Circuit complexity

IIC3242

Motivation

"One might imagine that PTIME $\neq$ NP, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ"

Karp and Lipton, 1982

Motivation

Turing machines: a **uniform model** of computation

- Evry size instance is treated with the same machine

Boolean circuits: a **nonuniform model** of computation:

- One circuit for each input size
- Similar to silicone chips (the same basically)

Even if PTIME $\neq$ NP, there might exist a chip that's efficient for SAT with 100,000 variables

Here we show that this is unlikely

A simpler model: easier lower bounds (maybe)

Boolean circuits

Definition

A Boolean circuit is a directed **acyclic** *graph where:*

- *Each node with no incoming edges has the label 0 or 1,*
- *Each node with an incoming edge has the label* $\neg$, $\wedge$ *or* $\vee$
- *Nodes with the label* $\neg$ *have one in-edge (***fan-in 1***), and the ones labelled* $\wedge$ *or* $\vee$ *have two (***fan-in 2***; can be arbitrary)*
- *There is an unique node with no out-edges*

Boolean circuits

Definition

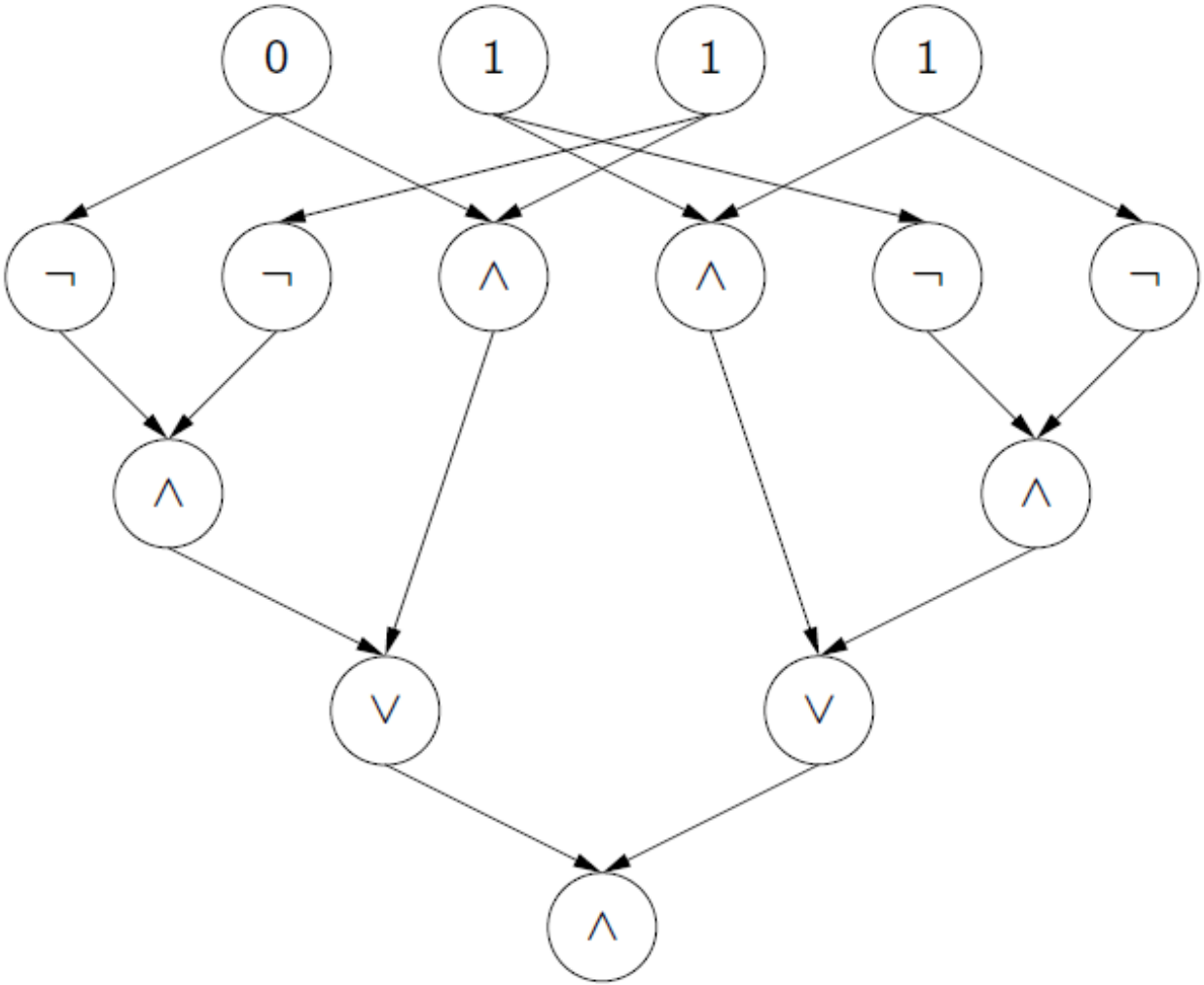
A Boolean circuit is a directed **acyclic** *graph where:*

- *Each node with no incoming edges has the label 0 or 1,*
- *Each node with an incoming edge has the label* $\neg$, $\wedge$ *or* $\vee$
- *Nodes with the label* $\neg$ *have one in-edge (***fan-in 1***), and the ones labelled* $\wedge$ *or* $\vee$ *have two (***fan-in 2***; can be arbitrary)*
- *There is an unique node with no out-edges*

Notation

- *Input node: Node with no in-edges*
- *Interior node: Nod with in-edges (also called a* **gate***)*
- *Output node: Node with no out-edges*
- *Note that* **fan-out** *is arbitrary*

An example of a boolean circuit

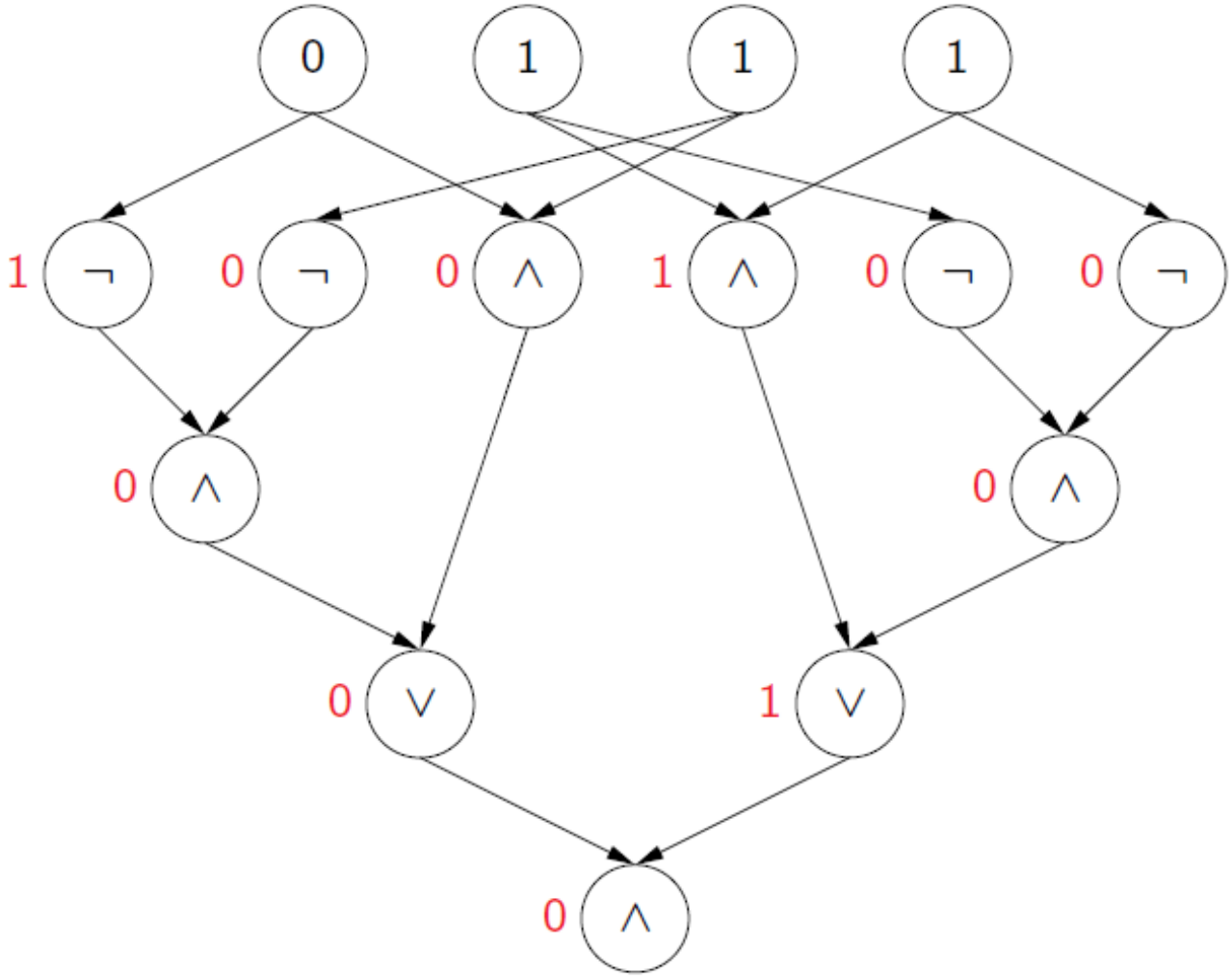


Boolean circuits as functions

Each Boolean circuit can be interpreted as a function:

- The input are the values of the input nodes
- Associate to each interior node u the result of evaluating the label of u over the values associated to nodes which have incoming edges to u
- The result is the value associated to the output node

