fixed input, for which all $p_1,\ldots,p_k$ give correct values, and for sets $S_1,\ldots,S_t\subseteq\{1,\ldots,k\}$ chosen randomly, the probability that p gives an incorrect value is at most $1/2^t$
- Thus: for an input chosen randomly among those inputs for which all $p_1,\ldots,p_k$ give correct values, and for sets $S_1,\ldots,S_t\subseteq\{1,\ldots,k\}$ chosen randomly, the probability that p gives an incorrect value is at most $1/2^t$

Proof of Lemma 1 (⋆)

Consider an <u>OR gate</u> of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

  $p=1-(1-q_1)\cdot\ldots\cdot(1-q_t)$ where $q_i=(\sum_{j\in S_i} p_j)^2$

- For a <u>fixed</u> input, for which all $p_1,\dots,p_k$ give correct values, and for sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ <u>chosen randomly</u>, the probability that p gives an incorrect value is at most $1/2^t$
- Thus: for an input <u>chosen randomly</u> among those inputs for which all $p_1,\dots,p_k$ give correct values, and for sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ <u>chosen randomly</u>, the probability that p gives an incorrect value is at most $1/2^t$
- Thus: there <u>exist</u> sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ such that for an input <u>chosen randomly</u> among those inputs for which all $p_1,\dots,p_k$ give correct values, the probability that p gives an incorrect value is at most $1/2^t$

Proof of Lemma 1 (∗)

Consider an OR gate of fan-in k. To its arguments we have assigned some polynomials $p_1,\dots,p_k$.

- we will take the polynomial:

  $$p=1-(1-q_1)\cdot\ldots\cdot(1-q_t) \qquad \text{where} \qquad q_i=(\textstyle\sum_{j\in S_i} p_j)^2$$

- Thus: there exist sets $S_1,\dots,S_t\subseteq\{1,\dots,k\}$ such that for an input chosen randomly among those inputs for which all $p_1,\dots,p_k$ give correct values, the probability that p gives an incorrect value is at most $1/2^t$
- We take an arbitrary list of sets having this property
- The considered gate introduces a mistake on at most $2^n/2^t$ inputs
- Altogether, the value will be incorrect (for some gate) for at most $|C|\cdot 2^n/2^t$ inputs

  [THE END OF THE PROOF OF LEMMA 1]