Boolean circuits and efficient addition

Corentin Barloy Michael Walter Thomas Zeume

RUHR UNIVERSITÄT BOCHUM RUB

Introduction

How can we measure the ressources used during a computation?

Introduction

How can we measure the ressources used during a computation?

- Heavily depends on the hardware (NASA supercomputer, phone, a student, ...)
- Heavily depends on the language (Python, C++, Makefiles, assembly, ...)
- Heavily depends on interactions between the two (e.g. the compilation procedure).

Introduction

How can we measure the ressources used during a computation?

- Heavily depends on the hardware (NASA supercomputer, phone, a student, ...)
- Heavily depends on the language (Python, C++, Makefiles, assembly, ...)
- Heavily depends on interactions between the two (e.g. the compilation procedure).

Solution: very low-level models showing atomic steps of computations and standardized units.

Introduction

How can we measure the ressources used during a computation?

- Heavily depends on the hardware (NASA supercomputer, phone, a student, ...)
- Heavily depends on the language (Python, C++, Makefiles, assembly, ...)
- Heavily depends on interactions between the two (e.g. the compilation procedure).

Solution: very low-level models showing atomic steps of computations and standardized units.

Sequential

Parallel

Introduction

How can we measure the ressources used during a computation?

- Heavily depends on the hardware (NASA supercomputer, phone, a student, ...)
- Heavily depends on the language (Python, C++, Makefiles, assembly, ...)
- Heavily depends on interactions between the two (e.g. the compilation procedure).

Solution: very low-level models showing atomic steps of computations and standardized units.

Sequential

- Turing Machines
- Infinite tape
- Each cell contains a single bit
- The head moves one position at a time according to simple control rules

Parallel

Introduction

How can we measure the ressources used during a computation?

- Heavily depends on the hardware (NASA supercomputer, phone, a student, ...)
- Heavily depends on the language (Python, C++, Makefiles, assembly, ...)
- Heavily depends on interactions between the two (e.g. the compilation procedure).

Solution: very low-level models showing atomic steps of computations and standardized units.

Sequential

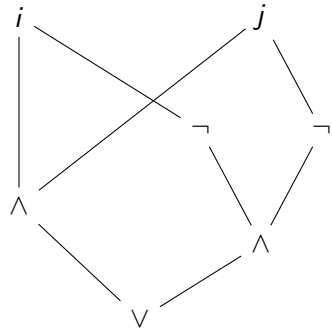
- Turing Machines
- Infinite tape
- Each cell contains a single bit
- The head moves one position at a time according to simple control rules

Parallel

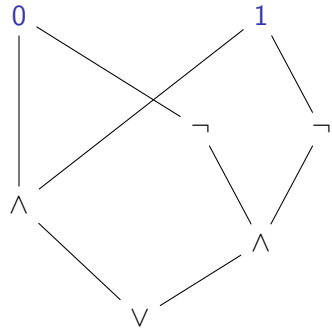
- Boolean Circuits
- Lots of processors
- Each of them computes a single bit, using a Boolean operation
- Either uses inputs bits or result from other processors
- Connections are fixed

Boolean circuits

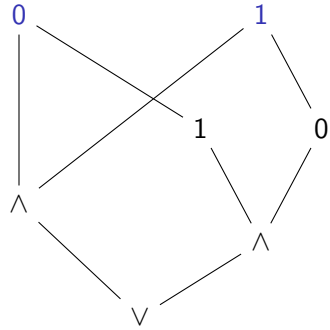
A first circuit



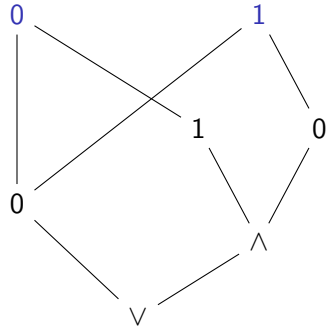
A first circuit



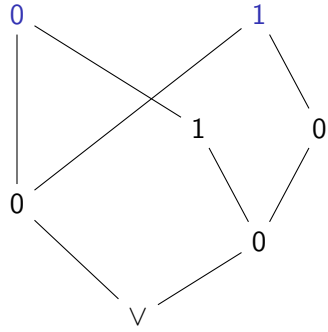
A first circuit



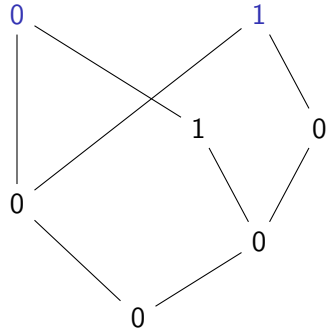
A first circuit



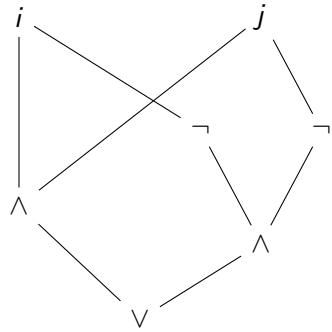
A first circuit



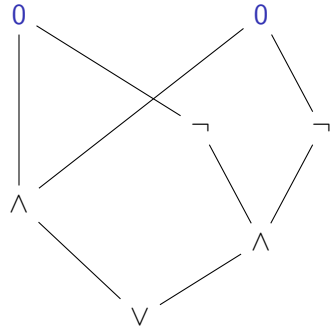
A first circuit



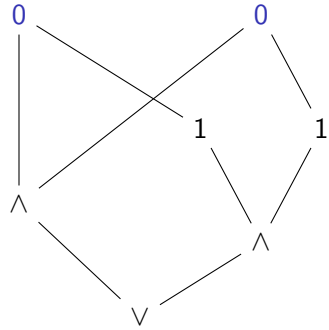
A first circuit



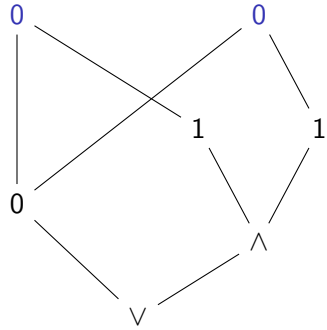
A first circuit



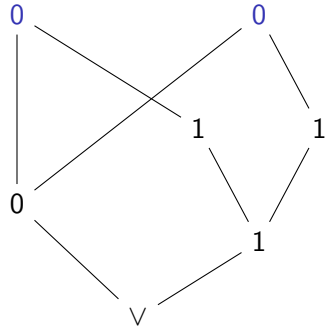
A first circuit



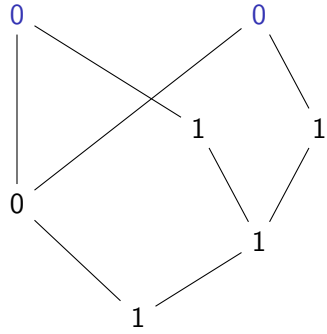
A first circuit



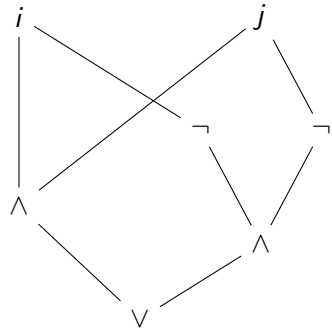
A first circuit



A first circuit

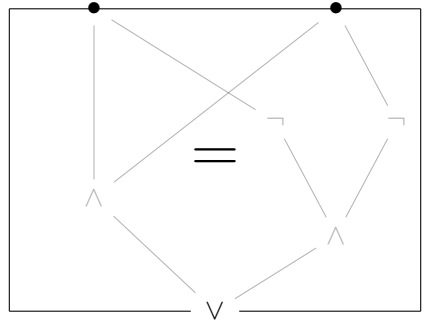


A first circuit



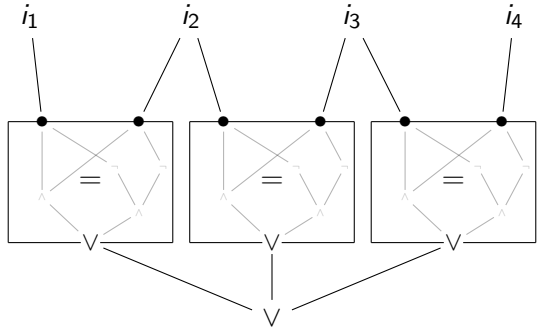
outputs $1 \Leftrightarrow i = j$

A first circuit

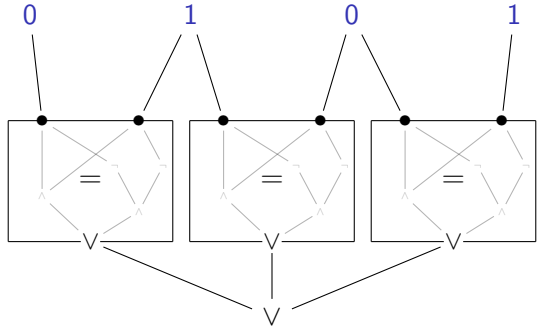


outputs $1 \Leftrightarrow i = j$

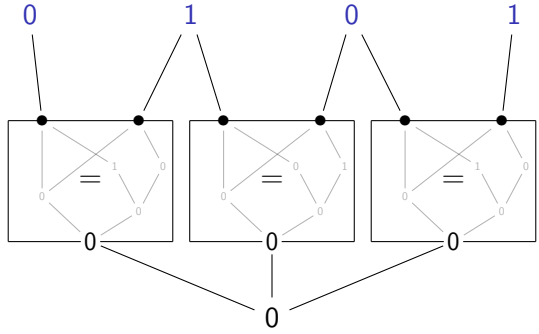
A first circuit



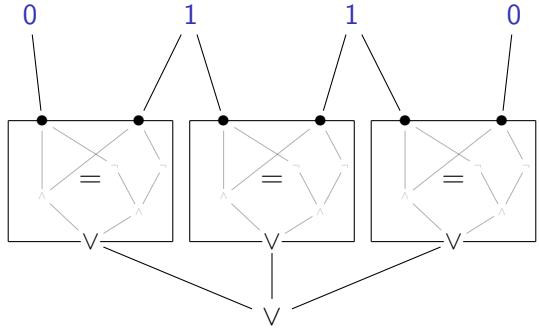
A first circuit



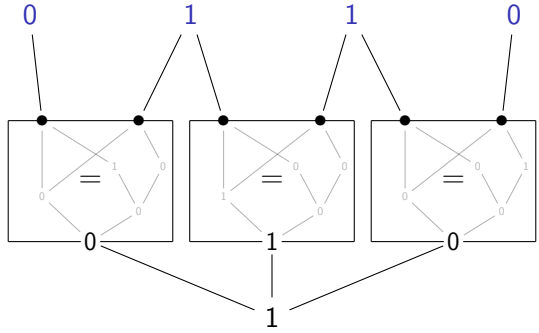
A first circuit



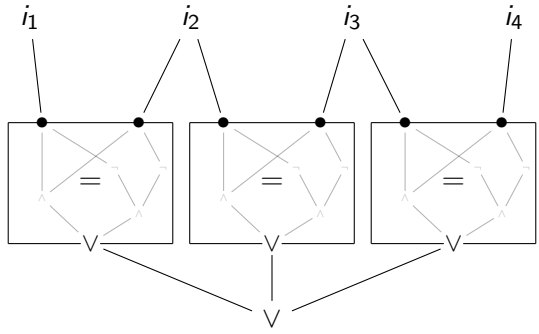
A first circuit



A first circuit

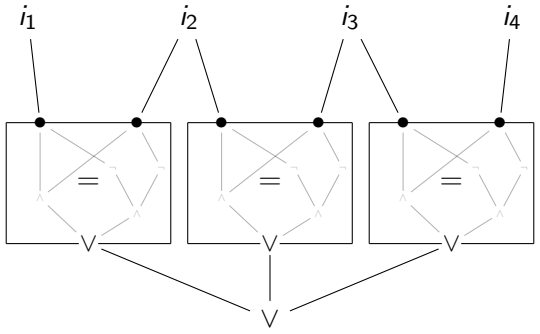


A first circuit



Computes $\Sigma^*(00 + 11)\Sigma^* \cap \Sigma^4$

A first circuit



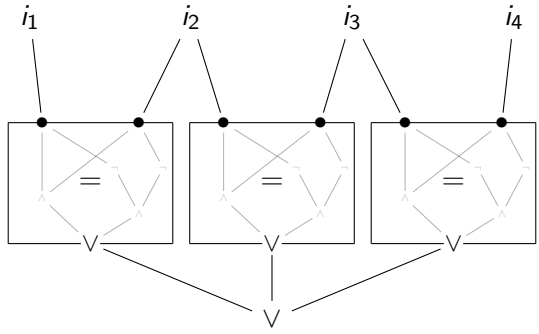
Computes $\Sigma^*(00 + 11)\Sigma^*$

A first circuit

size(n) = number of gates wire(n)= number of wires

depth(n) = maximal length of a path

fan-in(n) = maximal in-degree of a gate



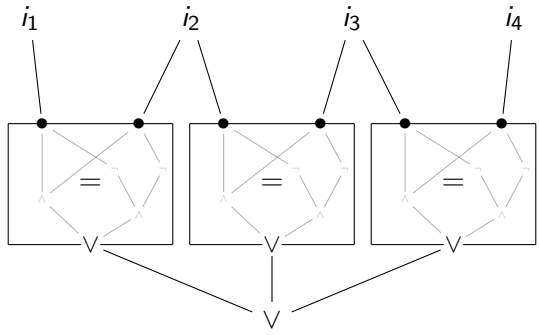
Computes $\Sigma^*(00 + 11)\Sigma^*$

A first circuit

size(n) = number of gates wire(n)= number of wires

depth(n) = maximal length of a path

fan-in(n) = maximal in-degree of a gate



Computes $\Sigma^*(00+11)\Sigma^*$

$\text{size}(n) = 5 \cdot (n-1) + 1$

$\text{wire}(n) = 9 \cdot (n-1)$

$\text{depth}(n) = 4$

$\text{fan-in}(n) = n - 1$

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Nodes are called gates and edges wires.

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Nodes are called gates and edges wires.

Definition (Ressources)

- size $\to$ number of gates
- wires $\to$ number of wires
- depth $\to$ maximal length of a path ($\approx$ execution time)
- fan-in $\to$ maximal indegree of a gate

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Nodes are called gates and edges wires.

Definition (Ressources)

- size $\to$ number of gates
- wires $\to$ number of wires
- depth $\to$ maximal length of a path ($\approx$ execution time)
- fan-in $\to$ maximal indegree of a gate

Gates labelled by 0 or 1 must have fan-in 0.
Gates labelled by $\neg$ must have fan-in 1.

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Nodes are called gates and edges wires.

Definition (Ressources)

- size $\to$ number of gates
- wires $\to$ number of wires
- depth $\to$ maximal length of a path ($\approx$ execution time)
- fan-in $\to$ maximal indegree of a gate

Gates labelled by 0 or 1 must have fan-in 0.
Gates labelled by $\neg$ must have fan-in 1.