Circuits as functions: an example



Circuits as functions

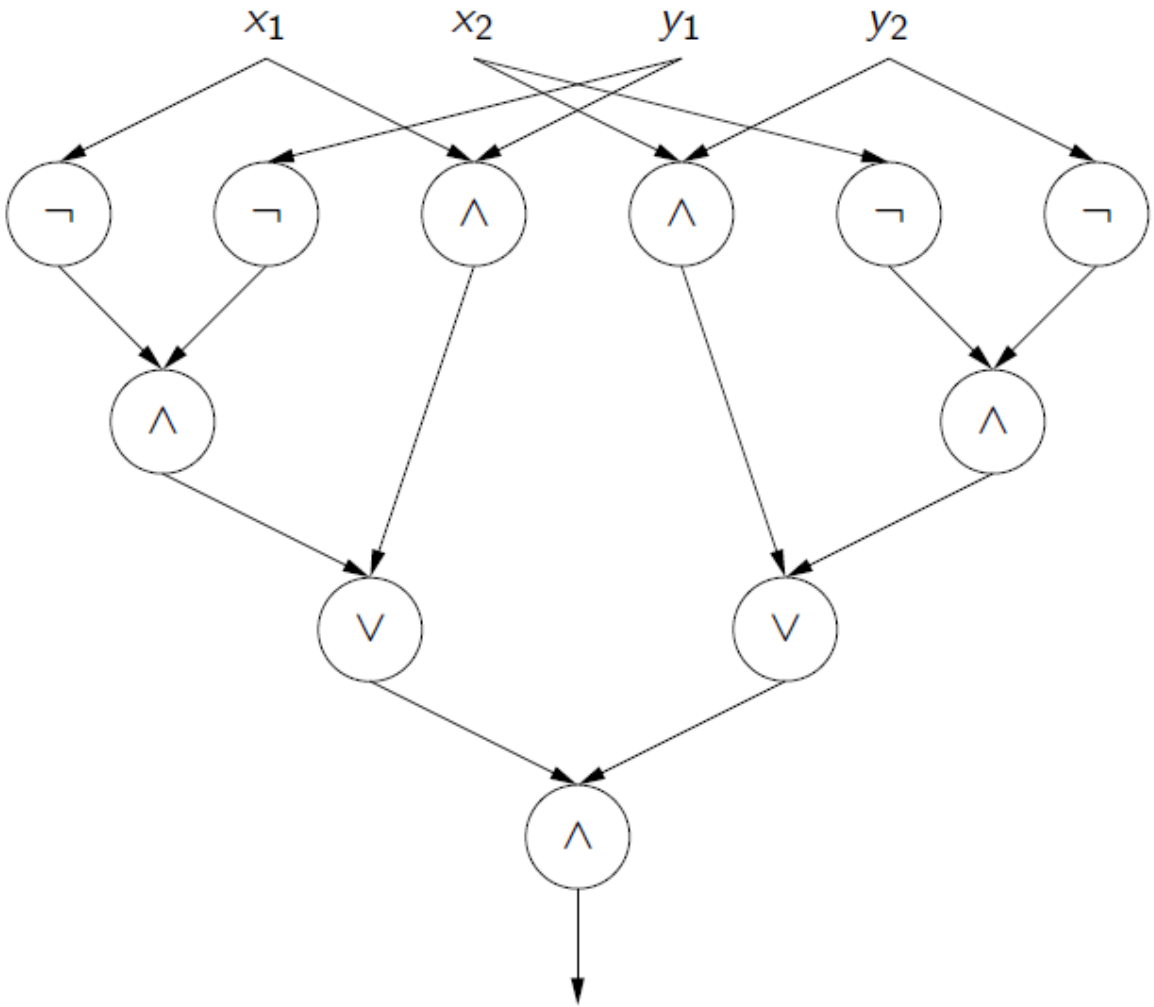
The circuit in our example represents the function:

$$f(x_1, x_2, y_1, y_2) = \begin{cases} 1 & x_1 = y_1 \text{ and } x_2 = y_2 \\ 0 & \text{otherwise} \end{cases}$$

To indicate the function a circuit represents:

- We replace the values 0,1 with variables
- We use and edge with no destination node to denote the output

Circuits as functions: an example



Circuit problems

Boolean circuit defines a value

Boolean circuit with variables defines a function

Theorem

For any function $f : \{0,1\}^n \to \{0,1\}^m$ *there is a boolean circuit with* n *inputs and* m *outputs that computes* f*.*

Prove this for homework (start small)

Measuring circuits

Definition

- *Size of a circuit:* *Number of nodes with the label* $\neg$, $\wedge$ *and* $\vee$ *(i.e. interior nodes)*
- *Depth of a circuit:* *Length of the* **longest** *path between any input node and the output node*

Measuring circuits

Definition

- *Size of a circuit:* *Number of nodes with the label* $\neg$, $\wedge$ *and* $\vee$ *(i.e. interior nodes)*
- *Depth of a circuit:* *Length of the* **longest** *path between any input node and the output node*

Example

For the circuit in our example the size is 11, and the depth 4.

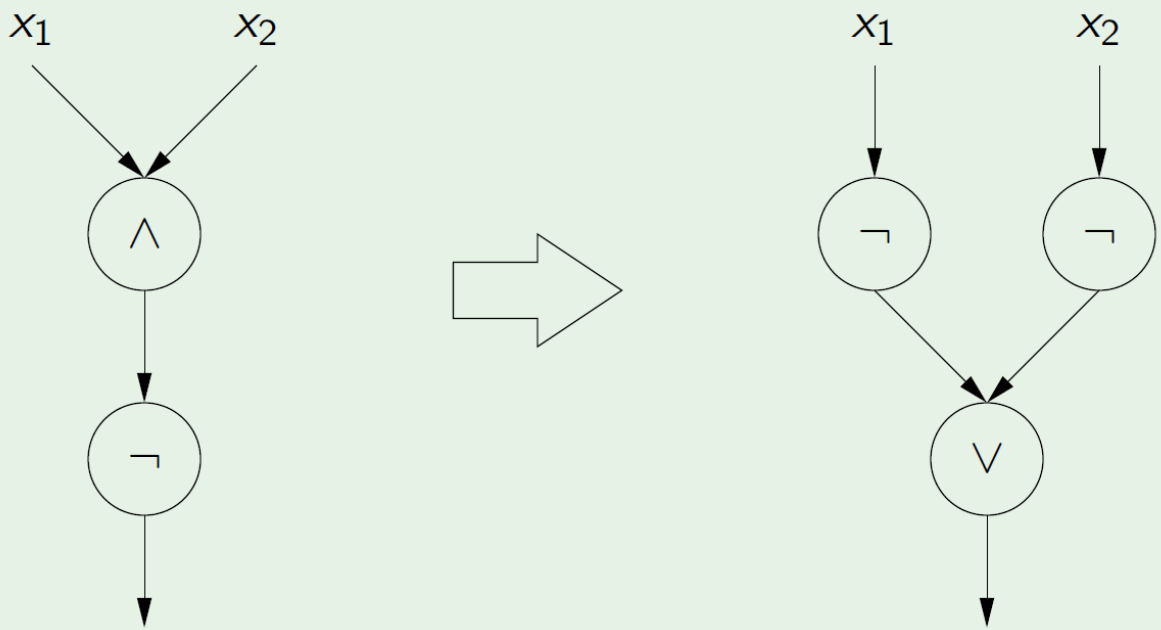
Circuit normal form

Using that $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ and $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ we can show that each circuit is equivalent to another circuit in which the negation appears only at the first level.

Circuit normal form

Using that $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ and $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ we can show that each circuit is equivalent to another circuit in which the negation appears only at the first level.

Example



Circuit normal form

Furthermore: The circuit obtained by using these transformations has the same depth as the original circuit, and at most double the amount of nodes.

From now on we assume that circuits use negation only at the first level,

- We (re)define the size of a circuit only considering the nodes labelled with $\wedge$ and $\vee$

Circuit normal form

Furthermore: The circuit obtained by using these transformations has the same depth as the original circuit, and at most double the amount of nodes.

From now on we assume that circuits use negation only at the first level,

- We (re)define the size of a circuit only considering the nodes labelled with $\wedge$ and $\vee$

Notation

- $size(C)$: *Size of the circuit* C
- $depth(C)$: *Depth of the circuit* C

Languages accepted by circuits

Given a circuit $C(\bar{x})$ with n inputs (we always have one output)

Definition

C defines the language:

$$\{w \in \{0,1\}^n \mid C(w) = 1\}$$

Languages accepted by circuits

Given a circuit $C(\bar{x})$ with n inputs (we always have one output)

Definition

C defines the language:

$$\{w \in \{0,1\}^n \mid C(w) = 1\}$$

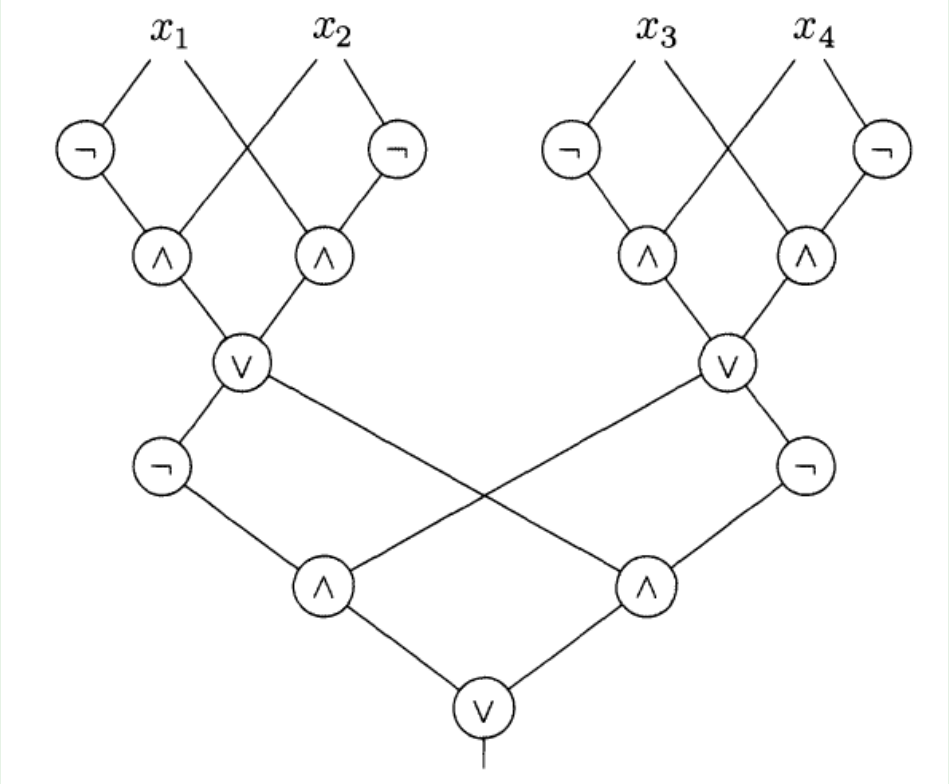
Example

In our example: $\{a_1 a_2 a_3 a_4 \in \{0,1\}^4 \mid a_1 = a_3 \text{ and } a_2 = a_4\}$