Definition (Semantic)

The value $v_\alpha(g)$ of gate g under an assigment $\alpha : X \to \{0, 1\}$ is defined inductively by:

- $\lambda(g) \in \{0, 1\}$: $\lambda(g)$
- $\lambda(g) = x$ for $x \in V$: $\alpha(g)$
- $\lambda(g) = \neg$: 1 iff $v_\alpha(g') = 0$ for the unique g' with $(g', g) \in E$
- $\lambda(g) = \wedge$: 1 iff $v_\alpha(g') = 1$ for all g' with $(g', g) \in E$
- $\lambda(g) = \vee$: 1 iff $v_\alpha(g') = 1$ for some g' with $(g', g) \in E$

Formal definitions

Definition (Syntax)

A circuit C over the variables $X = \{x_1, \cdots, x_n\}$ is a triple (G, λ, g_o) with

- a directed acyclic graph $G = (V, E)$
- a labelling of nodes $\lambda : V \to \{0, 1, \wedge, \vee, \neg\} \cup X$
- a distinguished ouput node g_o

Nodes are called gates and edges wires.

Definition (Ressources)

- size $\to$ number of gates
- wires $\to$ number of wires
- depth $\to$ maximal length of a path ($\approx$ execution time)
- fan-in $\to$ maximal indegree of a gate

Gates labelled by 0 or 1 must have fan-in 0.
Gates labelled by $\neg$ must have fan-in 1.

Definition (Semantic)

The value $v_\alpha(g)$ of gate g under an assigment $\alpha : X \to \{0, 1\}$ is defined inductively by:

- $\lambda(g) \in \{0, 1\}$: $\lambda(g)$
- $\lambda(g) = x$ for $x \in V$: $\alpha(g)$
- $\lambda(g) = \neg$: 1 iff $v_\alpha(g') = 0$ for the unique g' with $(g', g) \in E$
- $\lambda(g) = \wedge$: 1 iff $v_\alpha(g') = 1$ for all g' with $(g', g) \in E$
- $\lambda(g) = \vee$: 1 iff $v_\alpha(g') = 1$ for some g' with $(g', g) \in E$

The ouput of the circuit is $v_\alpha(g_o)$.
The circuit computes a function $f_C : \{0, 1\}^n \to \{0, 1\}$ defined by $f(\alpha) = v_\alpha(g_o)$.

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

By convention, we usually refer to "circuit families" simply by "circuits".

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

By convention, we usually refer to "circuit families" simply by "circuits".

Definition

Each circuit C_n defines a language $L(C_n)$, the set of words w such that $f_{C_n}(w) = 1$.
A circuit family $\mathcal{C}$ defines a language

$$L(\mathcal{C}) = \bigcup_n L(C_n).$$

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

By convention, we usually refer to "circuit families" simply by "circuits".

Definition

Each circuit C_n defines a language $L(C_n)$, the set of words w such that $f_{C_n}(w) = 1$.
A circuit family $\mathcal{C}$ defines a language

$$L(\mathcal{C}) = \bigcup_n L(C_n).$$

No connections required between the C_n.

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

By convention, we usually refer to "circuit families" simply by "circuits".

Definition

Each circuit C_n defines a language $L(C_n)$, the set of words w such that $f_{C_n}(w) = 1$.
A circuit family $\mathcal{C}$ defines a language

$$L(\mathcal{C}) = \bigcup_n L(C_n).$$

No connections required between the C_n.

Example

Denote by Acc_n the circuit with n inputs that always accepts (one "1" gate and no wires). Similarly, define Rej_n that always rejects.
Consider the family $\mathcal{C}$:

- $C_n = \text{Acc}_n$ if the n^{th} TM (in some order) always halts
- $C_n = \text{Rej}_n$ otherwise

Then $L(\mathcal{C})$ is undecidable but all circuits are trivial.

Formal definitions

Each circuit computes only a function of a fixed set of variables
$\rightarrow$ cannot consider complexity

Definition (Families)

A circuit family is a collection $\mathcal{C} = (C_n)_{n \geq 0}$ of circuits, with C_n over a variables set of size n.

By convention, we usually refer to "circuit families" simply by "circuits".

Definition

Each circuit C_n defines a language $L(C_n)$, the set of words w such that $f_{C_n}(w) = 1$.
A circuit family $\mathcal{C}$ defines a language

$$L(\mathcal{C}) = \bigcup_n L(C_n).$$

No connections required between the C_n.

Example

Denote by Acc_n the circuit with n inputs that always accepts (one "1" gate and no wires). Similarly, define Rej_n that always rejects.
Consider the family $\mathcal{C}$:

- $C_n = \text{Acc}_n$ if the n^{th} TM (in some order) always halts
- $C_n = \text{Rej}_n$ otherwise

Then $L(\mathcal{C})$ is undecidable but all circuits are trivial.

Can be avoided with uniformity: a procedure to compute the n^{th} circuit in the family.
$\rightarrow$ Not here
$\rightarrow$ See the lectures on "Complexity Theory"

Links with Turing Machines

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Definition (P/poly)

The complexity class P/poly is the class of languages accepted by a TM running in polynomial time with a polynomial size advice.

$\to$ A non-uniform version of P.

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Definition (P/poly)

The complexity class P/poly is the class of languages accepted by a TM running in polynomial time with a polynomial size advice.

$\to$ A non-uniform version of P.

Theorem

L is in P/poly
$\Leftrightarrow$
L is computable by a poly-size circuit

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Definition (P/poly)

The complexity class P/poly is the class of languages accepted by a TM running in polynomial time with a polynomial size advice.

$\to$ A non-uniform version of P.

Theorem

L is in P/poly
$\Leftrightarrow$
L is computable by a poly-size circuit

Proof ($\Leftarrow$)

- Evaluating a poly-size circuit can be done in poly time.
- The circuit is the advice.

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Definition (P/poly)

The complexity class P/poly is the class of languages accepted by a TM running in polynomial time with a polynomial size advice.

$\to$ A non-uniform version of P.

Theorem

L is in P/poly
$\Leftrightarrow$
L is computable by a poly-size circuit

Proof ($\Leftarrow$)

- Evaluating a poly-size circuit can be done in poly time.
- The circuit is the advice.

For the converse, we admit the following for the moment.

Lemma

There is a poly-size circuit for every language in P.

Machines with advice

Definition (advice)

A Turing machine with advice M is a TM with two inputs x and y, and an advice function $a : \mathbb{N} \to \{0,1\}^*$.
It accepts a word x if M accepts $(x, a(|x|))$.

Definition (P/poly)

The complexity class P/poly is the class of languages accepted by a TM running in polynomial time with a polynomial size advice.

$\to$ A non-uniform version of P.

Theorem

L is in P/poly
$\Leftrightarrow$
L is computable by a poly-size circuit

Proof ($\Leftarrow$)

- Evaluating a poly-size circuit can be done in poly time.
- The circuit is the advice.

For the converse, we admit the following for the moment.

Lemma

There is a poly-size circuit for every language in P.

Proof ($\Rightarrow$)

- Take M a P/poly TM.
- Construct D_n the circuit for M with inputs x and y of sizes n and $|a(n)|$, by the lemma.
- Take C_n to be D_n with the inputs in y replaced by $a(n)$.

Lemma
There is a poly-size circuit for every language in P.

Proof

Lemma
There is a poly-size circuit for every language in P.

Proof
- Let M a 1-tape TM in time $t(n)$.

Lemma
There is a poly-size circuit for every language in P.

Proof
- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.

Lemma
There is a poly-size circuit for every language in P.

Proof

- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.

Lemma
There is a poly-size circuit for every language in P.

Proof

- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.

Lemma
There is a poly-size circuit for every language in P.

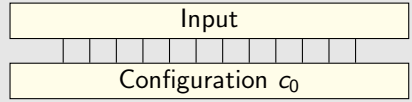
Proof

- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.
- We can compute c_{i+1} from c_i, the input and c_{j_i}.
  $\rightarrow$ Constant size circuit.

Lemma
There is a poly-size circuit for every language in P.

Proof

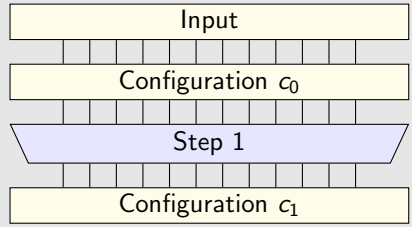
- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.
- We can compute c_{i+1} from c_i, the input and c_{j_i}.
  $\rightarrow$ Constant size circuit.



Lemma
There is a poly-size circuit for every language in P.

Proof

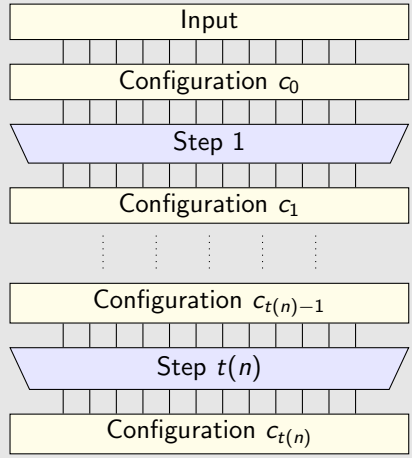
- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.
- We can compute c_{i+1} from c_i, the input and c_{j_i}.
  $\rightarrow$ Constant size circuit.



Lemma
There is a poly-size circuit for every language in P.

Proof

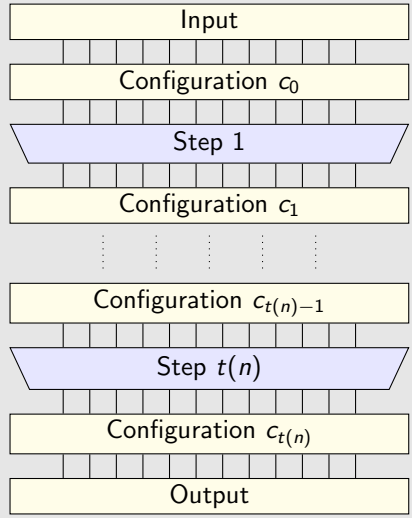
- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.
- We can compute c_{i+1} from c_i, the input and c_{j_i}.
  $\rightarrow$ Constant size circuit.



Lemma
There is a poly-size circuit for every language in P.

Proof

- Let M a 1-tape TM in time $t(n)$.
- Wlog. oblivious: the direction of the head does not depend on the input.
  $\rightarrow$ See simulation of a TM by a 1-tape TM.
- A configuration c_i at time i consists in the state of the TM and the symbol read by all heads.
  $\rightarrow$ c_i constant size string.
- The content of a cell c at time i only depends on the last time j_i the head was in c, which only depends on i.
- We can compute c_{i+1} from c_i, the input and c_{j_i}.
  $\rightarrow$ Constant size circuit.



Solving hard problems?

Claim
Showing that any $L \in \mathsf{NP}$ cannot be computed by a poly-size circuit would prove $\mathsf{P} \neq \mathsf{NP}$.

Solving hard problems?

Claim
Showing that any $L \in \mathrm{NP}$ cannot be computed by a poly-size circuit would prove $\mathrm{P} \neq \mathrm{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Solving hard problems?

Claim

Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)

Almost every Boolean function needs a circuit of exponential size to be computed.

Solving hard problems?

Claim

Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)

Almost every Boolean function needs a circuit of exponential size to be computed.

Proof $\rightarrow$ counting

Solving hard problems?

Claim

Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)

Almost every Boolean function needs a circuit of exponential size to be computed.

Proof $\rightarrow$ counting

Number of Boolean functions with n inputs: 2^{2^n}.

Solving hard problems?

Claim
Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)
Almost every Boolean function needs a circuit of exponential size to be computed.

Proof $\rightarrow$ counting

Number of Boolean functions with n inputs: 2^{2^n}.

Number of circuits with n inputs and size s: $\leq ((n+5)2^s)^s$.
$\rightarrow$ Each gate has a type among $n+5$ choices.
$\rightarrow$ Each gate has 2^s choices for its parents
$\rightarrow$ There are s gates.

Solving hard problems?

Claim
Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)
Almost every Boolean function needs a circuit of exponential size to be computed.

Proof $\rightarrow$ counting

Number of Boolean functions with n inputs: 2^{2^n}.

Number of circuits with n inputs and size s: $\leq ((n+5)2^s)^s$.
$\rightarrow$ Each gate has a type among $n+5$ choices.
$\rightarrow$ Each gate has 2^s choices for its parents
$\rightarrow$ There are s gates.

For $s = \sqrt{2}^n/n$, the ratio 2^{2^n} by $2^{s\log(n+1)+s^2}$ tends to 1.

Solving hard problems?

Claim
Showing that any $L \in \text{NP}$ cannot be computed by a poly-size circuit would prove $\text{P} \neq \text{NP}$.

$\rightarrow$ Circuits seems easier to work with than TMs.

Lemma (Shannon)
Almost every Boolean function needs a circuit of exponential size to be computed.

Proof $\rightarrow$ counting

Number of Boolean functions with n inputs: 2^{2^n}.

Number of circuits with n inputs and size s: $\leq ((n+5)2^s)^s$.
$\rightarrow$ Each gate has a type among $n+5$ choices.
$\rightarrow$ Each gate has 2^s choices for its parents
$\rightarrow$ There are s gates.

For $s = \sqrt{2}^n/n$, the ratio 2^{2^n} by $2^{s\log(n+1)+s^2}$ tends to 1.

We still do not know any explicit such function!

Classes of small circuits

We need to restrict circuits to study them.
Gives a notion of efficiently parallelizable languages.

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

Classes of small circuits

We need to restrict circuits to study them.
Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded.
Otherwise, the output only depends in finitely many inputs.

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded. Otherwise, the output only depends in finitely many inputs.

Definition (NC^1)

The class NC^1 consists of languages recognizable by circuits of polynomial size, logarithmic depth and fan-in 2.

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded. Otherwise, the output only depends in finitely many inputs.

Definition (NC^1)

The class NC^1 consists of languages recognizable by circuits of polynomial size, logarithmic depth and fan-in 2.

What if we restrict the number of communications?

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded. Otherwise, the output only depends in finitely many inputs.

Definition (NC^1)

The class NC^1 consists of languages recognizable by circuits of polynomial size, logarithmic depth and fan-in 2.

What if we restrict the number of communications?

Definition ($WLAC^0$)

The class $WLAC^0$ consists of languages recognizable by circuits of constant depth with linearly many wires.

$\rightarrow$ Extremely efficient parallel algorithms.

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded. Otherwise, the output only depends in finitely many inputs.

Definition (NC^1)

The class NC^1 consists of languages recognizable by circuits of polynomial size, logarithmic depth and fan-in 2.

What if we restrict the number of communications?

Definition ($WLAC^0$)

The class $WLAC^0$ consists of languages recognizable by circuits of constant depth with linearly many wires.

$\rightarrow$ Extremely efficient parallel algorithms.

Claim

$WLAC^0 \subseteq AC^0 \subseteq NC^1$

Classes of small circuits

We need to restrict circuits to study them. Gives a notion of efficiently parallelizable languages.

Definition (AC^0)

The class AC^0 consists of languages recognizable by circuits of polynomial size, constant depth.

$\rightarrow$ In particular, the fan-in is unbounded. Otherwise, the output only depends in finitely many inputs.

Definition (NC^1)

The class NC^1 consists of languages recognizable by circuits of polynomial size, logarithmic depth and fan-in 2.

What if we restrict the number of communications?

Definition ($WLAC^0$)

The class $WLAC^0$ consists of languages recognizable by circuits of constant depth with linearly many wires.

$\rightarrow$ Extremely efficient parallel algorithms.

Claim

$WLAC^0 \subseteq AC^0 \subseteq NC^1$

Proof

- We can remove the gates not linked to any wire.
- a $\vee$ gate of fan-in n can be transformed into a binary tree of size $2n$ and depth $\log(n)$.

Adding numbers

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)
$\rightarrow$ numbers are $x = x_n \cdots x_1$ from high weights to low weigths

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)
$\rightarrow$ numbers are $x = x_n \cdots x_1$ from high weights to low weigths

Example (A circuit for ADD)

- **Notation:** $\oplus$ denotes XOR
- **Idea:** Computes the successive carries:
  - $c_0 = x_0 \wedge y_0$
  - $c_i = (x_i \wedge y_i) \vee ((x_i \vee y_i) \wedge c_{i-1})$

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)
$\rightarrow$ numbers are $x = x_n \cdots x_1$ from high weights to low weigths

Example (A circuit for ADD)

- **Notation:** $\oplus$ denotes XOR
- **Idea:** Computes the successive carries:
  - $c_0 = x_0 \wedge y_0$
  - $c_i = (x_i \wedge y_i) \vee ((x_i \vee y_i) \wedge c_{i-1})$
- The outputs are then:
  - $z_0 = x_0 \oplus y_0$
  - $z_i = x_i \oplus y_i \oplus c_{i-1}$
  - $z_{n+1} = c_n$

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)
$\rightarrow$ numbers are $x = x_n \cdots x_1$ from high weights to low weigths

Example (A circuit for ADD)

- **Notation:** $\oplus$ denotes XOR
- **Idea:** Computes the successive carries:
  - $c_0 = x_0 \wedge y_0$
  - $c_i = (x_i \wedge y_i) \vee ((x_i \vee y_i) \wedge c_{i-1})$
- The outputs are then:
  - $z_0 = x_0 \oplus y_0$
  - $z_i = x_i \oplus y_i \oplus c_{i-1}$
  - $z_{n+1} = c_n$

$\rightarrow$ linear size
$\rightarrow$ linear depth

Adding two numbers

Problem (ADD)

- **Given:** Two n-bits numbers x and y
- **Output:** A $n+1$-bits number $z = x + y$

Not a language, but a function.
$\rightarrow$ Consider circuits with several outputs (one for each bit of the answer)
$\rightarrow$ numbers are $x = x_n \cdots x_1$ from high weights to low weigths

Example (A circuit for ADD)

- **Notation:** $\oplus$ denotes XOR
- **Idea:** Computes the successive carries:
  - $c_0 = x_0 \wedge y_0$
  - $c_i = (x_i \wedge y_i) \vee ((x_i \vee y_i) \wedge c_{i-1})$
- The outputs are then:
  - $z_0 = x_0 \oplus y_0$
  - $z_i = x_i \oplus y_i \oplus c_{i-1}$
  - $z_{n+1} = c_n$

$\rightarrow$ linear size
$\rightarrow$ linear depth

This is roughly the sequential algorithm.
$\rightarrow$ We can do better in parallel!

A better circuit for ADD

Lemma

ADD$\in$ AC^0

A better circuit for ADD

Lemma

ADD$\in$ AC^0

Proof

- **Idea:** Still computes the carries.
  The carry c_i is 1 iff
  - a carry is created at a position $j \leq i$, i.e. $x_j \wedge y_j$, and
  - the carry is not lost between i and j, i.e. $\bigwedge_{l=j+1}^{i}(x_l \vee y_l)$.

A better circuit for ADD

Lemma

ADD$\in$ AC^0

Proof

- **Idea:** Still computes the carries.
  The carry c_i is 1 iff
  - a carry is created at a position $j \leq i$, i.e. $x_j \wedge y_j$, and
  - the carry is not lost between i and j, i.e. $\bigwedge_{l=j+1}^{i}(x_l \vee y_l)$.
- Thus:

$$c_i = \bigvee_{j=0}^{i}((x_j \wedge y_j) \wedge \bigwedge_{l=j+1}^{i}(x_l \vee y_l)$$

A better circuit for ADD

Lemma

ADD$\in$ AC^0

Proof

- **Idea:** Still computes the carries.
  The carry c_i is 1 iff
  - a carry is created at a position $j \leq i$, i.e. $x_j \wedge y_j$, and
  - the carry is not lost between i and j, i.e. $\bigwedge_{l=j+1}^{i}(x_l \vee y_l)$.
- Thus:

$$c_i = \bigvee_{j=0}^{i}((x_j \wedge y_j) \wedge \bigwedge_{l=j+1}^{i}(x_l \vee y_l)$$

- As before: $z_0 = x_0 \oplus y_0$, $z_i = x_i \oplus y_i \oplus c_i$ and $z_{n+1} = c_n$

A better circuit for ADD

Lemma

ADD$\in$ AC^0

Proof

- **Idea:** Still computes the carries.
  The carry c_i is 1 iff
  - a carry is created at a position $j \leq i$, i.e. $x_j \wedge y_j$, and
  - the carry is not lost between i and j, i.e. $\bigwedge_{l=j+1}^{i}(x_l \vee y_l)$.
- Thus:

$$c_i = \bigvee_{j=0}^{i}((x_j \wedge y_j) \wedge \bigwedge_{l=j+1}^{i}(x_l \vee y_l)$$

- As before: $z_0 = x_0 \oplus y_0$, $z_i = x_i \oplus y_i \oplus c_i$ and $z_{n+1} = c_n$

$\rightarrow$ Quadratic size (by factorizing the $(x_j \wedge y_j)$ and $(x_j \vee y_j)$)

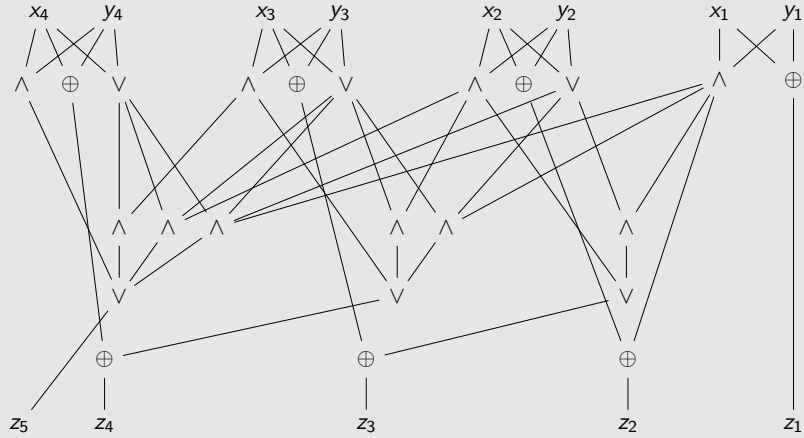
$\rightarrow$ Constant depth

A better circuit for ADD

Lemma

ADD$\in$ AC^0

Proof



Adding many numbers

What if we want to add several numbers in parallel?

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n

- **Given:** n many n-bits numbers $x_1, \ldots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$

- **Given:** $\log(n)$ many n-bits numbers $x_1, \ldots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n
- **Given:** n many n-bits numbers $x_1, \ldots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$
- **Given:** $\log(n)$ many n-bits numbers $x_1, \ldots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n

- **Given:** n many n-bits numbers $x_1, \dots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$

- **Given:** $\log(n)$ many n-bits numbers $x_1, \dots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Claim

$\mathrm{ADD}^n \in \mathrm{NC}^1$

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n

- **Given:** n many n-bits numbers $x_1, \dots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$

- **Given:** $\log(n)$ many n-bits numbers $x_1, \dots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Claim

$\mathrm{ADD}^n \in \mathrm{NC}^1$

Proof

A binary tree of AC^0 circuits for ADD.

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n
- **Given:** n many n-bits numbers $x_1, \dots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$
- **Given:** $\log(n)$ many n-bits numbers $x_1, \dots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Claim
$\mathrm{ADD}^n \in \mathrm{NC}^1$

Proof
A binary tree of AC^0 circuits for ADD.

Lemma
$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

A binary tree only gives depth $\log\log(n)$
$\rightarrow$ next slide

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n
- **Given:** n many n-bits numbers $x_1, \dots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$
- **Given:** $\log(n)$ many n-bits numbers $x_1, \dots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Claim
$\mathrm{ADD}^n \in \mathrm{NC}^1$

Proof
A binary tree of AC^0 circuits for ADD.

Lemma
$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

A binary tree only gives depth $\log\log(n)$
$\rightarrow$ next slide

Theorem
$\mathrm{ADD}^n \notin \mathrm{AC}^0$

$\rightarrow$ next lecture

Adding many numbers

What if we want to add several numbers in parallel?

Problem ADD^n
- **Given:** n many n-bits numbers $x_1, \dots x_n$
- **Output:** A $n + \log(n)$-bits number $z = x_1 + \cdots + x_n$

Problem $\mathrm{ADD}^{\log(n)}$
- **Given:** $\log(n)$ many n-bits numbers $x_1, \dots x_{\log(n)}$
- **Output:** A $n + \log\log(n)$-bits number $z = x_1 + \cdots + x_{\log(n)}$

What are their respective complexities?

Claim
$\mathrm{ADD}^n \in \mathrm{NC}^1$

Proof
A binary tree of AC^0 circuits for ADD.

Lemma
$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

A binary tree only gives depth $\log\log(n)$
$\rightarrow$ next slide

Theorem
$\mathrm{ADD}^n \notin \mathrm{AC}^0$

$\rightarrow$ next lecture

The case of WLAC^0 will be the topic of the rest of this lecture.

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Intuition

- Adding the following numbers...

$$
\begin{aligned}
x_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\
x_2 &= 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\
x_3 &= 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1
\end{aligned}
$$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Intuition

- Adding the following numbers...

$$
\begin{aligned}
x_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\
x_2 &= 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\
x_3 &= 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\
s_4 &= 10
\end{aligned}
$$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Intuition

- Adding the following numbers...

$$\begin{aligned} x_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ x_2 &= 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\ x_3 &= 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\ s_4 &= 10 \end{aligned}$$

- ...reduces to adding the numbers:

$$\begin{aligned} y_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ y_2 &= \quad 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \\ & \qquad\qquad\ \ s_4 \end{aligned}$$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Intuition

- Adding the following numbers...

$$\begin{aligned} x_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ x_2 &= 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\ x_3 &= 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \\ s_4 &= 10 \end{aligned}$$

- ...reduces to adding the numbers:

$$\begin{aligned} y_1 &= 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ y_2 &= \quad 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \\ & \qquad\qquad\ \ s_4 \end{aligned}$$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$,
  $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$,
  $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$,
  $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$,
  $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow$ $\log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow$ $n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$, $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow$ $\log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow$ $n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

- **New goal:** Add $\log\log(n)$ many $(n + \log\log(n))$ bits numbers

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$, $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

- **New goal:** Add $\log\log(n)$ many $(n + \log\log(n))$ bits numbers
- **New new goal:** Add $\log\log\log(n)$ many $(n + \log\log(n) + \log\log\log(n))$ bits numbers

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$, $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

- **New goal:** Add $\log\log(n)$ many $(n + \log\log(n))$ bits numbers
- **New new goal:** Add $\log\log\log(n)$ many $(n + \log\log(n) + \log\log\log(n))$ bits numbers
- ...
- **Last goal:** Add two $n + \log\log(n) + \cdots + \log^k(n)$ bits number where k = smallest integer such that the k-fold application of log on n is ≤ 2.

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$,
  $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

- **New goal:** Add $\log\log(n)$ many $(n + \log\log(n))$ bits numbers
- **New new goal:** Add $\log\log\log(n)$ many $(n + \log\log(n) + \log\log\log(n))$ bits numbers
- ...
- **Last goal:** Add two $n + \log\log(n) + \cdots + \log^k(n)$ bits number where k = smallest integer such that the k-fold application of log on n is ≤ 2.
- Small computation, for n big:
  $\log\log(n) + \cdots + \log^k(n) \leq \log(n)$

Adding $\log(n)$ numbers

Lemma

$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

Proof

- For $1 \leq j \leq n$, $s_j = \sum_{i=1}^{\log(n)} j^{\text{th}}$ bit of x_i
  $\rightarrow \log\log(n)$ bits numbers.
- For $1 \leq l \leq \log\log(n)$, let y_l be the concatenation of the l^{th} bits of the s_j, with $l-1$ zeroes in the end.
  $\rightarrow$ the earlier diagonal form
  $\rightarrow n + \log\log(n)$ bits numbers
- Clearly: $\sum_i x_i = \sum_l y_l$
- All s_j and y_l depends on only $\log(n)$ bits of the input
  $\rightarrow$ they can be computed in AC^0 by "brute force"

- **New goal:** Add $\log\log(n)$ many $(n + \log\log(n))$ bits numbers
- **New new goal:** Add $\log\log\log(n)$ many $(n + \log\log(n) + \log\log\log(n))$ bits numbers
- ...
- **Last goal:** Add two $n + \log\log(n) + \cdots + \log^k(n)$ bits number where $k =$ smallest integer such that the k-fold application of log on n is ≤ 2.
- Small computation, for n big: $\log\log(n) + \cdots + \log^k(n) \leq \log(n)$
- We just have to add two $n + \log(n)$ numbers, that can be computed in AC^0 (depends on $\log(n)$ bits of the y_l)

The power of WLAC^0

Concentration of computation

We will study circuits from the properties of the underlying graph.

Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

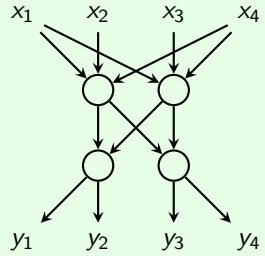
Concentration of computation

We will study circuits from the properties of the underlying graph.

Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



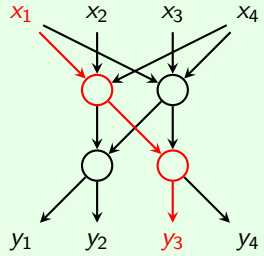
Concentration of computation

We will study circuits from the properties of the underlying graph.

Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



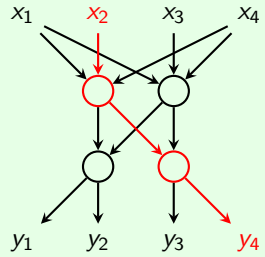
Concentration of computation

We will study circuits from the properties of the underlying graph.

Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



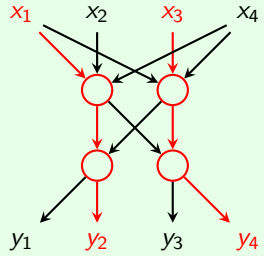
Concentration of computation

We will study circuits from the properties of the underlying graph.

Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



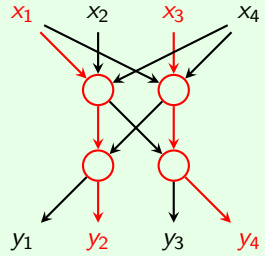
Concentration of computation

We will study circuits from the properties of the underlying graph.

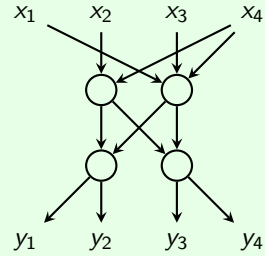
Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



Counter-example



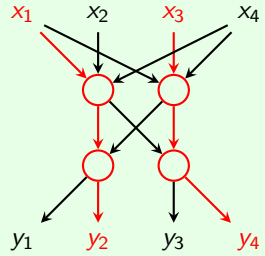
Concentration of computation

We will study circuits from the properties of the underlying graph.

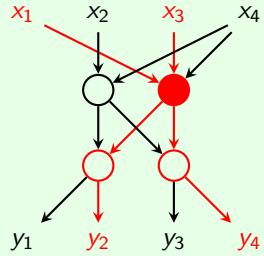
Definition (superconcentrator)

Let G be a DAG with n inputs $x_1, \ldots, x_n$ and n outputs $y_1, \ldots, y_n$. It is a superconcentrator if for all k and set of indices $i_1 < j_1 < \cdots < i_k < j_k$, there are k vertex disjoint paths from $\{x_{i_1}, \ldots, x_{i_k}\}$ to $\{y_{j_1}, \ldots, y_{j_k}\}$.

Example



Counter-example



Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

To have a lower bound against WLAC^0, we only need to prove that certain circuits have to be superconcentrators.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

To have a lower bound against WLAC^0, we only need to prove that certain circuits have to be superconcentrators.

We have tools for that.

Definition (Cut)

Let X and Y be two sets of vertices in a graph. A cut between X and Y is a set of vertices whose removal disconnect X and Y.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

To have a lower bound against WLAC^0, we only need to prove that certain circuits have to be superconcentrators.

We have tools for that.

Definition (Cut)

Let X and Y be two sets of vertices in a graph. A cut between X and Y is a set of vertices whose removal disconnect X and Y.

Theorem (Menger)

For any two disjoints X and Y, the minimum size of a cut between X and Y is also the maximum number of vertex disjoint paths between X and Y.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

To have a lower bound against WLAC^0, we only need to prove that certain circuits have to be superconcentrators.

We have tools for that.

Definition (Cut)

Let X and Y be two sets of vertices in a graph. A cut between X and Y is a set of vertices whose removal disconnect X and Y.

Theorem (Menger)

For any two disjoints X and Y, the minimum size of a cut between X and Y is also the maximum number of vertex disjoint paths between X and Y.

$\rightarrow$ Can be seen as a special case of the max-flow min-cut theorem.

Efficient superconcentrators

Intuition: to have many vertex disjoint paths, there must be many edges. But:

Lemma (Valiant, Pippinger)

There is a superconcentrator with linearly many edges.

$\rightarrow$ it has logarithmic depth.

To tackle WLAC^0, we have more structure.

Theorem (Dolev, Dwork, Pippinger, Widgerson)

There are no superconcentrator with constant-depth and linearly many edges.

To have a lower bound against WLAC^0, we only need to prove that certain circuits have to be superconcentrators.

We have tools for that.

Definition (Cut)

Let X and Y be two sets of vertices in a graph. A cut between X and Y is a set of vertices whose removal disconnect X and Y.

Theorem (Menger)

For any two disjoints X and Y, the minimum size of a cut between X and Y is also the maximum number of vertex disjoint paths between X and Y.

$\rightarrow$ Can be seen as a special case of the max-flow min-cut theorem.

$\rightarrow$ To show that there are many vertex disjoint paths, we must show that there are no small cuts.

Back to ADD

Theorem

$\mathrm{ADD} \notin \mathrm{WLAC}^0$

Proof

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.
- C' is C with:
  - x_i set to 1 and y_i set to 0 for $i \notin I$
  - x_i and y_i merged for $i \in I$

  $\rightarrow$ for $i \in I$, x_i and y_i must be set to the same value.
  $\rightarrow$ k vertex disjoint paths in C' gives the same in G.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.
- C' is C with:
  - x_i set to 1 and y_i set to 0 for $i \notin I$
  - x_i and y_i merged for $i \in I$

  $\rightarrow$ for $i \in I$, x_i and y_i must be set to the same value.
  $\rightarrow$ k vertex disjoint paths in C' gives the same in G.

- **Claim:** the output z_{j_l} is 0 if x_{i_l} and y_{i_l} are both 1.
  the output z_{j_l} is 1 if x_{i_l} and y_{i_l} are both 0.
  $\rightarrow$ if $x_{i_l} = y_{i_l}$, what is on the right does not matter.
  $\rightarrow$ a carry is created and propagated if and only if both are 1.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.
- C' is C with:
  - x_i set to 1 and y_i set to 0 for $i \notin I$
  - x_i and y_i merged for $i \in I$

  $\rightarrow$ for $i \in I$, x_i and y_i must be set to the same value.
  $\rightarrow$ k vertex disjoint paths in C' gives the same in G.

- **Claim:** the output z_{j_l} is 0 if x_{i_l} and y_{i_l} are both 1.
  the output z_{j_l} is 1 if x_{i_l} and y_{i_l} are both 0.
  $\rightarrow$ if $x_{i_l} = y_{i_l}$, what is on the right does not matter.
  $\rightarrow$ a carry is created and propagated if and only if both are 1.
- Thus there are 2^k possible outcomes for then outputs in J, depending on the inputs in I.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.
- C' is C with:
  - x_i set to 1 and y_i set to 0 for $i \notin I$
  - x_i and y_i merged for $i \in I$

  $\rightarrow$ for $i \in I$, x_i and y_i must be set to the same value.
  $\rightarrow$ k vertex disjoint paths in C' gives the same in G.

- **Claim:** the output z_{j_l} is 0 if x_{i_l} and y_{i_l} are both 1.
  the output z_{j_l} is 1 if x_{i_l} and y_{i_l} are both 0.
  $\rightarrow$ if $x_{i_l} = y_{i_l}$, what is on the right does not matter.
  $\rightarrow$ a carry is created and propagated if and only if both are 1.
- Thus there are 2^k possible outcomes for then outputs in J, depending on the inputs in I.
- If there were a cut of size $< k$, there would be $< 2^k$ possible outcomes.

Back to ADD

Theorem

ADD $\notin$ WLAC^0

Proof

- Let C be a circuit for ADD with n inputs.
- G is the graph with inputs x_i and y_i merged.
  $\rightarrow$ we want to show that it is a superconcentrator.
- Let $i_1 < j_1 < \cdots < i_k < j_k$.
  $\rightarrow$ sets I and J.
- C' is C with:
  - x_i set to 1 and y_i set to 0 for $i \notin I$
  - x_i and y_i merged for $i \in I$

  $\rightarrow$ for $i \in I$, x_i and y_i must be set to the same value.
  $\rightarrow$ k vertex disjoint paths in C' gives the same in G.

- **Claim:** the output z_{j_l} is 0 if x_{i_l} and y_{i_l} are both 1.
  the output z_{j_l} is 1 if x_{i_l} and y_{i_l} are both 0.
  $\rightarrow$ if $x_{i_l} = y_{i_l}$, what is on the right does not matter.
  $\rightarrow$ a carry is created and propagated if and only if both are 1.
- Thus there are 2^k possible outcomes for then outputs in J, depending on the inputs in I.
- If there were a cut of size $< k$, there would be $< 2^k$ possible outcomes.
- We conclude by Menger's theorem.

Recap

We have seen today:

	WLAC^0	AC^0	NC^1
ADD	✗	✓	✓
$\mathrm{ADD}^{\log(n)}$	✗	✓	✓
ADD^n	✗	✗	✓

Recap

We have seen today:

	WLAC^0	AC^0	NC^1
ADD	✗	✓	✓
$\mathrm{ADD}^{\log(n)}$	✗	✓	✓
ADD^n	✗	✗	✓

Whether ADD can be done in constant depth with linearly many nodes is a major open problem.

Limitations of constant-depth circuits

Corentin Barloy Michael Walter Thomas Zeume

RUHR UNIVERSITÄT BOCHUM RUB

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma

$ADD^{\log(n)} \in AC^0$

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$ADD^{\log(n)} \in AC^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$ADD^n \notin AC^0$

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$ADD^{\log(n)} \in AC^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$ADD^n \notin AC^0$

$\rightarrow$ Not so easy!

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$ADD^{\log(n)} \in AC^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$ADD^n \notin AC^0$

$\rightarrow$ Not so easy!

Simplification: Go back to languages.
$\rightarrow$ Only look at the last bit of the output.

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$\mathrm{ADD}^{\log(n)} \in \mathrm{AC}^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$\mathrm{ADD}^n \notin \mathrm{AC}^0$

$\rightarrow$ Not so easy!

Simplification: Go back to languages.
$\rightarrow$ Only look at the last bit of the output.

$$\begin{array}{rccccc} & x_1^1 & x_2^1 & x_3^1 & \cdots & x_n^1 \\ + & x_1^2 & x_2^2 & x_3^2 & \cdots & x_n^2 \\ & & & \vdots & & \\ + & x_1^n & x_2^n & x_3^n & \cdots & x_n^n \\ \hline & y_1 & y_2 & y_3 & \cdots & y_n \end{array}$$

$\rightarrow$ y_n only depends on the number of 1 among $x_n^1, \ldots, x_n^n$.

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$ADD^{\log(n)} \in AC^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$ADD^n \notin AC^0$

$\rightarrow$ Not so easy!

Simplification: Go back to languages.
$\rightarrow$ Only look at the last bit of the output.

$$\begin{array}{rccccc} & x_1^1 & x_2^1 & x_3^1 & \cdots & x_n^1 \\ + & x_1^2 & x_2^2 & x_3^2 & \cdots & x_n^2 \\ & & & \vdots & & \\ + & x_1^n & x_2^n & x_3^n & \cdots & x_n^n \\ \hline & y_1 & y_2 & y_3 & \cdots & y_n \end{array}$$

$\rightarrow y_n$ only depends on the number of 1 among $x_n^1, \ldots, x_n^n$.

Problem (Parity)

- **Given:** n bits $x_1, \ldots, x_n$
- **Output:** The parity of the number of 1 in the inputs: $\sum_i x_i \bmod 2$.

Introduction

Recall: AC^0 is the class of languages computable but constant depth and polynomial size circuits.

We showed that it is rather powerful:

Lemma
$ADD^{\log(n)} \in AC^0$

We now want to show lower bounds, i.e. inexpressibility results.
The goal today:

Theorem
$ADD^n \notin AC^0$

$\rightarrow$ Not so easy!

Simplification: Go back to languages.
$\rightarrow$ Only look at the last bit of the output.

$$\begin{array}{rccccc} & x_1^1 & x_2^1 & x_3^1 & \cdots & x_n^1 \\ + & x_1^2 & x_2^2 & x_3^2 & \cdots & x_n^2 \\ & & & \vdots & & \\ + & x_1^n & x_2^n & x_3^n & \cdots & x_n^n \\ \hline & y_1 & y_2 & y_3 & \cdots & y_n \end{array}$$

$\rightarrow$ y_n only depends on the number of 1 among $x_n^1, \ldots, x_n^n$.

Problem (Parity)

- **Given:** n bits $x_1, \ldots, x_n$
- **Output:** The parity of the number of 1 in the inputs: $\sum_i x_i \bmod 2$.

Theorem (New goal)

$$\text{Parity} \notin AC^0$$

The Parity language

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.

Complexity of regular languages

Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.

w_1 w_2 w_3 w_4

Complexity of regular languages

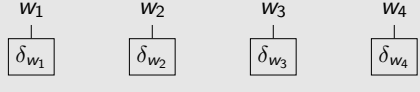
Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.



Complexity of regular languages

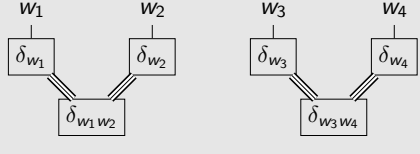
Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.



Complexity of regular languages

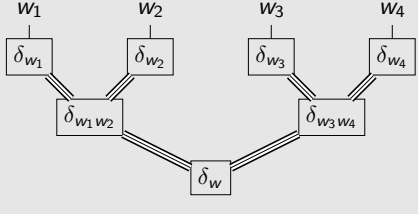
Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.



Complexity of regular languages

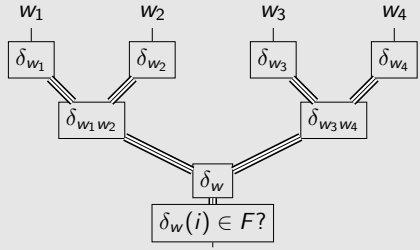
Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.



Complexity of regular languages

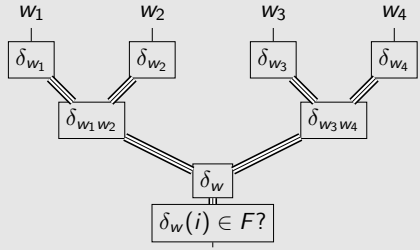
Parity $= (0^*10^*10^*)^*$ is regular. This already gives an upper bound.

Theorem

All regular languages are in NC^1.

Proof

- Let $\mathcal{A} = (Q, \delta, i, F)$ be an automaton.
- For w a word, δ_w is the extended transition function.
- Finitely many functions $Q \to Q$ $\to$ can be represented by strings of constant length.
- We have constant size circuits for:
  - computing δ_x from a bit x,
  - function composition $\delta_1 \circ \delta_2$,
  - whether a function maps i to a state in F.



A binary tree of constant size circuits: logarithmic depth and linear size

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

$\rightarrow$ depth-3 AC^0 circuits.
$\rightarrow$ Useful is practice.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

$\rightarrow$ depth-3 AC^0 circuits.
$\rightarrow$ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

$\rightarrow$ depth-3 AC^0 circuits.
$\rightarrow$ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

$\rightarrow$ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

- Every T_i has n literals:
  → all variables appear.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

- Every T_i has n literals:
  → all variables appear.
- T_i accepts only one word.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

- Every T_i has n literals:
  → all variables appear.
- T_i accepts only one word.
- Thus D accepts at most N words.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

- Every T_i has n literals:
  → all variables appear.
- T_i accepts only one word.
- Thus D accepts at most N words.
- There are 2^{n-1} words in Parity.
  → $N \geq 2^{n-1}$.

A first easy lower bound

Definition (DNF)

A literal is a variable or its negation. A term is a conjunction of literals. A DNF is a disjunction of clauses.