Languages accepted by circuits

Example

Consider the **parity function** $parity_n : \{0,1\}^n \to \{0,1\}$, which returns 1 if and only if the number of 1s in the input is odd. The following circuit computes $parity_4$:



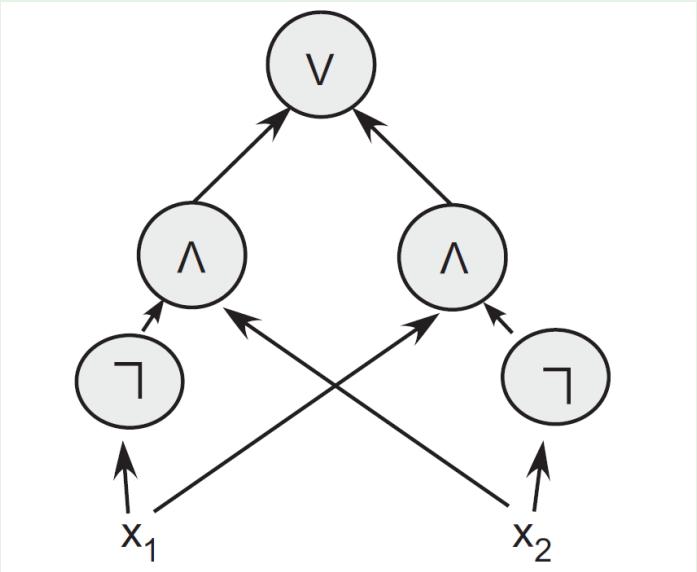
Another way of looking at this

Circuits can actually be viewed as straight line programs:

- Programs with no while, if, goto
- You execute operations one by one

Example

The following circuit computes $C(x_1, x_2) = 1$ iff $x_1 \neq x_2$:



How to express it by a straight line program?

Languages accepted by circuits

The language defined by a circuit is always finite.

- We have to use families of circuits to generate infinite languages.

Definition

A family of circuits $\{C_n\}_{n \geq 0}$ *accepts* $L \subseteq \{0,1\}^*$ *if:*

- C_n *has* n *inputs* (C_0 *is 0 or 1*)
- *For every word* w *of length* n: $w \in L$ *if and only if* $C_n(w) = 1$

Languages accepted by circuits

The notion of acceptance we have is too broad.

Proposition

Every language $L \subseteq \{0,1\}^*$ is accepted by some family of circuits $\{C_n\}$.

Languages accepted by circuits

The notion of acceptance we have is too broad.

Proposition

Every language $L \subseteq \{0,1\}^*$ is accepted by some family of circuits $\{C_n\}$.

Exercise

Prove the proposition.

Bounded circuits

We will need to bound the size of our circuits.

Definition

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$-**size circuit family** is a sequence $\{C_n\}_n$ of Boolean circuits, where C_n has n inputs and a single output, and $\text{size}(C_n) \leq T(n)$, for every n.
We say that a language L is in $\mathbf{SIZE}(T(n))$, if there there is an $O(T(n))$-size circuit family accepting L.

Example

The language $\{1^n \mid n \in \mathbb{N}\}$ is accepted by linear-sized circuit family.

$\{(m, n, m+n) \mid m, n \in \mathbb{N}\}$ also has linear-sized circuits.

Connection between time complexity and circuit complexity

Theorem

Let $T : \mathbb{N} \to \mathbb{N}$ be a function with $T(n) \geq n$. Then any $A \in DTIME(T(n))$ is also in **SIZE**$(T^2(n))$.

Proof: We already proved this when talking about PTIME-complexity.

That, this is the reduction showing that CIRCUIT-VALUE is PTIME-hard

Let us recall this

PTIME-completeness: circuits

For convenience we modify the tableau representation as follows:

- First and last column are #
- The alphabet of the tableau has the machine alphabet plus the symbols a_q, for each a in the alphabet of M and state q
- Head position is represented by a_q
- The last row always has $1_{q_{accept}}$ or $0_{q_{reject}}$ in the third position which signals that the machine accepts/rejects
- The machine always does precisely $t_M(n)$ steps for an input of length n
- Tableau accepts iff the machine does

PTIME-completeness: circuits

Tableau selfie:

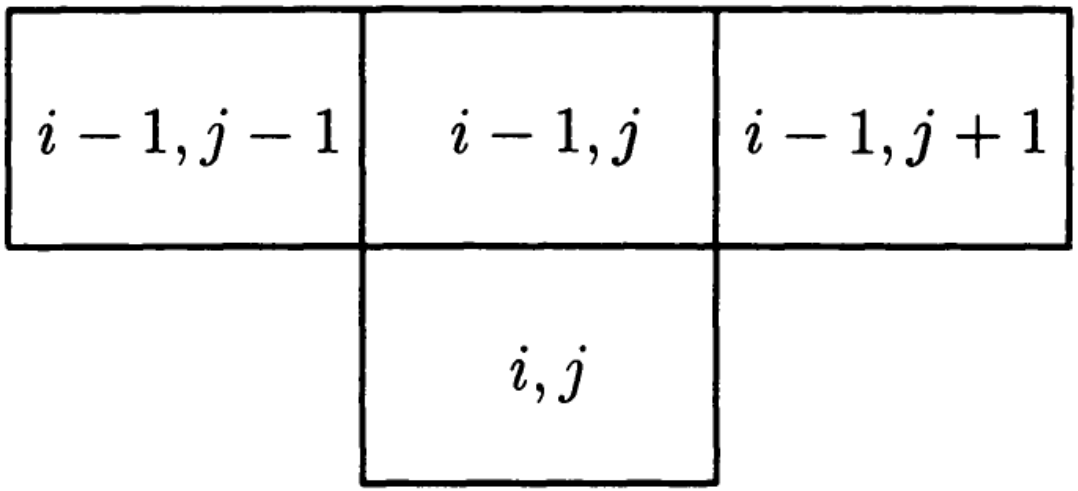
$$
\begin{array}{ccccccccc}
\# & \vdash & 0_{q_0} & 1 & 1 & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1_{q_1} & 1 & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1 & 1_{q_1} & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1 & 1 & 1_{q_1} & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1 & 1 & 1 & \mathtt{B}_{q_1} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1 & 1 & 1_{q_3} & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1 & 1_{q_3} & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0 & 1_{q_3} & 1 & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0_{q_3} & 1 & 1 & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash_{q_3} & 0 & 1 & 1 & 1 & \mathtt{B} & \mathtt{B} & \# \\
\# & \vdash & 0_{q_{reject}} & 1 & 1 & 1 & \mathtt{B} & \mathtt{B} & \#
\end{array}
$$

PTIME-completeness: circuits

For a string w we will construct a circuit $R(w)$ such that M accepts w iff $R(w)$ evaluates to 1

$T_{i,j}$ the cell in the tableau at position i, j

- The value of $T_{i,j}$ depends only on:
- The values of $T_{i-1,j-1}$, $T_{i-1,j}$ and $T_{i-1,j+1}$
- Some formalities with border cases



PTIME-completeness: circuits

Let A contain all the symbols appearing on the tableau:

- Each $a \in A$ is represented using a vector $(s_1, \ldots, s_m) \in \{0,1\}^m$
- Here $m : \lceil log_2 |A| \rceil$

Our tableau in position $T_{i,j}$ actually has entries $S_{i,j,\ell}$, with $\ell = 1 \ldots m$

Each $S_{i,j,\ell}$ depends on $3m$ binary entries:

- $S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}$
- $S_{i-1,j,1}, \ldots, S_{i-1,j,m}$
- $S_{i-1,j+1,1}, \ldots, S_{i-1,j+1,m}$

PTIME-completeness: circuits

That is, there exist binary functions $F_1, \ldots F_m$ such that for each i, j

$$S_{i,j,\ell} = F_\ell(S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}, S_{i-1,j,1}, \ldots, S_{i-1,j+1,m})$$

Every boolean function can be computed by a circuit:

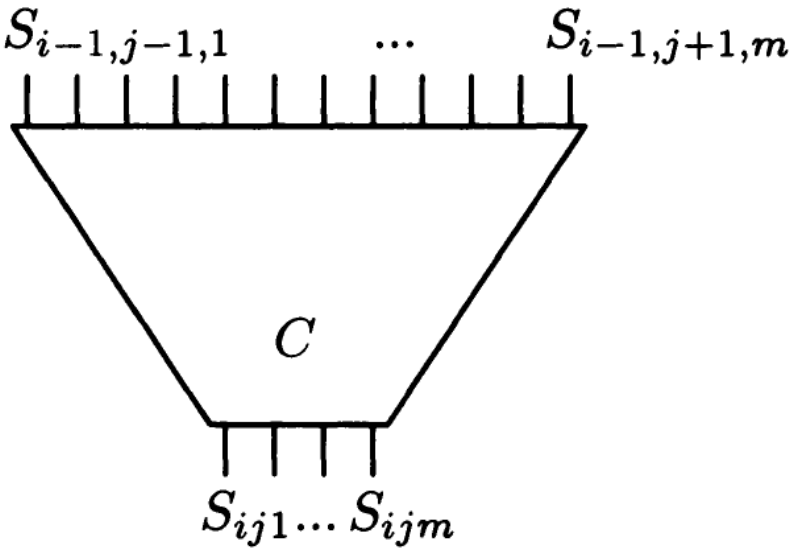
- There is C with $3m$ entries
- C computes (the binary encoding of) $T_{i,j}$ (for **any** i, j)
- When its input are encodings of $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$

C depends only on M and not on the input string

Remember valid windows from Cook-Levin

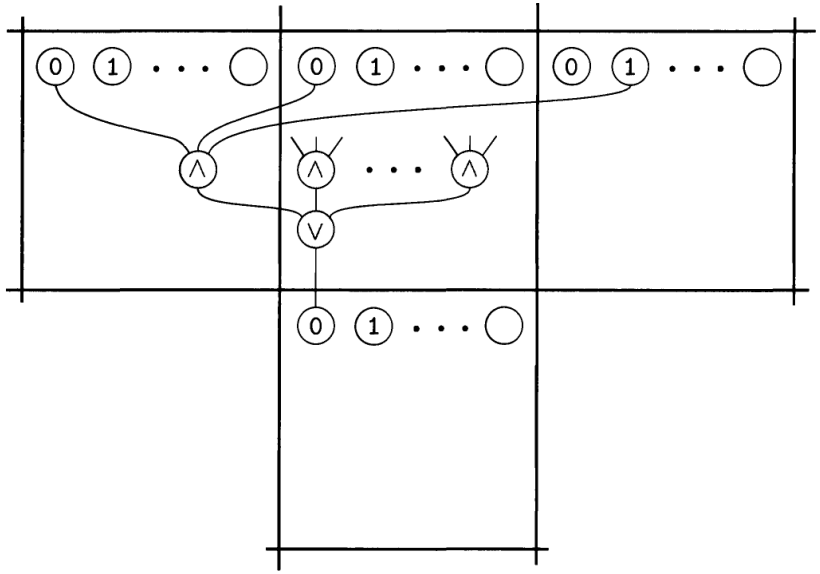
PTIME-completeness: circuits

Visually:



PTIME-completeness: circuits

Visually in a bit more detail:



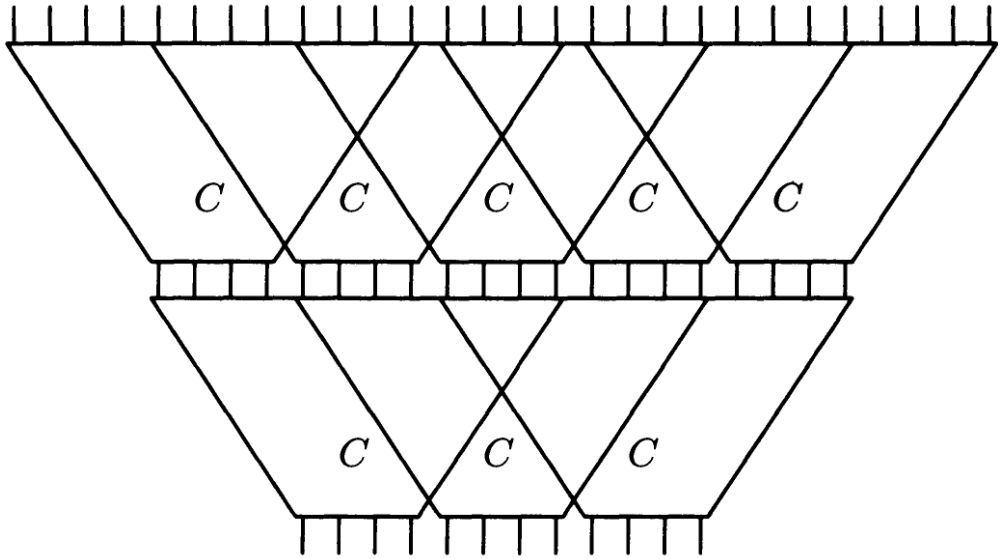
PTIME-completeness: circuits

For input word w we define the circuit $R(w)$:

- We have $t_M(|w|)^2$ copies of C
- Each copy is for one $T_{i,j}$
- The number is a bit smaller, as first row/column are not there
- $C_{i,j}$ the (i,j)-th copy of C
- The input gates of $C_{i,j}$ are the output gates of $C_{i-1,j-1}, C_{i-1,j}, C_{i-1,j+1}$
- **The input gates (coding the word w)connect to first row/column and last column**
- The output gate is the first output of the circuit $C_{t_M(|w|),3}$ (recall the initial assumption)

PTIME-completeness: circuits

The whole circuit:



PTIME-completeness: circuits

To finish:

- Show that M accepts W iff $R(w)$ is true
- Show that $R(w)$ is computable in LOGSPACE
- For latter recall that C is fixed
- And recall that to construct the input gates you only need to count up to $t_M(|w|)$

So we actually showed that the circuits can be constructed in LOGSPACE(remember for later).

The class P/poly

We want small-size circuits. The obvious candidate: polynomials

Definition (The class P/poly)

P/poly is the class of languages that are accepted by a polynomial-sized family of circuits. That is,

$$\text{P/poly} = \bigcup_c \textbf{SIZE}(n^c).$$

From the above we already know:

Corollary

PTIME $\subseteq$ *P/poly.*

P/poly and PTIME

What about the other direction?

We might run into some issues:

Theorem

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then $L \in P/poly$.

How do we prove this?

P/poly and PTIME

To see why this might be an issue consider:

$$\text{UHALT} = \{1^n \mid \text{ the binary expansion of } n \text{ encodes } \langle M, w\rangle \text{ such that } M \text{ halts with the input } w\}.$$

But this problem is undecidable (why)?

Theorem

$PTIME \subsetneq P/poly$.

Capturing PTIME with circuits

Why are circuits so powerful?

- We just want them to exist
- We do not need to build them (efficiently)

When we change this we get:

Definition (PTIME-uniform circuits)

A circuit family $\{C_n\}$ is PTIME-uniform, if there is a polynomial-time TM, which on input 1^n outputs the description of C_n.

Capturing PTIME with circuits

We can now capture PTIME

Theorem

A language L is PTIME-uniform, if and only if $L \in PTIME$.

Proof: For ($\Rightarrow$) we just compute the circuit and evaluate it

For the other direction, recall our proof of simulating machines with circuits. Since this was a reduction we know how to compute it for any n.

Capturing PTIME with circuits

We can be even stricter:

Definition (LOGSPACE-uniform circuits)

A circuit family $\{C_n\}$ is PTIME-uniform, if there is a LOGSPACETM, which on input 1^n outputs the description of C_n.

Theorem

A language L is LOGSPACE-uniform, if and only if $L \in PTIME$.

Why does this also hold?

PTIME/poly, NP and the polynomial hierarchy

What about NP? Can SAT have small circuits?

Theorem (Karp-Lipton)

If $NP \subseteq PTIME/poly$, then $PH = \Sigma_2^P$.

Intuition:

If we assume that poly-sized circuits for SAT:

- Both exist (like in the theorem assumption)
- And can be computed in poly-time
- Then we can solve SAT in poly-time

Σ_2^p: $\exists x \forall y \psi(x, y)$

- So we can guess the circuit (x) and use $\forall$ to verify it is good

Karp-Lipton theorem: more intuition

Key property of SAT: **self-reducibility**:

- There is a poly-time algorithm which, given a SAT instance, and a circuit for evaluating SAT, produces a valuation which is correct iff the formula is in SAT

Therefore, for a family $\{C_n\}$ of circuits for evaluating SAT, there is a family $\{D_n\}$ of circuits which output a satisfying assignment on true instances

Karp-Lipton theorem: more intuition

In the proof we show that $\Pi_2^p \subseteq \Sigma_2^p$

A problem in Π_2^p is defined by $\{z \mid \forall x \exists y \phi(x, y, z)\}$

Note that $\exists y \phi(x, y, z)$ is an instance of SAT, so:

- we replace it by $D_n(x, z)$ and get:
- $\forall x \phi(x, D_n(x, z), z)$

But in Σ_2^p I can guess the circuit:

- $\exists D \forall x \phi(x, D(x, z), z)$

Karp-Lipton theorem: proof

Proof: Let $\text{SAT}_{\bot,\top}$ be an extended version of SAT where the propositional formulas can include the symbols $\bot$ and $\top$.

- $\bot$ represents the value 0, and $\top$ the value 1

It is easy to see that $\text{SAT}_{\bot,\top}$ is NP-complete.

- Note that $\text{SAT}_{\bot,\top}$ is NP-hard because the identity is a reduction from SAT to $\text{SAT}_{\bot,\top}$

Karp-Lipton theorem: proof

By the hypothesis, there is a family of circuits $\{C_n\}_{n \geq 0}$ of polynomial size, that accepts $\text{SAT}_{\bot,\top}$.

- There exists a polynomial p such that $\text{size}(C_n) \leq p(n)$ for all $n \in \mathbb{N}$

Notation

We use φ to denote both a propositional formula, and a word in $\{0,1\}^*$ representing φ.

- *$|\varphi|$ is the length of the word representing the formula φ*

Karp-Lipton theorem: proof

By the hypothesis, there is a family of circuits $\{C_n\}_{n \geq 0}$ of polynomial size, that accepts $\text{SAT}_{\bot,\top}$.

- There exists a polynomial p such that $\text{size}(C_n) \leq p(n)$ for all $n \in \mathbb{N}$

Notation

We use φ to denote both a propositional formula, and a word in $\{0,1\}^*$ representing φ.

- *$|\varphi|$ is the length of the word representing the formula φ*

It follows that:

- φ is satisfiable if and only if $C_{|\varphi|}(\varphi) = 1$

Karp-Lipton theorem: proof

The following lemma is crucial for our proof.

Lemma (Self-reducibility of SAT)

There is a function H computable in polynomial time, such that for all formulas φ:

$\varphi \in SAT_{\bot,\top}$ *if and only if*

$H(\varphi, C_{|\varphi|})$ *is a valuation that satisfies* φ.

Karp-Lipton theorem: proof

The following lemma is crucial for our proof.

Lemma (Self-reducibility of SAT)

There is a function H computable in polynomial time, such that for all formulas φ:

$\varphi \in SAT_{\bot,\top}$ *if and only if*

$H(\varphi, C_{|\varphi|})$ *is a valuation that satisfies* φ.

How to prove the lemma?

- We will show a polynomial algorithm computing the function H with the desired properties

Karp-Lipton theorem: proof

We use the following notation:

- $\alpha[\frac{p}{\top}]$: Formula obtained by replacing p with $\top$ in α (and the same for $\bot$)

Karp-Lipton theorem: proof

We use the following notation:

- $\alpha[\frac{p}{\top}]$: Formula obtained by replacing p with $\top$ in α (and the same for $\bot$)

function $H(\varphi$: propositional formula, C: circuit)

$\sigma := \emptyset$

while φ has at least one propositional variable **do**

select a variable p from φ

if $C_{|\varphi|}(\varphi[\frac{p}{\top}]) = 1$

$\varphi := \varphi[\frac{p}{\top}]$

$\sigma := \sigma \cup \{p \mapsto 1\}$

else

$\varphi := \varphi[\frac{p}{\bot}]$

$\sigma := \sigma \cup \{p \mapsto 0\}$

return σ

Karp-Lipton theorem: proof

For the theorem we need to prove that $\Pi_2^P \subseteq \Sigma_2^P$.

Karp-Lipton theorem: proof

For the theorem we need to prove that $\Pi_2^P \subseteq \Sigma_2^P$.