→ depth-3 AC^0 circuits.
→ Useful is practice.

Claim

Any DNF for Parity must have at least 2^{n-1} terms.

Optimal with $\bigvee_{w \in \text{Parity}} T(w)$

→ $T(w)$ is the term with $\begin{cases} x_i & \text{if } w_i = 1 \\ \neg x_i & \text{if } w_i = 0 \end{cases}$

Proof

- $D = \bigvee_{i=1}^{N} T_i$ a DNF for Parity.
- We remove all terms with a contradiction $x_j \wedge \neg x_j$.
- Assume one T_i has $< n$ literals.
  → variable x_j does not appear in T_i.
- Take w accepted by T_i:
  → w with x_j flipped accepted by T_i.
  → it has different parity.

- Every T_i has n literals:
  → all variables appear.
- T_i accepts only one word.
- Thus D accepts at most N words.
- There are 2^{n-1} words in Parity.
  → $N \geq 2^{n-1}$.

Both bounds works for CNF.

An acceleration

If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

An acceleration

If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.

An acceleration

If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.

$$x_1 \quad x_2 \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad x_{n-1} \quad x_n$$

An acceleration

If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.

An acceleration

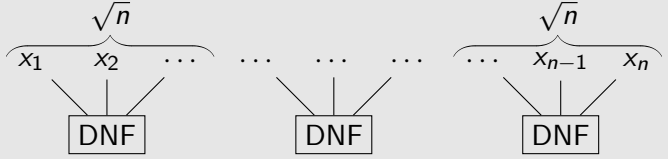
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



An acceleration

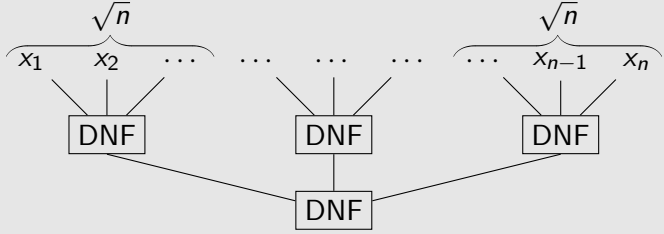
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



An acceleration

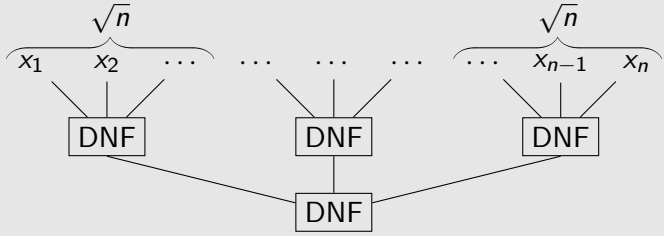
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



- So far: depth 6.

An acceleration

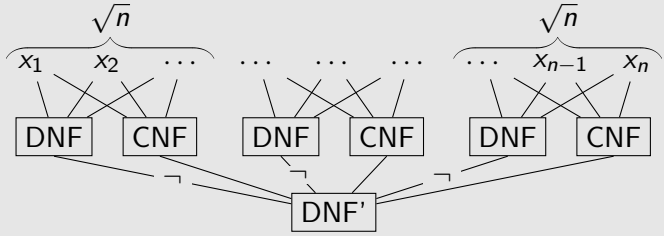
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



- So far: depth 6.
- DNF' has no negations.

An acceleration

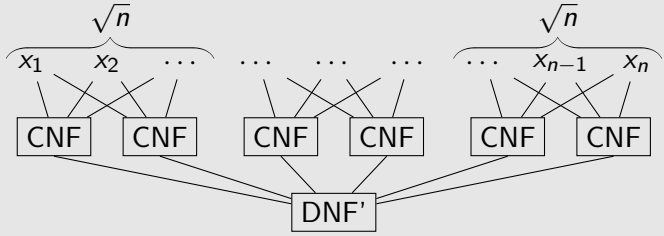
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



- So far: depth 6.
- DNF' has no negations.
- Use De Morgan's laws.

An acceleration

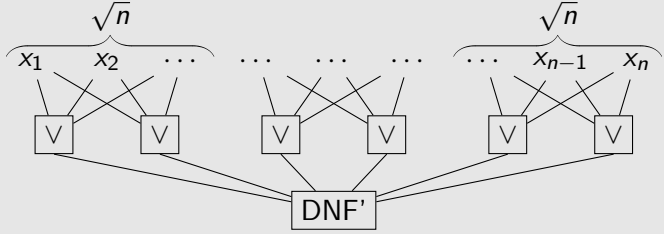
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



- So far: depth 6.
- DNF' has no negations.
- Use De Morgan's laws.

- Collapse the two layers of $\wedge$.
  $\rightarrow$ depth 4

An acceleration

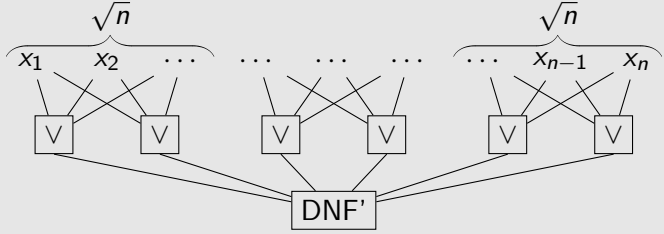
If more depth is available, this can be improved.

Lemma (Håstad)

There is a depth-4 AC^0 circuit with $O(\sqrt{n}2^{\sqrt{n}})$ gates for Parity.

Proof

Idea: Compute Parity of small blocks, then compute Parity of the parities.



- So far: depth 6.
- DNF' has no negations.
- Use De Morgan's laws.

- Collapse the two layers of $\wedge$.
  $\rightarrow$ depth 4
- $2\sqrt{n}+1$ circuits of size $2^{\sqrt{n}-1}$.
  $\rightarrow$ size $O(\sqrt{n}2^{\sqrt{n}})$

An acceleration

If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

An acceleration

If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.

An acceleration

If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.

An acceleration

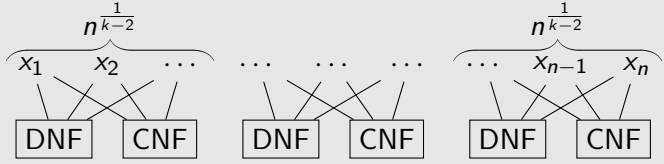
If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.



An acceleration

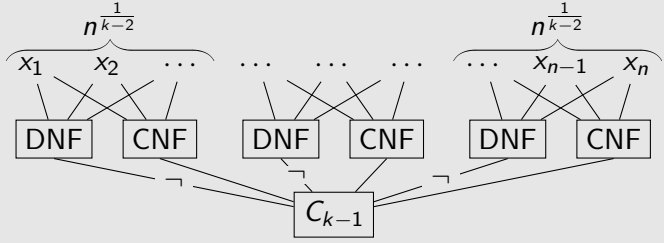
If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.



- C_{k-1} of depth k with a last $\wedge$ layer
  $\rightarrow$ has $n^{\frac{k-3}{k-2}}$ inputs
  $\rightarrow$ size $O(n \cdot 2^{n^{\frac{1}{k-2}}})$

An acceleration

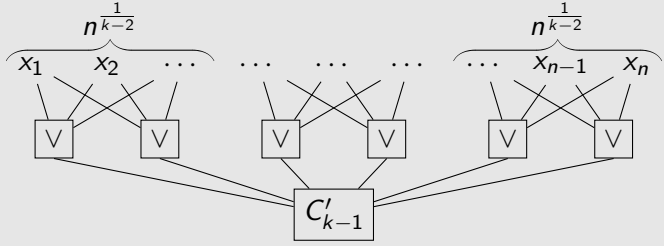
If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.



- C_{k-1} of depth k with a last $\wedge$ layer
  $\rightarrow$ has $n^{\frac{k-3}{k-2}}$ inputs
  $\rightarrow$ size $O(n \cdot 2^{n^{\frac{1}{k-2}}})$
- Collapse of two layers: depth k

An acceleration

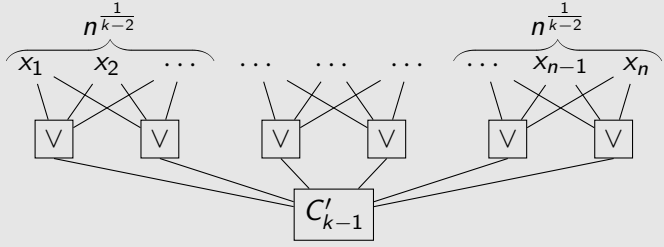
If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.



- C_{k-1} of depth k with a last $\wedge$ layer
  $\rightarrow$ has $n^{\frac{k-3}{k-2}}$ inputs
  $\rightarrow$ size $O(n \cdot 2^{n^{\frac{1}{k-2}}})$
- Collapse of two layers: depth k
- $2 \cdot n^{\frac{k-3}{k-2}}$ DNF of size $2^{n^{\frac{1}{k-2}}}$

An acceleration

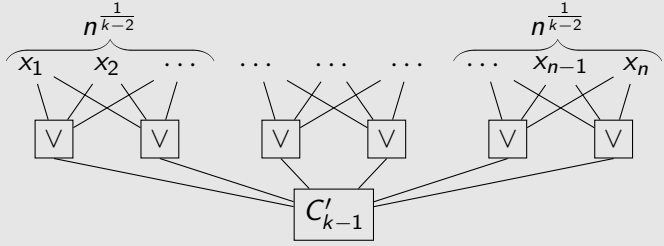
If more depth is available, this can be improved even further.

Lemma (Håstad)

There is a depth-k AC^0 circuit with $O(n \cdot 2^{n^{\frac{1}{k-2}}})$ gates for Parity.

Proof

By induction.



- C_{k-1} of depth k with a last $\wedge$ layer
  $\rightarrow$ has $n^{\frac{k-3}{k-2}}$ inputs
  $\rightarrow$ size $O(n \cdot 2^{n^{\frac{1}{k-2}}})$
- Collapse of two layers: depth k
- $2 \cdot n^{\frac{k-3}{k-2}}$ DNF of size $2^{n^{\frac{1}{k-2}}}$
- Total size $O(n \cdot 2^{n^{\frac{1}{k-2}}})$.

Proof 1: Switching lemma

A normal form

It is useful to assume that AC^0 circuit have a special shape.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

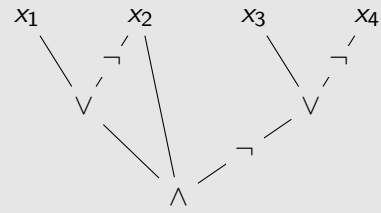
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

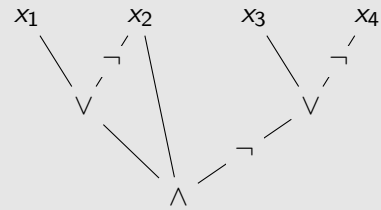
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

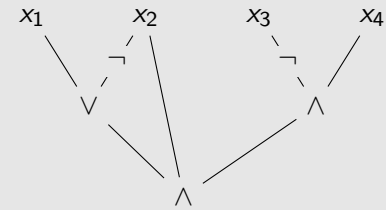
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

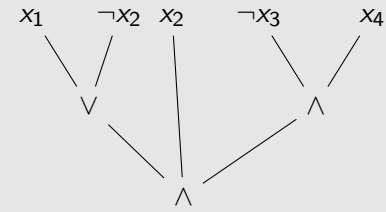
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

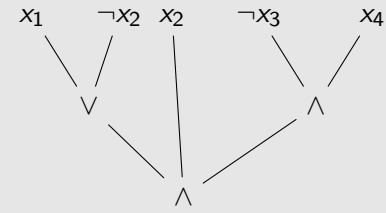
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).
- Fill with dummy gates between gates of the same type.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

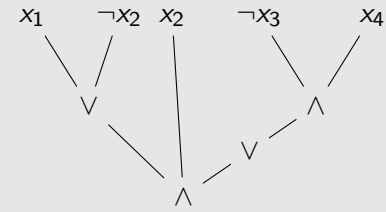
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).
- Fill with dummy gates between gates of the same type.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

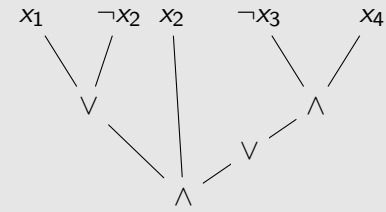
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).
- Fill with dummy gates between gates of the same type.
- Layer with dummy gates.

A normal form

It is useful to assume that AC^0 circuit have a special shape.

Definition (Alternating circuits)

An alternating circuit of depth d is a circuit with

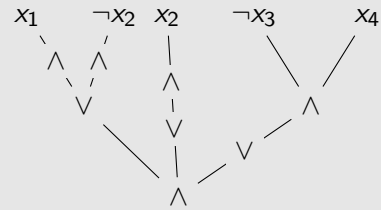
- variables or their negations as input
- d alternating layers of $\vee-$ and $\wedge-$ gates.

$\rightarrow$ negations are not counted in the depth.
$\rightarrow$ depth-2 alternating circuits are just DNFs and CNFs.
$\rightarrow$ the efficient circuits for Parity are already alternating.

Claim

Every AC^0 circuit of depth d can be transformed into an alternating circuit of depth d.

Proof



- Push negations to the leaves (potentially duplicating gates).
- Fill with dummy gates between gates of the same type.
- Layer with dummy gates.

Depth reduction

Idea for lower bound for alternating circuits:

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

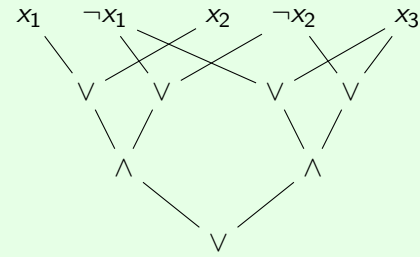
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

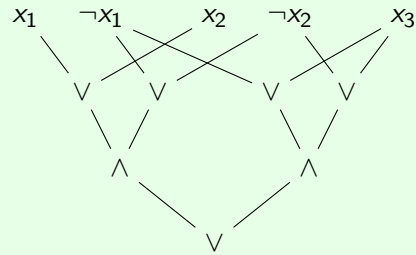
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

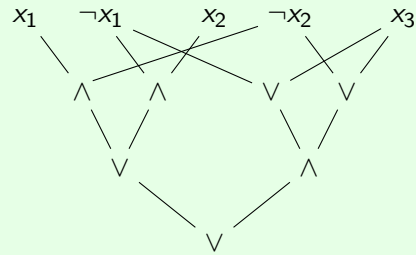
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

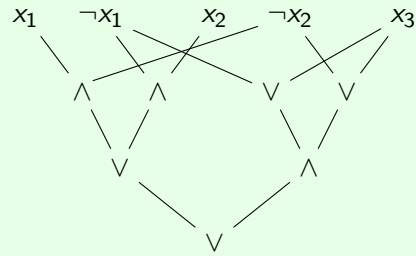
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
- $(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$
  $\equiv (\neg x_1 \wedge \neg x_2) \vee (x_3)$

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

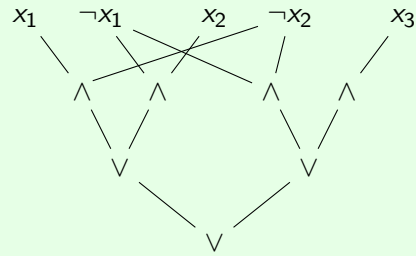
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
- $(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$
  $\equiv (\neg x_1 \wedge \neg x_2) \vee (x_3)$

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

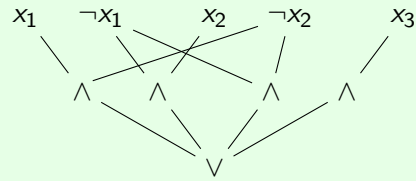
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
- $(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$
  $\equiv (\neg x_1 \wedge \neg x_2) \vee (x_3)$

Depth reduction

Idea for lower bound for alternating circuits:

- We have shown that alternating of depth 2 cannot compute Parity.
- We want to show that if Parity is computed by a depth k circuits, then it can be computed by a depth $k-1$ circuit.

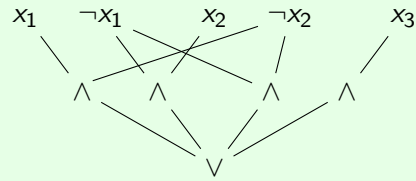
Claim (Switching)

Let C be an alternating circuit of size ≥ 3. Replacing all CNFs at the leaves by equivalent DNFs reduces the depth by one.

Proof

- By alternation, all parents of the CNFs are $\vee$-gates.
- After replacement, we can merge these $\vee$ with the DNF ones.

Example



- $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  $\equiv (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
- $(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$
  $\equiv (\neg x_1 \wedge \neg x_2) \vee (x_3)$

Definition (t-CNF)

A t-CNF is a CNF with t clauses.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.

Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.
$\to$ for f a Boolean function over X, a restriction ρ defines a subfunction f_ρ.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.
$\to$ for f a Boolean function over X, a restriction ρ defines a subfunction f_ρ.

Definition

A l-restriction is a restriction that assigns $*$ to exactly l variables.

$\to f_\rho$ has l variables for such ρ.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.
$\to$ for f a Boolean function over X, a restriction ρ defines a subfunction f_ρ.

Definition

A l-restriction is a restriction that assigns $*$ to exactly l variables.

$\to f_\rho$ has l variables for such ρ.
$\to$ we will draw such restrictions uniformly.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.
$\to$ for f a Boolean function over X, a restriction ρ defines a subfunction f_ρ.

Definition

A l-restriction is a restriction that assigns $*$ to exactly l variables.

$\to f_\rho$ has l variables for such ρ.
$\to$ we will draw such restrictions uniformly.

Definition (t-CNF and s-DNF)

A t-CNF is a CNF with t clauses.
A s-DNF is a DNF with s clauses.

The switching lemma

Problem: Replacing a CNF by a DNF can lead to an exponential blow-up.

Example

$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n})$ has only DNFs of size $\geq 2^n$.

Solution: CNFs can be replaced by small DNFs with high probability after fixing some inputs randomly.
Intuition: $\vee$ and $\wedge$ gates can easily be fixed.

Definition (Restriction)

A restriction for a set of variables X is a mapping

$$\rho : X \to \{0, 1, *\}.$$

$\to *$ means that the variable is unassigned.
$\to$ for f a Boolean function over X, a restriction ρ defines a subfunction f_ρ.

Definition

A l-restriction is a restriction that assigns $*$ to exactly l variables.

$\to f_\rho$ has l variables for such ρ.
$\to$ we will draw such restrictions uniformly.

Definition (t-CNF and s-DNF)

A t-CNF is a CNF with t clauses.
A s-DNF is a DNF with s clauses.

Theorem (Switching lemma)

For $0 \leq p \leq 1$ and f a Boolean function f with n variables that can be expressed as a t-CNF:

$$\mathbb{P}_\rho(f_\rho \text{ has no } s\text{-DNF}) \leq (8pt)^s$$

where the probability is taken over all pn-restrictions.

$\to$ Admitted.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.
- Assume a depth-$(k+1)$ circuit of size S for Parity.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.
- Assume a depth-$(k+1)$ circuit of size S for Parity.
- Wlog. the first layer has $\vee$-gates.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.
- Assume a depth-$(k+1)$ circuit of size S for Parity.
- Wlog. the first layer has $\vee$-gates.
- We want to apply the switching lemma:
  $\rightarrow$ but the fan-in of the first layer can be big.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.
- Assume a depth-$(k+1)$ circuit of size S for Parity.
- Wlog. the first layer has $\vee$-gates.
- We want to apply the switching lemma:
  $\rightarrow$ but the fan-in of the first layer can be big.
- We proceed in two steps:
  - **Step 1:** fan-in reduction of the first layer.
  - **Step 2:** depth reduction.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof

- By induction: we have proved the case $k = 2$ already.
- Assume a depth-$(k+1)$ circuit of size S for Parity.
- Wlog. the first layer has $\vee$-gates.
- We want to apply the switching lemma:
  $\rightarrow$ but the fan-in of the first layer can be big.
- We proceed in two steps:
  - **Step 1:** fan-in reduction of the first layer.
  - **Step 2:** depth reduction.
- Key idea: Subfunctions of Parityare Parityitself or its negation.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs
- We can apply the switching lemma with
  - $t = 1$
  - $s = m$
  - $p = \frac{1}{16}$

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs
- We can apply the switching lemma with
  - $t = 1$
  - $s = m$
  - $p = \frac{1}{16}$
- The probability that a chosen $\vee$-gate cannot be turned into a s-CNF is at most: $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs
- We can apply the switching lemma with
  - $t = 1$
  - $s = m$
  - $p = \frac{1}{16}$

- The probability that a chosen $\vee$-gate cannot be turned into a s-CNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the first layer cannot be turned into a s-CNF.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs
- We can apply the switching lemma with
  - $t = 1$
  - $s = m$
  - $p = \frac{1}{16}$

- The probability that a chosen $\vee$-gate cannot be turned into a s-CNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the first layer cannot be turned into a s-CNF.
  $\rightarrow$ There is a pn-restriction for which all $\vee$-gates can be turned into s-CNFs.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 1: fan-in reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- The gates of the first layer can be seen as 1-DNFs
- We can apply the switching lemma with
  - $t = 1$
  - $s = m$
  - $p = \frac{1}{16}$
- The probability that a chosen $\vee$-gate cannot be turned into a s-CNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the first layer cannot be turned into a s-CNF.
  $\rightarrow$ There is a pn-restriction for which all $\vee$-gates can be turned into s-CNFs.
- After collasping the $\wedge$-gates, we have a circuit for Parity with:
  - depth $k+1$
  - size at most S^2
  - fan-in of the first layer at most m
  - has $\frac{n}{16}$ variables.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$
- The probability that a chosen $\wedge$-gate cannot be turned into a s-DNF is at most: $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$

- The probability that a chosen $\wedge$-gate cannot be turned into a s-DNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the second layer cannot be turned into a s-DNF.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$

- The probability that a chosen $\wedge$-gate cannot be turned into a s-DNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the second layer cannot be turned into a s-DNF.
  $\rightarrow$ There is a pn-restriction for which all $\wedge$-gates can be turned into s-DNFs.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$
- The probability that a chosen $\wedge$-gate cannot be turned into a s-DNF is at most:
  $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the second layer cannot be turned into a s-DNF.
  $\rightarrow$ There is a pn-restriction for which all $\wedge$-gates can be turned into s-DNFs.
- After collasping the $\wedge$-gates, we have a circuit for Parity with:
  - depth k
  - size at most S^2
  - has $\frac{n}{16^2 m}$ variables.

Theorem (Furst, Saxe and Sipser, and Håstad)

All depth-k circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{k-1}})}$.

Proof (Step 2: depth reduction)

- The circuit has depth $k+1$, size S and the first layer has $\vee$-gates.
- Set $m = 2 \log S$.
- We can apply the switching lemma to the CNFs of the second layer with
  - $t = m$
  - $s = m$
  - $p = \frac{1}{16m}$
- The probability that a chosen $\wedge$-gate cannot be turned into a s-DNF is at most: $\leq (8pt)^s = \frac{1}{2^s} = \frac{1}{S^2}$
  $\rightarrow$ Union bound: with probability < 1 at least one the gate of the second layer cannot be turned into a s-DNF.
  $\rightarrow$ There is a pn-restriction for which all $\wedge$-gates can be turned into s-DNFs.
- After collasping the $\wedge$-gates, we have a circuit for Parity with:
  - depth k
  - size at most S^2
  - has $\frac{n}{16^2 m}$ variables.
- By induction: $S^2 \leq 2^{\left(\frac{n}{16^2 m}\right)^{1/(k-1)}}$, which gives $\log(S) \leq c_k n^{\frac{1}{k}}$ for a constant c_k.

Proof 2: Polynomial approximation

Polynomials

Polynomials are simpler objects than circuits.
$\rightarrow$ algebraic instead of combinatoric.

Polynomials

Polynomials are simpler objects than circuits.
$\rightarrow$ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

Polynomials

Polynomials are simpler objects than circuits.
$\rightarrow$ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1^{st} **try:** Boolean polynomials.

$+ = \vee$ $\times = \wedge$

Polynomials

Polynomials are simpler objects than circuits.
$\rightarrow$ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1^{st} **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

$\rightarrow$ circuits without sharing

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd **try:** polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd **try:** polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd **try:** polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$$+ = \vee \qquad \times = \wedge$$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st **try:** Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd **try:** polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd **try:** polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Definition

A Boolean function $f(\overline{x})$ is represented by a polynomial $p(\overline{x})$ over $\mathbb{F}_3$ if $f(\overline{a}) = p(\overline{a})$ for all $\overline{a}$ with only 0 and 1.

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$0 + 1 = 1 + 0 = 1$
$0 + 0 = 1 + 1 = 0$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Definition

A Boolean function $f(\overline{x})$ is represented by a polynomial $p(\overline{x})$ over $\mathbb{F}_3$ if $f(\overline{a}) = p(\overline{a})$ for all $\overline{a}$ with only 0 and 1.

Example

$$\text{Parity} = \left(\prod_{i=1}^{n}(x_i + 1)\right) - 1$$

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$+ = \vee \qquad \times = \wedge$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Definition

A Boolean function $f(\overline{x})$ is represented by a polynomial $p(\overline{x})$ over $\mathbb{F}_3$ if $f(\overline{a}) = p(\overline{a})$ for all $\overline{a}$ with only 0 and 1.

Example

$$\text{Parity} = (\prod_{i=1}^{n}(x_i + 1)) - 1$$

→ We look at the degree.
→ It looks that Parity needs high degree.

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$$+ = \vee \qquad \times = \wedge$$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$$0+1=1+0=1$$
$$0+0=1+1=0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0,1,2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Definition

A Boolean function $f(\overline{x})$ is represented by a polynomial $p(\overline{x})$ over $\mathbb{F}_3$ if $f(\overline{a}) = p(\overline{a})$ for all $\overline{a}$ with only 0 and 1.

Example

$$\text{Parity} = (\textstyle\prod_{i=1}^{n}(x_i+1)) - 1$$

→ We look at the degree.
→ It looks that Parity needs high degree.

Example

$$\neg x = 1 - x$$
$$\textstyle\bigvee_{i=1}^{n} x_i = 1 - \prod_{i=1}^{n}(2x_i+1)$$

Polynomials

Polynomials are simpler objects than circuits.
→ algebraic instead of combinatoric.

Proof Idea

1) Capture functions computed by AC^0 circuits by simple polynomials.
2) Show that Parity cannot be captured by such simple polynomials.

1st try: Boolean polynomials.

$$+ = \vee \qquad \times = \wedge$$

→ circuits without sharing
→ Not helping
→ We want a field.

2nd try: polynomials over $\mathbb{F}_2$.

$$0 + 1 = 1 + 0 = 1$$
$$0 + 0 = 1 + 1 = 0$$

→ Parity is $\sum_{i=1}^{n} x_i$
→ One of the simplest polynomial.
→ Not a chance to have 2).

3rd try: polynomials over $\mathbb{F}_3 = \{0, 1, 2\}$.
$+$ and $\times$ are the usual mod 3
→ What do we do with 2?

Definition

A Boolean function $f(\overline{x})$ is represented by a polynomial $p(\overline{x})$ over $\mathbb{F}_3$ if $f(\overline{a}) = p(\overline{a})$ for all $\overline{a}$ with only 0 and 1.

Example

$$\text{Parity} = (\textstyle\prod_{i=1}^{n}(x_i + 1)) - 1$$

→ We look at the degree.
→ It looks that Parity needs high degree.

Example

$$\neg x = 1 - x$$
$$\textstyle\bigvee_{i=1}^{n} x_i = 1 - \prod_{i=1}^{n}(2x_i + 1)$$

→ Seems that $\vee$ also needs high degree...

Last try: the approximation technique

Idea: Relax the notion of representation.
$\rightarrow$ we will approximate circuits with polynomials over $\mathbb{F}_3$.

Proof Idea

1) Approximate functions computed by AC^0 circuits by low-degree polynomials over $\mathbb{F}_3$.
2) Parity cannot be approximated by low-degree polynomials over $\mathbb{F}_3$.

Approximate means being correct on many inputs.

Definition

For $f(\overline{x})$ a function with n inputs and p a polynomial:
$\text{distance}(f,p) = |\{\overline{a} \in \{0,1\}^n \mid p(\overline{a}) \neq f(\overline{a})\}|$

Formalization of 1):

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f,p) \leq M \cdot 2^{n-r}.$$

$\rightarrow$ proved later.

Formalization of 2):

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity},p) \geq c \cdot 2^n.$$

$\rightarrow$ Admitted.

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity}, p) \geq c \cdot 2^n.$$

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity}, p) \geq c \cdot 2^n.$$

Proof

- Let C be a circuit for Parity of depth d and size M.

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity}, p) \geq c \cdot 2^n.$$

Proof

- Let C be a circuit for Parity of depth d and size M.
- With $r = n^{\frac{1}{2d}}/2$: there is p of degree $\leq \sqrt{n}$ such that:
  $\text{distance}(\text{Parity}, p) \leq M \cdot 2^{n - n^{1/2d}/2}$.

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity}, p) \geq c \cdot 2^n.$$

Proof

- Let C be a circuit for Parity of depth d and size M.
- With $r = n^{\frac{1}{2d}}/2$: there is p of degree $\leq \sqrt{n}$ such that:
  $\text{distance}(\text{Parity}, p) \leq M \cdot 2^{n-n^{1/2d}/2}$.
- $\text{distance}(\text{Parity}, p) \geq c \cdot 2^n$.

Parity $\notin AC^0$

Theorem (Razborov, Smolenski)

All depth-d circuits for Parity have size at least $2^{\Omega(n^{\frac{1}{2d}})}$.