- From this it follows that $\Pi_2^P = \Sigma_2^P$ and, therefore, $\text{PH} = \Sigma_2^P$. Why?

Karp-Lipton theorem: proof

For the theorem we need to prove that $\Pi_2^P \subseteq \Sigma_2^P$.

- From this it follows that $\Pi_2^P = \Sigma_2^P$ and, therefore, $\text{PH} = \Sigma_2^P$. Why?

Let $L \in \Pi_2^P$.

- We will show that $L \in \Sigma_2^P$

Karp-Lipton theorem: proof

For the theorem we need to prove that $\Pi_2^P \subseteq \Sigma_2^P$.

- From this it follows that $\Pi_2^P = \Sigma_2^P$ and, therefore, $\text{PH} = \Sigma_2^P$. Why?

Let $L \in \Pi_2^P$.

- We will show that $L \in \Sigma_2^P$

Given that $L \in \Pi_2^P$, there is a language $L_1 \in \text{NP}$ and a polynomial q such that for all $x \in \{0,1\}^*$:

$x \in L$ if and only if $\forall y \in \{0,1\}^*$ with $|y| = q(|x|)$, we have that $x\#y \in L_1$

Karp-Lipton theorem: proof

Since $\text{SAT}_{\bot,\top}$ is NP-complete, there is a reduction g from L_1 to $\text{SAT}_{\bot,\top}$.

Karp-Lipton theorem: proof

Since $\text{SAT}_{\bot,\top}$ is NP-complete, there is a reduction g from L_1 to $\text{SAT}_{\bot,\top}$.

- Assume that g is computable in time $r(n)$, with r some polynomial

Karp-Lipton theorem: proof

Since $\text{SAT}_{\bot,\top}$ is NP-complete, there is a reduction g from L_1 to $\text{SAT}_{\bot,\top}$.

- Assume that g is computable in time $r(n)$, with r some polynomial
- We also assume that $|w_1| = |w_2|$ implies $|g(w_1)| = |g(w_2)|$. Why can we assume this?

Karp-Lipton theorem: proof

Since $\text{SAT}_{\bot,\top}$ is NP-complete, there is a reduction g from L_1 to $\text{SAT}_{\bot,\top}$.

- Assume that g is computable in time $r(n)$, with r some polynomial
- We also assume that $|w_1| = |w_2|$ implies $|g(w_1)| = |g(w_2)|$. Why can we assume this?

We conclude that for all $x \in \{0,1\}^*$:

$x \in L$ if and only if $\forall y \in \{0,1\}^*$ with $|y| = q(|x|)$,

we have that $g(x\#y) \in \text{SAT}_{\bot,\top}$

Karp-Lipton theorem: proof

Combining this with self-reducibility of SAT (our lemma), we get that for all $x \in \{0,1\}^*$:

$x \in L$ if and only if $\forall y \in \{0,1\}^*$ with $|y| = q(|x|)$,
we have that $H(g(x\#y), C_{|g(x\#y)|})$
is a valuation satisfying $g(x\#y)$

Karp-Lipton theorem: proof

Size check:

$$\begin{aligned} \text{size}(C_{|g(x\#y)|}) &\leq p(|g(x\#y)|) \\ &\leq p(r(|x\#y|)) \\ &= p(r(|x| + |y| + 1)) \\ &= p(r(|x| + q(|x|) + 1)) \end{aligned}$$

Karp-Lipton theorem: proof

Size check:

$$\begin{aligned} \text{size}(C_{|g(x\#y)|}) &\leq p(|g(x\#y)|) \\ &\leq p(r(|x\#y|)) \\ &= p(r(|x| + |y| + 1)) \\ &= p(r(|x| + q(|x|) + 1)) \end{aligned}$$

Important: we assume that $p(n_1) \leq p(n_2)$ if $n_1 \leq n_2$

- Why can we assume this?

Karp-Lipton theorem: proof

Combining this with the lemma, we get that for all $x \in \{0,1\}^*$:

$x \in L$ if and only if

$\exists$ a circuit C where C has $|g(x\#0^{q(|x|)})|$ inputs,

$\text{size}(C) \leq p(r(|x| + q(|x|) + 1))$ and

$\forall y \in \{0,1\}^*$ with $|y| = q(|x|)$

we have that $H(g(x\#y), C)$

is a valuation satisfying $g(x\#y)$

Karp-Lipton theorem: proof

Combining this with the lemma, we get that for all $x \in \{0,1\}^*$:

$x \in L$ if and only if

$\exists$ a circuit C where C has $|g(x\#0^{q(|x|)})|$ inputs,

size$(C) \leq p(r(|x| + q(|x|) + 1))$ and

$\forall y \in \{0,1\}^*$ with $|y| = q(|x|)$

we have that $H(g(x\#y), C)$

is a valuation satisfying $g(x\#y)$

We can verify that $H(g(x\#y), C)$ is a valuation satisfying $g(x\#y)$ in polynomial time (CIRCUIT-SAT).

- Therefore $L \in \Sigma_2^P$. Why is this? □

Meyer's theorem

Karp-Lipton has a companion theorem:

Theorem (Meyer)

If EXPTIME $\subseteq$ P/poly, then EXPTIME $= \Sigma_2^p$.

Note that this implies that if PTIME = NP, then EXPTIME $\not\subseteq$ P/poly

- Since PTIME = NP implies PTIME $= \Sigma_2^P$
- So EXPTIME $\subseteq$ P/poly implies PTIME = EXPTIME
- But then time-hierarchy theorem...

CIRCUIT LOWER BOUNDS

Lower bounds by circuits

We have already stated that circuits compute any Boolean function.

To be more precise:

Theorem

Every function $f : \{0,1\}^* \to \{0,1\}$ *is computable by a family of circuits belonging to* $\mathbf{SIZE}(n2^n)$.

Proof: We first prove this with unbounded fan-in, and then comment how it works for bounded fan-in.

Defining functions by circuits

Fix a function f and input length n

Define $F_0 = \{x \in \{0,1\}^n \mid f(x) = 0\}$, and F_1 analogously

Then it holds (for inputs of length n):

$$f(x) = \bigvee_{x' \in F_1} [x = x']$$

(Here $[x = x']$ denotes a boolean expression true iff $x = x'$)

We have to write $[x = x']$ as simple expressions

Defining functions by circuits

The ith bit of x is denoted by x_i; then

$$[x = x'] \text{ iff } (\wedge_{i:x'_i=1} x_i) \wedge (\wedge_{i:x'_i=0} \neg x_i)$$

$$f(x) = \bigvee_{x' \in F_1} \left((\wedge_{i:x'_i=1} x_i) \wedge (\wedge_{i:x'_i=0} \neg x_i) \right)$$

This is a circuit of depth 2

- Negated gates are not counted, and we have unbounded fan-in
- The size is at worst $O(2^n)$
- You can do a similar thing with disjunction

Defining functions by circuits

To finish we need to do this with fan-in 2 gates

Every $\vee$-gate of in-degree k:

- Replaced by a tree of $k-1$ gates
- All of them of fan-in 2
- The depth is $logk$
- How is this done for e.g. $k = 3$?

From our equation for f we get a $O(n \cdot 2^n)$ circuit

Lower bounds for circuits

We know that all languages can be accepted by exponential circuits.

- Now we show that almost no languages are accepted by polynomial circuits

Lower bounds for circuits

We know that all languages can be accepted by exponential circuits.

- Now we show that almost no languages are accepted by polynomial circuits

Notation

For $n \geq 1$*:*

$$\mathcal{F}_n = \{f \mid f \text{ is a Boolean function with } n \text{ inputs}\}$$

$$\mathcal{P}_n = \{C \mid C \text{ is a circuit with } n \text{ inputs of the size less than } 2^{\frac{n}{4}}\}$$

Lower bounds by circuits

We will actually show something much stronger

- Not only is there a function not accepted by circuits of size at most $2^{\frac{n}{4}}$
- Almost no function with n inputs has circuits of size $2^{\frac{n}{4}}$

Lower bounds by circuits

We will actually show something much stronger

- Not only is there a function not accepted by circuits of size at most $2^{\frac{n}{4}}$
- Almost no function with n inputs has circuits of size $2^{\frac{n}{4}}$

Proposition

$$\lim_{n \to \infty} \frac{|\mathcal{P}_n|}{|\mathcal{F}_n|} = 0$$

Lower bounds for circuits

Proof: How many circuits with k nodes and n entries are there?

- Each gate is defined by its two predecessors, and its label
- We designate one output node
- Input nodes can be negated (so $2n$ of them)

Number of such circuits is at most: $(2 \cdot (k + 2n)^2)^k \cdot k$

Using $k \geq n$ we get $k \cdot (18 \cdot k^2)^k$

Taking now $k = 2^{\frac{n}{4}}$ we get:

$$|\mathcal{P}_n| \leq 2^{\frac{n}{4}} \cdot (18 \cdot 2^{\frac{n}{2}})^{2^{\frac{n}{4}}}$$

Lower bounds for circuits

So for big enough n we have:

$$\begin{aligned} |\mathcal{P}_n| &\leq 2^{\frac{n}{4}} \cdot 2^{\frac{n}{2} \cdot 2^{\frac{n}{4}}} \cdot 2^{10 \cdot 2^{\frac{n}{4}}} \\ &= 2^{\frac{n}{4} + \frac{n}{2} \cdot 2^{\frac{n}{4}} + 10 \cdot 2^{\frac{n}{4}}} \\ &\leq 2^{2^{\frac{n}{2}} + 2^{\frac{n}{4}} \cdot 2^{\frac{n}{4}} + 10 \cdot 2^{\frac{n}{2}}} \\ &\leq 2^{30 \cdot 2^{\frac{n}{2}}} \end{aligned}$$

Since $|\mathcal{F}_n| = 2^{2^n}$, we get that:

$$0 \leq \lim_{n \to \infty} \frac{|\mathcal{P}_n|}{|\mathcal{F}_n|} \leq \lim_{n \to \infty} \frac{2^{30 \cdot 2^{\frac{n}{2}}}}{2^{2^n}} = \lim_{n \to \infty} \frac{1}{2^{2^{\frac{n}{2}} \cdot (2^{\frac{n}{2}} - 30)}} = 0$$

Lower bounds for circuits

So there are many languages that require big circuits

If we show that some NP language needs big circuits, then PTIME $\neq$ NP

- These need not be exponential, only super-polynomial
- Since PTIME $\subsetneq$ P/poly we get the result

Unfortunately, we still don't know how to do this

But there is some progress

Lower bounds for circuits

For non-uniform complexity there is a hierarchy theorem as well:

Theorem (Non-uniform hierarchy)

For every k it holds that $\mathbf{SIZE}(n^k) \subsetneq \mathbf{SIZE}(n^{k+1})$.

The proof follows from the bounds we gave on hard to compute functions

It can also be generalized (see Arora-Barak)

Using this we can get some nice lower bounds

Lower bounds for circuits: Kannan's theorem

Here we prove that for every c there is a language in $\Sigma_2^p \cap \Pi_2^p$ which is not in $\mathbf{SIZE}(O(n^c))$

- This does not mean that $\Sigma_2^p \cap \Pi_2^p \subsetneq \mathrm{P/poly}$
- Since for every c the language is different

We start with something simpler

Theorem

For every c there is a language in $\Sigma_4^p \cap \Pi_4^p$ which is not in $\mathbf{SIZE}(n^c)$.

Proof: Fix a c. The idea is to use the non-uniform hierarchy theorem to find the desired language.

Lower bounds for circuits: Kannan's theorem

For each n let C_n be a circuit on n inputs such that:

- C_n is **lexicographically first** circuit on n inputs which computes a function that is not solvable by circuits of size n^c
- C_n is of size n^{c+1} (hierarchy theorem; n large enough)
- You have to count circuits (not direct from hierarchy)

Let L be the language decided by $\{C_n\}_n$

- Then $L \notin \mathbf{SIZE}(n^c)$

Next we show that $L \in \Sigma_4^p$

The idea is to guess the circuit and then verify that it is good.

Lower bounds for circuits: Kannan's theorem

Algorithm for x, where $|x| = n$:

$$x \in L \text{ if and only if}$$

1. There is a circuit C of size at most n^{c+1} such that
2. For all circuits C' (on n inputs) of size at most n^c, and for all circuits B (on n inputs) lexicographically smaller than C
3. There is an input $x' \in \{0,1\}^n$ with $C(x') \neq C'(x')$ and, there is a circuit B' of size at most n^c, such that
4. For all $w \in \{0,1\}^n$ it holds that $B(w) = B'(w)$ and
5. $C(x) = 1$.

By flipping the final condition we get that $\overline{L} \subseteq \Sigma_4^p$ □

Lower bounds for circuits: Kannan's theorem

Now we can prove:

Theorem (Kannan)

For every c there is a language in $\Sigma_2^p \cap \Pi_2^p$ which is not in $\mathbf{SIZE}(n^c)$

Proof: There are two cases:

1. If NP $\not\subseteq$ P/poly, then SAT$\in \Sigma_2^p \cap \Pi_2^p$, but SAT$\notin$ $\mathbf{SIZE}(n^c)$
2. If NP $\subseteq$ P/poly, then by Karp-Lipton we get that PH $= \Sigma_2^p = \Pi_2^p$, so the language from the previous theorem suffices.

□

WHAT ABOUT DEPTH?

Small depth circuits and parallel algorithms

So far we were restricting the size of the circuit, but what about **depth**?

- Depth corresponds to the time needed to evaluate the circuit
- Recall how we simulate TMs using circuits
- Also, depth/size equals parallel computation (wait a few slides)

We will introduce classes based on circuit depth

We also want them to be below P/poly

We will differentiate the classes based on the fan-in of operators

Complexity classes based on circuit depth

Definition (The complexity class AC^i)

For $i \geq 0$: A language $L \subseteq \{0,1\}^*$ is in the class AC^i if there is a family of circuits $\{C_n\}$ with **unbounded fan-in**, *a deterministic TM M and a constant k such that:*

1. *L is accepted by $\{C_n\}$*
2. *M generates $\{C_n\}$ and works in logarithmic space*
3. *$depth(C_n) \leq k \cdot (\log n)^i$*

The class AC is defined as $\bigcup_i AC^i$.

Note that each C_n is of polynomial size.

Complexity classes based on circuit depth

Definition (The complexity class NC^i)

For $i \geq 0$ the complexity class NC^i is defined just like AC^i, but now using circuits which have **fan-in** *2, meaning that each gate can have at most two incoming edges.*

The class NC is defined as $\bigcup_i NC^i$.

We know that each gate with polynomially many inputs can be simulated using a tree of depth $O(logp(n))$, so this automatically gives us:

$$\mathrm{NC}^i \subseteq \mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$$

NC, AC and parallel computation

Problems solved in any of these two classes are called highly parallelisable

The reason? They correspond to languages with efficient parallel algorithms.

An efficient parallel algorithm solves an instance of size n using a parallel computer with $n^{O(1)}$ processors in time $log^{O(1)} n$

One can show:

Theorem

A language has efficient parallel algorithm iff it is in $NC = AC$.

(See Arora - Barak for more details)

What can you solve like this?

Example

Parity $= \{x : x \text{ has an odd number of 1s}\}$ is in NC^1.

The circuit is a binary tree:

- The answer is at the root
- The left child computes the parity of first $|x|/2$ bits
- The right child the parity of the second part
- The gate at the top computes the parity of these two bits

AC^i: computational power

AC^0: Circuits of constant depth.

AC^i: computational power

AC^0: Circuits of constant depth.

Exercise

Prove that multiplication of boolean matrices is in AC^0 ($+$ is interpreted as $\vee$ and $\cdot$ as $\wedge$)

AC^i: computational power

AC^0: Circuits of constant depth.

Exercise

Prove that multiplication of boolean matrices is in AC^0 ($+$ is interpreted as $\vee$ and $\cdot$ as $\wedge$)

AC^1: Circuits of logarithmic depth. This will allow us to solve many problems.

AC^i: computational power

AC^0: Circuits of constant depth.

Exercise

Prove that multiplication of boolean matrices is in AC^0 ($+$ is interpreted as $\vee$ and $\cdot$ as $\wedge$)

AC^1: Circuits of logarithmic depth. This will allow us to solve many problems.

Exercise

1. Show that $L = \{w \in \{0,1\}^* \mid w \text{ has an odd number of 1s}\}$ is in AC^1
2. Show that checking if two nodes in a graph are connected is in AC^1

Addition is in AC^0

We will show that you can compute the sum of two numbers in AC^0, that is, using circuits of constant depth.

We want to construct a circuit $C(x_n, \ldots, x_1, y_n, \ldots, y_1)$ which:

- Takes as input two binary numbers $\bar{x} = x_n \cdots x_1$ and $\bar{y} = y_n \cdots y_1$
- Has $n + 1$ output which correspond to the sum of the two numbers: $z_{n+1} z_n \cdots z_1$

Addition is in AC^0

Notation

$$
\begin{aligned}
AND_i &= x_i \wedge y_i \\
OR_i &= x_i \vee y_i \\
XOR_i &= (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i)
\end{aligned}
$$

Addition is in AC^0

Notation

$$\begin{aligned} AND_i &= x_i \wedge y_i \\ OR_i &= x_i \vee y_i \\ XOR_i &= (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i) \end{aligned}$$

Let $r_0 = 0$ and r_i ($i \in [1, n]$) the ith carry bit of the sum of $\bar{x}$ and $\bar{y}$

Therefore for $i \geq 1$:

$$r_i = \text{AND}_i \vee (\text{OR}_i \wedge r_{i-1})$$

Addition is in AC^0

Expanding the equation once we obtain:

$$r_i = \text{AND}_i \vee (\text{OR}_i \wedge \text{AND}_{i-1}) \vee (\text{OR}_i \wedge \text{OR}_{i-1} \wedge r_{i-2})$$

If we continue expanding:

$$\begin{aligned}
r_1 &= \text{AND}_1 \\
r_2 &= \text{AND}_2 \vee (\text{OR}_2 \wedge \text{AND}_1) \\
&\cdots \\
r_n &= \text{AND}_n \vee (\text{OR}_n \wedge \text{AND}_{n-1}) \vee (\text{OR}_n \wedge \text{OR}_{n-1} \wedge \text{AND}_{n-2}) \vee \cdots \\
&\qquad \cdots \vee (\text{OR}_n \wedge \text{OR}_{n-1} \wedge \cdots \wedge \text{OR}_2 \wedge \text{AND}_1).
\end{aligned}$$

Addition is in AC^0

We can use the carry bit to define the result of the sum:

$$z_i \quad = \quad (\neg \mathrm{OR}_i \wedge r_{i-1}) \vee (\mathrm{XOR}_i \wedge \neg r_{i-1}) \vee (\mathrm{AND}_i \wedge r_{i-1})$$

In particular: $z_1 = \mathrm{XOR}_1$ and $z_{n+1} = r_n$.

Addition is in AC^0

We can use the carry bit to define the result of the sum:

$$z_i \quad = \quad (\neg \mathrm{OR}_i \wedge r_{i-1}) \vee (\mathrm{XOR}_i \wedge \neg r_{i-1}) \vee (\mathrm{AND}_i \wedge r_{i-1})$$

In particular: $z_1 = \mathrm{XOR}_1$ and $z_{n+1} = r_n$.

We can use the definition of r_i and z_i to construct the circuit.

- What is the depth of the circuit?