Lemma

Let C be a circuit of depth d and size M that computes a function f.
Then, for $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

There is a constant $c > 0$ such that every polynomial of degree $\leq \sqrt{n}$ satisfies:

$$\text{distance}(\text{Parity}, p) \geq c \cdot 2^n.$$

Proof

- Let C be a circuit for Parity of depth d and size M.
- With $r = n^{\frac{1}{2d}}/2$: there is p of degree $\leq \sqrt{n}$ such that: $\text{distance}(\text{Parity}, p) \leq M \cdot 2^{n-n^{1/2d}/2}$.
- $\text{distance}(\text{Parity}, p) \geq c \cdot 2^n$.
- Thus, $c \cdot 2^n \leq M \cdot 2^{n-n^{1/2d}/2}$.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.

$\rightarrow$ Wlog. there are no $\wedge$-gates.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.

$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Degree bound:

- $\neg$-gates do not increase the degree.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Degree bound:

- $\neg$-gates do not increase the degree.
- $\vee$-gates multiply the degree by $2r$: $\deg(p_g) \leq 2r \cdot \max(\deg(p_{h_k}))$.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Degree bound:

- $\neg$-gates do not increase the degree.
- $\vee$-gates multiply the degree by $2r$: $\deg(p_g) \leq 2r \cdot \max(\deg(p_{h_k}))$.
- For a gate at depth i, $\deg(g) \leq (2r)^i$.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Degree bound:

- $\neg$-gates do not increase the degree.
- $\vee$-gates multiply the degree by $2r$: $\deg(p_g) \leq 2r \cdot \max(\deg(p_{h_k}))$.
- For a gate at depth i, $\deg(g) \leq (2r)^i$.

Distance bound:

- $\neg$-gates do not introduce errors.

Lemma

For $1 \leq r \leq n$, there is a polynomial p of degree $\leq (2r)^d$ such that:

$$\text{distance}(f, p) \leq M \cdot 2^{n-r}.$$

Proof by induction, we approximate gates in the circuit and combine them top-down.
$\rightarrow$ Wlog. there are no $\wedge$-gates.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that $\text{distance}(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ Proof next slide

Proof

We approximate inductively every gate g by p_g:

- An input gate x_i by x_i.
- A $\neg$-gate $\neg h$ by $1 - p_h$.
- A $\vee$-gate $\bigvee_{k=1}^{m} h_k$ by $p_\vee(p_{h_1}, \ldots, p_{h_m})$.

Degree bound:

- $\neg$-gates do not increase the degree.
- $\vee$-gates multiply the degree by $2r$: $\deg(p_g) \leq 2r \cdot \max(\deg(p_{h_k}))$.
- For a gate at depth i, $\deg(g) \leq (2r)^i$.

Distance bound:

- $\neg$-gates do not introduce errors.
- for a $\vee$-gate, assume p_{h_k} is wrong for at most $M_k \cdot 2^{n-r}$ inputs, where M_k is the size of the subcircuit of h_k.
  Then p_g is wrong for at most $(M_1 + \cdots + M_k + 1) \cdot 2^{n-r}$ inputs.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then $p(\overline{a}) = 0$, hence we have 2).

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then $p(\overline{a}) = (d_{i_1} + \cdots + d_{i_m})^2$ for some indices.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
$p(\overline{a}) = (d_{i_1} + \cdots + d_{i_m})^2$ for some indices.
$\rightarrow$ the components are independent thus the sum has the same probability of being 0,1 or 2.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
$p(\overline{a}) = (d_{i_1} + \cdots + d_{i_m})^2$ for some indices.
$\rightarrow$ the components are independent thus the sum has the same probability of being 0,1 or 2.
$\rightarrow$ Thus we have 2).

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

- For $\overline{c} \in \mathbb{F}_3^n$ define:
$p(\overline{x}) = (c_1 x_1 + \cdots + d_n x_n)^2$
$\rightarrow$ Always in $\{0,1\}$: $0^2 = 0$ and $1^2 = 2^2 = 1$.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
$p(\overline{a}) = (d_{i_1} + \cdots + d_{i_m})^2$ for some indices.
$\rightarrow$ the components are independent thus the sum has the same probability of being 0,1 or 2.
$\rightarrow$ Thus we have 2).
- This proves the claim for $r = 1$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
$\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
$\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
$\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
$\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
$\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
$\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.
- Fix some $\overline{a}$:

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
   $$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
   $X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
   Its expectancy is
   $\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
  $\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
  $\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then all $p_i(\overline{a}) = 0$ and thus $p(\overline{a}) = 0$, hence we have 2).

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
   $$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
   $X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
   Its expectancy is
   $\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
  $\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
  $\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then all $p_i(\overline{a}) = 0$ and thus $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
  $\mathbb{P}[p(\overline{a}) \neq 1] = \prod_r \mathbb{P}[p_r(\overline{a}) \neq 1]$ by independence.

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
$$\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}.$$
3) Define the random variable for the distance between p and $\vee$:
$X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
Its expectancy is
$\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
$\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
$\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then all $p_i(\overline{a}) = 0$ and thus $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
$\mathbb{P}[p(\overline{a}) \neq 1] = \prod_r \mathbb{P}[p_r(\overline{a}) \neq 1]$ by independence.
$\rightarrow$ Thus we have 2).

Claim

For $1 \leq r \leq n$, there is a polynomial $p_\vee$ of degree $\leq 2r$ such that distance$(\vee, p_\vee) \leq 2^{n-r}$

$\rightarrow$ The return of the probabilistic method.

Proof

1) Define a polynomial p for all $\overline{c} \in \mathbb{F}_3^N$ uniformly drawn.
2) Show that for all $\overline{a}$:
   $\mathbb{P}_{\overline{c}}[p(\overline{a}) \neq \vee(\overline{a})] \leq 3^{-r}$.
3) Define the random variable for the distance between p and $\vee$:
   $X = \sum_{\overline{a} \in \{0,1\}^n} \mathbb{1}_{p(\overline{a}) \neq \vee(\overline{a})}$.
   Its expectancy is
   $\leq \sum_{\overline{a} \in \{0,1\}^n} 3^{-r} \leq 2^{n-r}$.
4) There is a $\overline{c}$ such that distance$(\vee, p) \leq 2^{n-r}$.

Use probability amplification.

- Set $N = rn$, and define r polynomials $p_1, \ldots, p_r$ as before.
  $\rightarrow$ depends on $\overline{c} \in \mathbb{F}_3^N$.
- Define $p(\overline{x}) = 1 - \prod_{i=1}^{r}(1 - q_i(\overline{x}))$.
  $\rightarrow$ Evaluates to 0 iff all q_i evaluates to 0.
- Fix some $\overline{a}$:
- if $\vee(\overline{a}) = 0$ then all $p_i(\overline{a}) = 0$ and thus $p(\overline{a}) = 0$, hence we have 2).
- if $\vee(\overline{a}) = 1$, then
  $\mathbb{P}[p(\overline{a}) \neq 1] = \prod_r \mathbb{P}[p_r(\overline{a}) \neq 1]$ by independence.
  $\rightarrow$ Thus we have 2).
- This proves the claim for any r.

Recap

We have seen:

- Reduction from ADD^n to Parity.

Recap

We have seen:

- Reduction from ADD^n to Parity.
- Every regular language is in NC^1.

Recap

We have seen:

- Reduction from ADD^n to Parity.
- Every regular language is in NC^1.
- Parity can be rather efficiently computed but we have one of the highlights of complexity:

Theorem

Parity $\notin \mathrm{AC}^0$

Recap

We have seen:

- Reduction from ADD^n to Parity.
- Every regular language is in NC^1.
- Parity can be rather efficiently computed but we have one of the highlights of complexity:

> **Theorem**
> Parity $\notin \mathrm{AC}^0$

- Two different proofs:
  - Reducing the depth iteratively with random restrictions: switching lemma.
  - Approximate AC^0 circuits by low-degree polynomials.

Boolean circuits and regular languages

Corentin Barloy Michael Walter Thomas Zeume

RUHR UNIVERSITÄT BOCHUM RUB

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

Introduction

Why studying regular languages from the point-of-view of circuit complexity?
$\rightarrow$ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

$\rightarrow$ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

$\rightarrow$ optimizing them is important

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

  → optimizing them is important

  → complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes.

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Completness

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Definition (Separator)

A separator for a class $\mathcal{C}_2$ from a class $\mathcal{C}_1$ is a language L that belongs in $\mathcal{C}_2/\mathcal{C}_1$.

Completness

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Definition (Separator)

A separator for a class $\mathcal{C}_2$ from a class $\mathcal{C}_1$ is a language L that belongs in $\mathcal{C}_2/\mathcal{C}_1$.

→ We want to find separators to compare the expressive power of classes.

Completness

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Definition (Separator)

A separator for a class $\mathcal{C}_2$ from a class $\mathcal{C}_1$ is a language L that belongs in $\mathcal{C}_2/\mathcal{C}_1$.

→ We want to find separators to compare the expressive power of classes.

Completness

Definition (Reduction)

A projection from L_1 to L_2 is a circuit that computes L_1 with a single gate labelled by L_2. It is polynomial if the fan-in of the gate is polynomial.

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Definition (Separator)

A separator for a class $\mathcal{C}_2$ from a class $\mathcal{C}_1$ is a language L that belongs in $\mathcal{C}_2/\mathcal{C}_1$.

→ We want to find separators to compare the expressive power of classes.

Completness

Definition (Reduction)

A projection from L_1 to L_2 is a circuit that computes L_1 with a single gate labelled by L_2. It is polynomial if the fan-in of the gate is polynomial.

Definition (Completeness)

A language L is complete under projections for a class $\mathcal{C}$ if $L \in \mathcal{C}$ and there is a projection from every language in $\mathcal{C}$ to L.

Introduction

Why studying regular languages from the point-of-view of circuit complexity?

→ regular languages are everywhere (linguistics, text processing/editing, bioinformatics, ...)

→ optimizing them is important

→ complexity under P: sequential vs parallel

→ Many classes's behaviours are reflected on the regular languages it computes. (under NC^1.)

Separation

Definition (Separator)

A separator for a class $\mathcal{C}_2$ from a class $\mathcal{C}_1$ is a language L that belongs in $\mathcal{C}_2/\mathcal{C}_1$.

→ We want to find separators to compare the expressive power of classes.

Completness

Definition (Reduction)

A projection from L_1 to L_2 is a circuit that computes L_1 with a single gate labelled by L_2. It is polynomial if the fan-in of the gate is polynomial.

Definition (Completeness)

A language L is complete under projections for a class $\mathcal{C}$ if $L \in \mathcal{C}$ and there is a projection from every language in $\mathcal{C}$ to L.

Many separators and complete languages can be chosen regular.

Importance of regular languages

Separation

Parity is a regular separator for NC^1 from AC^0.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

$\rightarrow$ Similar proof as last lecture.
$\rightarrow$ Gives regular separators for $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{NC}^1$ for a prime p.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

$\rightarrow$ Similar proof as last lecture.
$\rightarrow$ Gives regular separators for $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{NC}^1$ for a prime p.

Now is $\mathsf{ACC}^0 = \mathsf{NC}^1$?

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

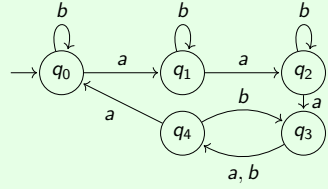
Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

$\rightarrow$ Similar proof as last lecture.
$\rightarrow$ Gives regular separators for $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{NC}^1$ for a prime p.

Now is $\mathsf{ACC}^0 = \mathsf{NC}^1$? We do not know but:

Example (S_5)



Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

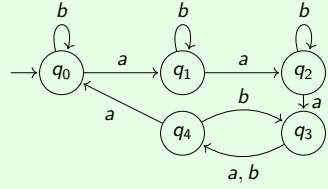
Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

$\rightarrow$ Similar proof as last lecture.
$\rightarrow$ Gives regular separators for $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{NC}^1$ for a prime p.

Now is $\mathsf{ACC}^0 = \mathsf{NC}^1$? We do not know but:

Example (S_5)



Theorem (Barrington)

S_5 is complete under projections for NC^1.

Separation

Parity is a regular separator for NC^1 from AC^0.
$\rightarrow$ Can we express all of NC^1 using Parity gates for free?

Definition (Modular languages)

For $m \in \mathbb{N}$, the language of binary strings with an number of 1 divisible by m is denoted by Mod_m.

Definition (AC^0 with counting)

For $m \in \mathbb{N}$, $\mathsf{ACC}^0[m]$ is the class of languages computable by an AC^0 circuits with some gates labelled by Mod_m.
If it can use any Mod_m gates, it gives the class ACC^0.

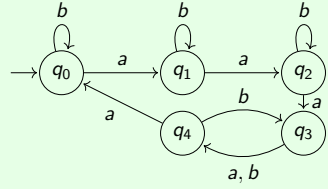
Theorem (Razborov, Smolenski)

For primes $p \neq q$, Mod_q is not in $\mathsf{ACC}^0[p]$.

$\rightarrow$ Similar proof as last lecture.
$\rightarrow$ Gives regular separators for $\mathsf{AC}^0 \subsetneq \mathsf{ACC}^0[p] \subsetneq \mathsf{NC}^1$ for a prime p.

Now is $\mathsf{ACC}^0 = \mathsf{NC}^1$? We do not know but:

Example (S_5)



Theorem (Barrington)

S_5 is complete under projections for NC^1.

$\rightarrow$ If there is a separator for NC^1 from ACC^0, then there is one regular.

Below AC^0

Let us look at the depth hierarchy of AC^0.

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma_d^* / A_d$

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma_d^* / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow$ $AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma^*_{d+1}\#_d A_d \#_d \Sigma^*_{d+1}$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow$ $O_d = \Sigma^*_d / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow$ $AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow$ $O_d = \Sigma_d^* / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Do they also separates the hierarchies?

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma_d^* / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Do they also separates the hierarchies?

Theorem (Sipser, Håstad)
O_d is not in $\Sigma_{d-1} \cup \Pi_d$.
A_d is not in $\Sigma_d \cup \Pi_{d-1}$.

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma_d^* / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Do they also separates the hierarchies?

Theorem (Sipser, Håstad)
O_d is not in $\Sigma_{d-1} \cup \Pi_d$.
A_d is not in $\Sigma_d \cup \Pi_{d-1}$.

$\rightarrow$ proof by switching lemma

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma_{d+1}^* \#_d A_d \#_d \Sigma_{d+1}^*$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma_d^* / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Do they also separates the hierarchies?

Theorem (Sipser, Håstad)
O_d is not in $\Sigma_{d-1} \cup \Pi_d$.
A_d is not in $\Sigma_d \cup \Pi_{d-1}$.

$\rightarrow$ proof by switching lemma
$\rightarrow$ Unlike Parity, random restrictions easily make O_d and A_d trivial.

Below AC^0

Let us look at the depth hierarchy of AC^0.