Addition is in AC^0

We can use the carry bit to define the result of the sum:

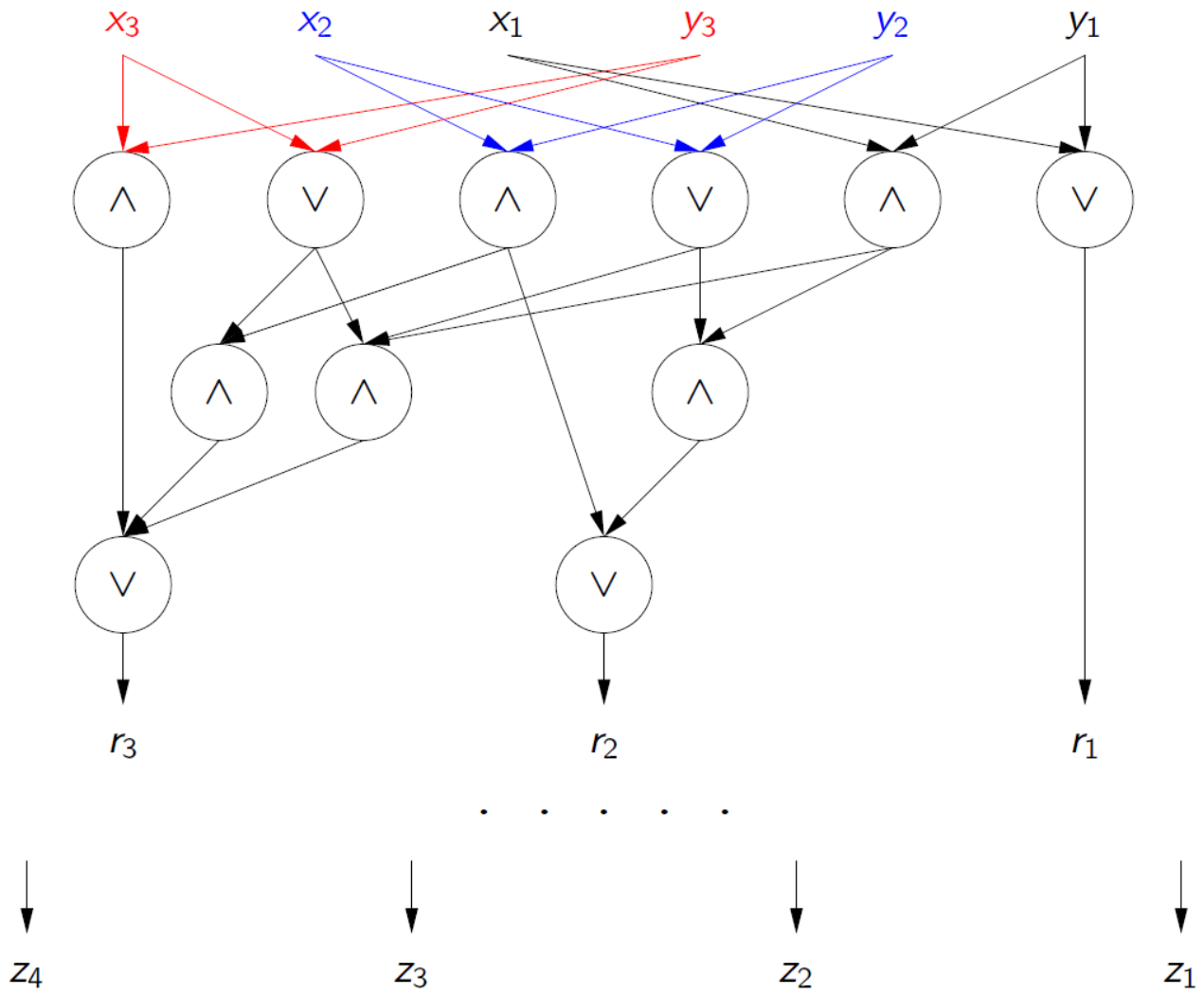
$$z_i \quad = \quad (\neg \mathrm{OR}_i \wedge r_{i-1}) \vee (\mathrm{XOR}_i \wedge \neg r_{i-1}) \vee (\mathrm{AND}_i \wedge r_{i-1})$$

In particular: $z_1 = \mathrm{XOR}_1$ and $z_{n+1} = r_n$.

We can use the definition of r_i and z_i to construct the circuit.

- What is the depth of the circuit?
- Let's look at an example: $C(x_3, x_2, x_1, y_3, y_2, y_1)$

Addition is in AC^0



Regular languages via circuits

One can also show:

Theorem

If L is a regular language, then $L \in NC^1$.

Regular languages via circuits

One can also show:

Theorem

If L is a regular language, then $L \in NC^1$.

Exercise

Prove this.

- Show first that the language of the expression $(01)^*$ is in NC^1
- Then extend the proof to the general case

AC, NC and other classes

By definition we know that $\mathsf{AC} = \mathsf{NC} \subseteq \mathsf{PTIME} \subsetneq \mathsf{P/poly}$

We now relate space complexity and circuit depth:

Theorem

If $s(n) \geq \log n$, then any $L \in \mathit{NSPACE}(s)$ can be decided by circuits of depth $O(s(n)^2)$ when fan-in 2 is allowed, and in depth $O(s(n))$, when using unbounded fan-in.

Proof sketch: We use reachability in the configuration graph.

Let M be a machine deciding L in space t. Then there are $N(n) = 2^{O(s(n))}$ configurations on inputs of length n.

Nondeterministic space by circuits

Fix n, we construct C_n as follows:

On input $x \in \{0,1\}^n$ the circuit does the following:

1. Construct $N \times N$ matrix M_x where $M_x(i,j) = 1$ iff M can move from configuration i to the configuration j in one step (on input x)
2. Compute the transitive closure of M_x (and check if there is a path from the initial to the final configuration)

Nondeterministic space by circuits

To see why this works observe that:

- The matrix M_x is constructed in constant depth (it depends on the head position only)
- Transitive closure is the same as $(M_x \vee I)^{N(n)}$, so we use fast squaring
- This is a $logN = O(s(n))$ algorithm (with unbounded fan-in)
- For fan-in 2 each multiplication is in $logN$ depth as well, so total $log^2 N = O(s(n)^2)$

AC, NC and other classes

This allows us to get:

Theorem

$AC^0 \subseteq NC^1 \subseteq LOGSPACE \subseteq NLOGSPACE \subseteq AC^1$

Exercise

How to evaluate NC^1 in LOGSPACE?

AC, NC and other classes

This allows us to get:

Theorem

$AC^0 \subseteq NC^1 \subseteq LOGSPACE \subseteq NLOGSPACE \subseteq AC^1$

Exercise

How to evaluate NC^1 in LOGSPACE?

Theorem (Furst-Saxe-Sipser)

$AC^0 \subsetneq LOGSPACE$

AC, NC and other classes

This allows us to get:

Theorem

$AC^0 \subseteq NC^1 \subseteq LOGSPACE \subseteq NLOGSPACE \subseteq AC^1$

Exercise

How to evaluate NC^1 in LOGSPACE?

Theorem (Furst-Saxe-Sipser)

$AC^0 \subsetneq LOGSPACE$

Idea: Parity is in LOGSPACE $\setminus$ AC^0. Also, parity is in NC^1.

SPARSE SETS AND P vs NP

Turing machines for P/poly

Can we define P/poly in term of TMs?

Definition (TMs which "take advice")

Let $T, a : \mathbb{N} \to \mathbb{N}$ be two functions. The class of languages decidable by time-$T(n)$ machine with $a(n)$-bits of advice, denoted by DTIME$(T(n)/a(n))$ s defined as the set of all languages L, where:

- there is a sequence $\{\alpha_n\}_n$ of strings in $\{0,1\}^{a(n)}$; and
- a TM M, such that for all $x \in \{0,1\}^n$:

$$x \in L \text{ iff } M(x, \alpha_n) \text{ accepts in time } O(T(n)).$$

Note that the machine is $T(n)$ over its first input.

Turing machines for P/poly

This can characterise P/poly:

Theorem

$L \in P/poly$ iff there are two polynomials p and q, such that $L \in DTIME(p(n)/q(n))$.

Proof sketch: If $L \in$ P/poly, then it has polynomial circuit family $\{C_n\}_n$ computing it.

The advice string is then (the description of) C_n, and our machine on input x computes $C_n(x)$, which we know is in PTIME

Turing machines for P/poly

For the other direction assume $L \in \text{DTIME}(p(n)/q(n))$.

We basically use the fact that each machine can be simulated by circuits.

We just hard-wire the advice string as part of the input:

- For each n there is a poly-size circuit D_n with:
- $D_n(x, \alpha_n) = M(x, \alpha_n)$, for any $x \in \{0,1\}^n$
- $C_n(x) = D_n(x, \alpha_n)$ (we hard-wire α_n)

□

Example

What is the advice string for UHALT?

Sparse languages

Recall the UHALT language:

$$\text{UHALT} = \{1^n \mid \text{ the binary expansion of } n \text{ encodes } \langle M, x\rangle \text{ such that } M \text{ halts with the input } w\}.$$

This language is in P/poly, because it has very few words in it

- It is possible to generalise this idea

Sparse languages

Recall the UHALT language:

$$\text{UHALT} = \{1^n \mid \text{ the binary expansion of } n \text{ encodes } \langle M, x\rangle \text{ such that } M \text{ halts with the input } w\}.$$

This language is in P/poly, because it has very few words in it

- It is possible to generalise this idea

Definition

A language L is sparse if there is a polynomial $p(n)$ such that for all $n \geq 0$:

$$|\{w \in L \mid |w| = n\}| \quad \leq \quad p(n)$$

The class P/poly and sparse languages

Theorem

Every sparse language belongs to PTIME/poly.

Exercise

Prove this.

Is there a stronger connection between the two classes?

The class P/poly and sparse languages

Theorem

$L \in P/poly$ if and only if there is a sparse set S such that $L \in PTIME^S$.

Proof: [$\Leftarrow$] Assume L is decided by a polynomial machine M with a sparse oracle S.

Let $q(n)$ be the running time of M, and $p(n)$ the sparseness-polynomial for S.

On inputs of length n, the query to S can be at most of size $q(n)$.

But there are at most $p(q(n))$ strings of this size in S.

The class P/poly and sparse languages

[$\Leftarrow$ continued] We will use the machine definition of P/poly

- Our advice function is just a string α_n containing all $q(p(n))$ strings in S that M might potentially query
- (Technically you need the shorter ones as well)
- But this is all polynomial

So our machine can do the same as M, but when M queries S, we just look at α_n to see if the word is there

Since M was polynomial, so is our machine, so $L \in$ P/poly

The class P/poly and sparse languages

[$\Rightarrow$] Take any $L \in \text{P/poly}$

Then there are polynomials p, q such that $L \in \text{DTIME}(p(n)/q(n))$

That is, there is:

- A sequence α_n of strings in $\{0,1\}^{q(n)}$
- A TM N s.t. $N(x, \alpha_{|x|})$ accepts iff $x \in L$
- And the machine runs in $p(|x|)$ (the first input)

We define the oracle S and a machine M such that:

- M can discover α_n (bit by bit)
- Using a sparse S as an oracle

With this M we can then simulate N, since we know the whole α_n

The class P/poly and sparse languages

[$\Rightarrow$ continued] How do we discover α_n?

$$S = \{1^n 0 p \mid n \geq 0, \text{ and } p \text{ is a prefix of } \alpha_n\}$$

S is sparse:

- The number of elements of length n is n

Using S as an oracle we can discover α_n:

- Start by asking the oracle if $1^n 00$ and $1^n 01$ are in S
- Let $1^n 0b$ be the one that is (it is unique)
- Now ask if $1^n 0b0$ and $1^n 0b1$ are in S
- Do this $q(n)$ times (total $2 \cdot q(n)$ queries)

Now on input of length n run M to discover α_n, and then just simulate what N does. $\square$

Sparse sets, NP and Karp-Lipton

Theorem

If some sparse language is NP-complete, then $PH = \Sigma_2^P$.

Proof: Assume that $L \subseteq \{0,1\}^*$ is sparse and NP-complete.

Sparse sets, NP and Karp-Lipton

Theorem

If some sparse language is NP-complete, then $PH = \Sigma_2^P$.

Proof: Assume that $L \subseteq \{0,1\}^*$ is sparse and NP-complete.

We will show that this implies NP $\subseteq$ PTIME/poly.

Sparse sets, NP and Karp-Lipton

Theorem

If some sparse language is NP-complete, then $PH = \Sigma_2^P$.

Proof: Assume that $L \subseteq \{0,1\}^*$ is sparse and NP-complete.

We will show that this implies NP $\subseteq$ PTIME/poly.

- By the Karp-Lipton theorem: $PH = \Sigma_2^P$

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Since L is NP-complete, we have that $L^\star$ is also NP-complete.

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Since L is NP-complete, we have that $L^\star$ is also NP-complete.

- Why?

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Since L is NP-complete, we have that $L^\star$ is also NP-complete.

- Why?

Since L is sparse, we also have that $L^\star$ is sparse.

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Since L is NP-complete, we have that $L^\star$ is also NP-complete.

- Why?

Since L is sparse, we also have that $L^\star$ is sparse.

- We know that $|\{w \in L \mid |w| = n\}| \leq p(n)$

Sparse sets, NP and Karp-Lipton

Let $L^\star$ be the following language:

$$L^\star \quad = \quad \{0^n 1 w \mid w \in L \text{ and } n \geq 0\}$$

Since L is NP-complete, we have that $L^\star$ is also NP-complete.

- Why?

Since L is sparse, we also have that $L^\star$ is sparse.

- We know that $|\{w \in L \mid |w| = n\}| \leq p(n)$
- Therefore:

$$|\{w \in L^\star \mid |w| = n\}| \quad \leq \quad \sum_{i=0}^{n-1} p(n - (i+1))$$

Sparse sets, NP and Karp-Lipton

Since $L^\star$ is sparse: there is a family of circuits $\{C_n\}_{n \geq 0}$ of polynomial size that accepts $L^\star$.

- Using this we will prove that NP $\subseteq$ PTIME/poly

Sparse sets, NP and Karp-Lipton

Since $L^\star$ is sparse: there is a family of circuits $\{C_n\}_{n\geq 0}$ of polynomial size that accepts $L^\star$.

- Using this we will prove that NP $\subseteq$ PTIME/poly

Let $L' \subseteq \{0,1\}^*$ be an arbitrary language in NP.

Since L is NP-complete, there is some $f : \{0,1\}^* \to \{0,1\}^*$ such that:

- For every $w \in \{0,1\}^*$: $w \in L'$ if and only if $f(w) \in L$
- f can be computed in time $q(n)$, for a polynomial q

Sparse sets, NP and Karp-Lipton

Define $g : \{0,1\}^* \to \{0,1\}^*$ as:

$$g(w) \quad = \quad 0^{q(|w|)-|f(w)|}1f(w)$$

It holds that:

Sparse sets, NP and Karp-Lipton

Define $g : \{0,1\}^* \to \{0,1\}^*$ as:

$$g(w) \quad = \quad 0^{q(|w|)-|f(w)|}1f(w)$$

It holds that:

- For every $w \in \{0,1\}^*$: $w \in L'$ if and only if $g(w) \in L^\star$

Sparse sets, NP and Karp-Lipton

Define $g : \{0,1\}^* \to \{0,1\}^*$ as:

$$g(w) \quad = \quad 0^{q(|w|)-|f(w)|}1f(w)$$

It holds that:

- For every $w \in \{0,1\}^*$: $w \in L'$ if and only if $g(w) \in L^\star$
- g can be computed in polynomial time

Sparse sets, NP and Karp-Lipton

Define $g : \{0,1\}^* \to \{0,1\}^*$ as:

$$g(w) \quad = \quad 0^{q(|w|)-|f(w)|}1f(w)$$

It holds that:

- For every $w \in \{0,1\}^*$: $w \in L'$ if and only if $g(w) \in L^\star$
- g can be computed in polynomial time
- For every $w \in \{0,1\}^*$: $|g(w)| = q(|w|) + 1$

Sparse sets, NP and Karp-Lipton

Define $g : \{0,1\}^* \to \{0,1\}^*$ as:

$$g(w) \quad = \quad 0^{q(|w|)-|f(w)|}1f(w)$$

It holds that:

- For every $w \in \{0,1\}^*$: $w \in L'$ if and only if $g(w) \in L^\star$
- g can be computed in polynomial time
- For every $w \in \{0,1\}^*$: $|g(w)| = q(|w|) + 1$
  - In particular: if $|w_1| = |w_2|$, then $|g(w_1)| = |g(w_2)|$

Sparse sets, NP and Karp-Lipton

Therefore: There is a family of circuits $\{D_n\}_{n \geq 0}$ of polynomial size accepting L'.

- There is a family of circuits $\{E_n\}_{n \geq 0}$ of polynomial size computing g.
  - Why?
- Each circuit D_n is constructed by composing E_n with $C_{q(n)+1}$.
  - How is this done?

We conclude that $L' \in \text{PTIME/poly}$. □

Mahaney's theorem

In fact something much stronger is true:

Theorem (Mahaney)

There exists a sparse language that is NP-hard if and only if PTIME = NP.

Proof: [$\Leftarrow$] If PTIME = NP, then there is a PTIME algorithm for SAT.

So we can reduce SAT to $\{1\}$, by mapping positive instances to 1, and the rest to 0.

[$\Rightarrow$] This direction is much more interesting.

Mahaney's theorem: the proof

Assume that some NP-hard language is poly-time reducible to a sparse set S

Then any language in NP is poly-time reducible to S (reductions compose)

Consider the following language:

$$\text{LSAT} = \{(\varphi, x) \mid \varphi \text{ is a formula with } n \text{ variables }, x \in \{0,1\}^n, \text{ and there is } y \leq x \text{ which satisfies } \varphi\}$$

- Here $y \leq x$ means in lexicographical order
- A valuation for n variables is just a string in $\{0,1\}^n$

Mahaney's theorem: the proof

Note that $\varphi \in \text{SAT}$ iff $(\varphi, 1^n) \in \text{LSAT}$

- Therefore LSAT is NP-complete
- So there is a polynomial reduction f from LSAT to S

Let $p'(n)$ be a polynomial bounding f:

- Then $|f(\varphi, x)| \leq p'(2 \cdot |\varphi|)$, since the number of variables is less than $|\varphi|$
- So there is a polynomial $p(|\varphi|)$ which bounds the length of the output of f on (φ, x)

By sparseness of S we get that:

- $q(n) = |S \cap \{0,1\}^{\leq p(n)}|$ is also polynomial

Mahaney's theorem: the proof

We now give a PTIME algorithm for SAT, thus showing that PTIME = NP

The idea is to do a BFS over the tree of assignments where:

- The root is the empty string
- A node labelled v has children $v0$ and $v1$
- The depth is n

A node v corresponds to a valuation where the first $|v|$ variables are set to v

Notation: φ has an assignment lexicographically at most v, with $|v| < n$, if this holds for $v1^{n-|v|}$

Mahaney's theorem: the proof

We use $f(v)$ to denote $f(\varphi, v1^{n-|v|})$

The following is a PTIME algorithm for SAT:

On input φ, with n variables:

1. Set $\text{VAL}_0 = \{\varepsilon\}$
2. for $i = 1$ to $n - 1$ do:
3. Let $\text{VAL}_i = \{v0, v1 \mid v \in \text{VAL}_{i-1}\}$ [children of VAL_{i-1}]
4. If $|\text{VAL}_i| > q(|\varphi|)$ then run PRUNE(VAL_i)
5. Output 1 iff one of the children of VAL_{n-1} gives a satisfying assignment

The procedure PRUNE is given below

Mahaney's theorem: the proof

Let V be a set; then

PRUNE(V):

- Compute $Z_V = \{f(v) \mid v \in V\}$
- Now use repeatedly the following rules:
  1. If $f(v_1) = f(v_2)$ and v_2 is lexicographically greater, then remove v_2 from V
  2. If Z_V contains more than $q(|\varphi|)$ distinct elements, remove the lexicographically smallest one

Clearly the whole thing is PTIME, since we have n levels and consider only $2 \cdot q(|\varphi|)$ nodes at every level

We next prove that the algorithm is also correct; i.e. it outputs 1 iff $\varphi \in$ SAT

Mahaney's theorem: the proof

A node v is an ancestor of a satisfying assignment w if v is a prefix of w

We prove that if φ is satisfiable, then VAL_i contains an ancestor of a lexicographically smallest satisfying assignment

This is enough for correctness, since for $\varphi \notin \text{SAT}$, the last step will output 0

We prove this claim by induction

It is clearly true for VAL_0

If this is true for VAL_{i-1}, then it is obviously true for VAL_i before we run $\text{PRUNE}(\text{VAL}_i)$

Mahaney's theorem: the proof

To prove that this is true after we run PRUNE(VAL_i) observe the following:

1. If $v_1 \leq v_2$ and $f(v_1) = f(v_2)$, then v_2 is removed. This is a problem only if v_2 is an ancestor of the smallest satisfying assignment. But then $f(v_2) \in S$, so also $f(v_1) \in S$, so φ has a satisfying assignment smaller than v_1.

2. Let $V = \text{VAL}_i$. If Z_V contains more than $q(|\varphi|)$ elements, then there is $z \in Z_V$ such that $z \notin S$. So for some $v \in V$ our formula φ does not have a satisfying assignment at most v. If v_0 is the lexicographically smallest element of V, then φ also does not have a satisfying assignment at most $v_0 \leq v$, so removing v_0 doesn't change anything.

Sparse sets and PTIME

There is also a version of Mahaney's theorem for PTIME and LOGSPACE:

Theorem (Cai and Sivakumar, 1999)

If there exists a sparse PTIME-complete problem, then PTIME = LOGSPACE.