Definition
For $d \in \mathbb{N}$, the class of languages computed by alternating circuits of polynomial size and depth d with an output $\vee$-gate (resp. $\wedge$-gate) is denoted Σ_d (resp. Π_d).

$\rightarrow AC^0 = \bigcup_i \Sigma_i = \bigcup_i \Pi_i$
Are some of these classes equal?

Definition
We define regular languages over an alphabet $\Sigma_d = \{0, 1, \#_1, \ldots, \#_{d-1}\}$.

- $O_1 = 0^*10^*$
- $A_1 = 1^*$
- $O_{d+1} = \Sigma^*_{d+1}\#_d A_d \#_d \Sigma^*_{d+1}$
- $A_{d+1} = (\#_d O_d)^* \#_d$

$\rightarrow O_d = \Sigma^*_d / A_d$

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

$\rightarrow$ proof next slide.
$\rightarrow$ Another regular languages that are complete for a natural class.

Do they also separates the hierarchies?

Theorem (Sipser, Håstad)
O_d is not in $\Sigma_{d-1} \cup \Pi_d$.
A_d is not in $\Sigma_d \cup \Pi_{d-1}$.

$\rightarrow$ proof by switching lemma
$\rightarrow$ Unlike Parity, random restrictions easily make O_d and A_d trivial.
$\rightarrow$ We need to draw random restrictions with a carefully chosen distribution.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.
- Let C_d be a circuit in Π_d for A_d.

Proof of completeness

Theorem

O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.

- Let C_d be a circuit in Π_d for A_d.
- Then O_{d+1} can be computed by:

$$\bigvee_{i<j} \#_d(i) \wedge \#_d(j) \wedge C_d[i,j]$$

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.

- Let C_d be a circuit in Π_d for A_d.
- Then O_{d+1} can be computed by:

$$\bigvee_{i<j} \#_d(i) \wedge \#_d(j) \wedge C_d[i,j]$$

- The extra $\wedge$ are absorbed by C_d

Proof of completeness

Theorem

O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.

- Let C_d be a circuit in Π_d for A_d.
- Then O_{d+1} can be computed by:

$$\bigvee_{i<j} \#_d(i) \wedge \#_d(j) \wedge C_d[i,j]$$

- The extra $\wedge$ are absorbed by C_d
- There is a quadratic number of poly size circuits.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are in Σ_d)

- By induction:
- O_1 is the $\vee$ function and A_1 is the $\wedge$ function.

- Let C_d be a circuit in Π_d for A_d.
- Then O_{d+1} can be computed by:

$$\bigvee_{i<j} \#_d(i) \wedge \#_d(j) \wedge C_d[i,j]$$

- The extra $\wedge$ are absorbed by C_d
- There is a quadratic number of poly size circuits.

- A_d is the complement of O_d: the negation of that circuit gives a Π_d circuit for A_d.

Proof of completeness

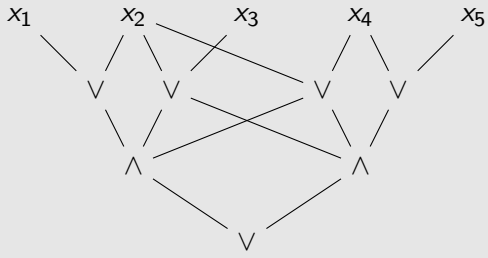
Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

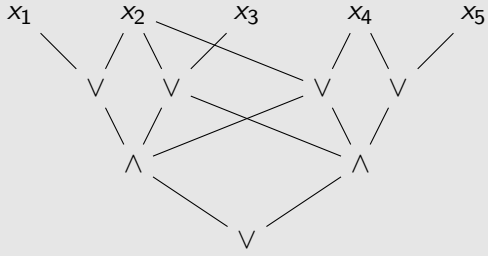
Proof (O_d and A_d are complete)



Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

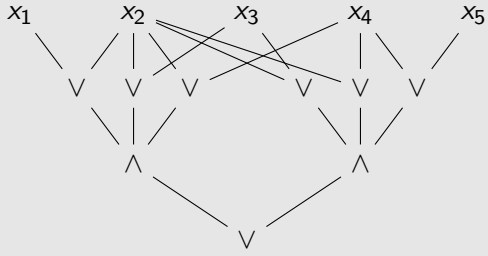


- Remove sharing. $\rightarrow$ remains of poly size

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

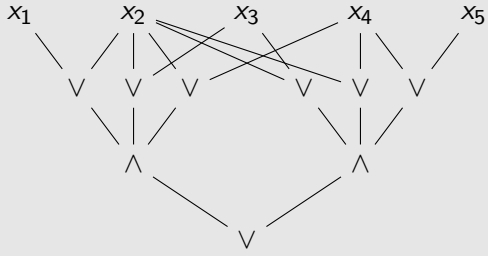


- Remove sharing. $\rightarrow$ remains of poly size

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

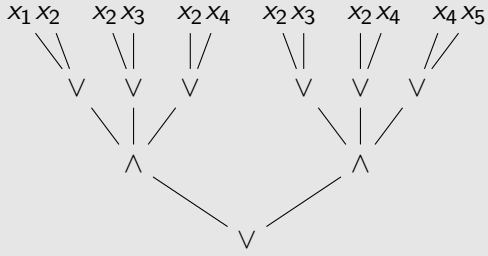


- Remove sharing. $\rightarrow$ remains of poly size
- Duplicate and order variables.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

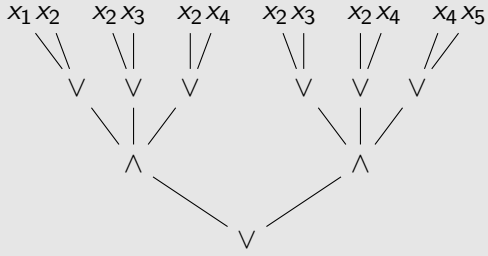


- Remove sharing. $\rightarrow$ remains of poly size
- Duplicate and order variables.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

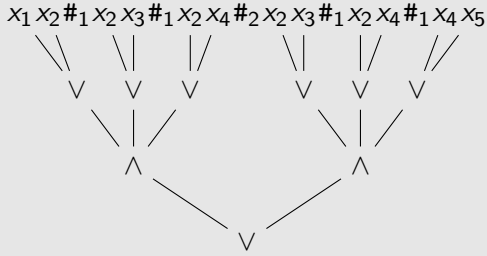


- Remove sharing. → remains of poly size
- Duplicate and order variables.
- Add delimiters.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)

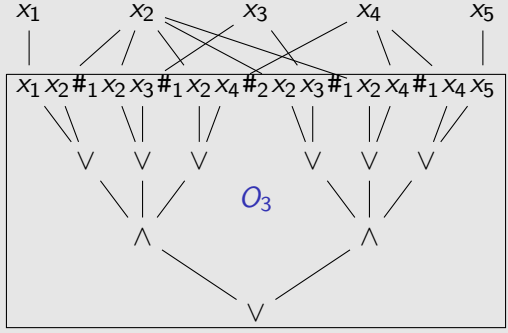


- Remove sharing. $\rightarrow$ remains of poly size
- Duplicate and order variables.
- Add delimiters.

Proof of completeness

Theorem
O_d (resp. A_d) is complete under projections for Σ_d (resp. Π_d).

Proof (O_d and A_d are complete)



- Remove sharing. $\rightarrow$ remains of poly size
- Duplicate and order variables.
- Add delimiters.

ADD and AC^0

ADD is regular:

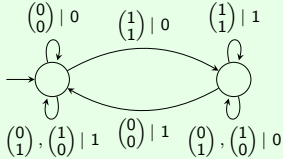
ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

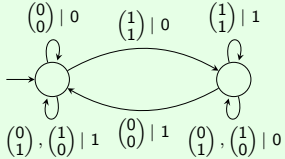
Example



ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example

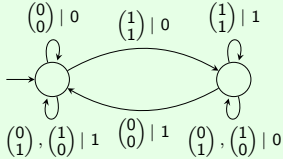


ADD is complete for AC^0 for a strong notion of reduction (but not projections).

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



ADD is complete for AC^0 for a strong notion of reduction (but not projections).

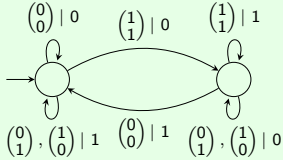
Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

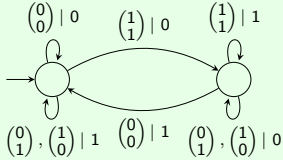
Proof

- Wlog. only $\neg$ and $\wedge$ gates.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

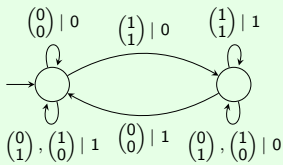
Proof

- Wlog. only $\neg$ and $\wedge$ gates.
- $\neg$-gate: $\neg x$ is the least significant bit of $1 + x$.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

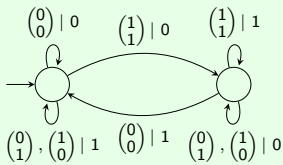
Proof

- Wlog. only $\neg$ and $\wedge$ gates.
- $\neg$-gate: $\neg x$ is the least significant bit of $1 + x$.
- $\wedge$-gate: $\bigwedge_{i=1}^{n} x_i$ is the most significant bit of $x_1 \cdots x_n + 1$.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

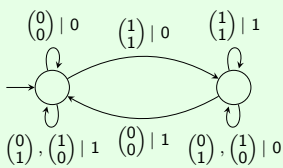
Proof

- Wlog. only $\neg$ and $\wedge$ gates.
- $\neg$-gate: $\neg x$ is the least significant bit of $1 + x$.
- $\wedge$-gate: $\bigwedge_{i=1}^{n} x_i$ is the most significant bit of $x_1 \cdots x_n + 1$.
- One ADD gate per layer, thanks to double zeroes.

ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



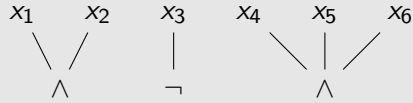
ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

Proof

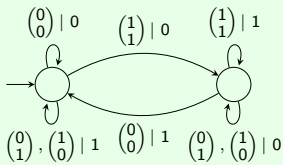
- Wlog. only $\neg$ and $\wedge$ gates.
- $\neg$-gate: $\neg x$ is the least significant bit of $1 + x$.
- $\wedge$-gate: $\bigwedge_{i=1}^{n} x_i$ is the most significant bit of $x_1 \cdots x_n + 1$.
- One ADD gate per layer, thanks to double zeroes.
- If we have a layer:



ADD and AC^0

ADD is regular:
$\rightarrow$ there is a finite automata that take a string $\binom{x_1}{y_1} \cdots \binom{x_n}{y_n}$ and outputs $x + y$.

Example



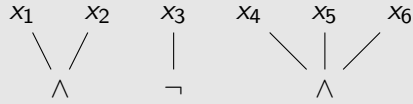
ADD is complete for AC^0 for a strong notion of reduction (but not projections).

Theorem

Every AC^0 language can be computed by a circuit with only a constant number of ADD gates of polynomial fan-in, and no other gates.

Proof

- Wlog. only $\neg$ and $\wedge$ gates.
- $\neg$-gate: $\neg x$ is the least significant bit of $1 + x$.
- $\wedge$-gate: $\bigwedge_{i=1}^{n} x_i$ is the most significant bit of $x_1 \cdots x_n + 1$.
- One ADD gate per layer, thanks to double zeroes.
- If we have a layer:



- Then we use:

	0	x_1	x_2	0	x_3	0	x_4	x_5	x_6
+	0	0	1	0	1	0	0	0	1
	$\downarrow$				$\downarrow$	$\downarrow$			

An algebraic toolbox

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$
- $(f : S \to S, \circ, \text{Id})$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$
- $(f : S \to S, \circ, \text{Id})$

Definition (Morphisms)

A morphism from M to N is a function $\mu : M \to N$ such that:

- $\mu(1_M) = 1_N$
- $\mu(x \cdot_M y) = \mu(x) \cdot_N \mu(y)$

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$
- $(f : S \to S, \circ, \text{Id})$

Definition (Morphisms)

A morphism from M to N is a function $\mu : M \to N$ such that:

- $\mu(1_M) = 1_N$
- $\mu(x \cdot_M y) = \mu(x) \cdot_N \mu(y)$

Examples

- $x \mapsto 2x$ is a morphism from $(\mathbb{N}, +)$ to itself.

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$
- $(f : S \to S, \circ, \text{Id})$

Definition (Morphisms)

A morphism from M to N is a function $\mu : M \to N$ such that:

- $\mu(1_M) = 1_N$
- $\mu(x \cdot_M y) = \mu(x) \cdot_N \mu(y)$

Examples

- $x \mapsto 2x$ is a morphism from $(\mathbb{N}, +)$ to itself.
- The length function is a morphism from Σ^* to $(\mathbb{N}, +)$.

A new object

Definition (Monoids)

A monoid is a triplet $(M, \cdot, 1)$ where:

- M is a set.
- $\cdot$ is an operation $M \times M \to M$ that is associative $((x \cdot y) \cdot z = x \cdot (y \cdot z))$.
- 1 is a neutral element of M $(1 \cdot x = x \cdot 1 = x)$.

$\to$ A generalization of groups.

$\to$ Usually denoted by the base set M.

Examples

- $(\mathbb{N}, +, 0)$
- $(\mathbb{N}, \times, 1)$
- $(\{0, 1\}, \wedge, 1)$
- $(\{0, 1\}, \vee, 0)$
- $(\Sigma^*, \text{concat}, \epsilon)$
- $(f : S \to S, \circ, \text{Id})$

Definition (Morphisms)

A morphism from M to N is a function $\mu : M \to N$ such that:

- $\mu(1_M) = 1_N$
- $\mu(x \cdot_M y) = \mu(x) \cdot_N \mu(y)$

Examples

- $x \mapsto 2x$ is a morphism from $(\mathbb{N}, +)$ to itself.
- The length function is a morphism from Σ^* to $(\mathbb{N}, +)$.
- The function $\Sigma^* \to (\{0, 1\}, \vee)$ that maps a word to 1 if and only if it has some letter a is a morphism.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$
  - $\delta_a(x) = x \cdot \mu(a)$

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$
  - $\delta_a(x) = x \cdot \mu(a)$
  - $i = 1$

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$
  - $\delta_a(x) = x \cdot \mu(a)$
  - $i = 1$
  - $F = P$

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$
  - $\delta_a(x) = x \cdot \mu(a)$
  - $i = 1$
  - $F = P$
- Invariant: $\delta_w(i) = \mu(w)$.

Links with regular languages

Definition
A language L is recognized by a monoid M if there is a morphism $\mu : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \mu^{-1}(P)$.

$\to$ Membership in L can be decided by looking only at information contained in M.

Claim
A language L is regular if and only if it is recognized by a finite monoid.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ DFA for L.
- δ_w is the extended transition function when reading w.
- Let M the set of functions $\delta_w : Q \to Q$ with the composition.
  $\to$ it is finite.
- Let $\mu : \Sigma^* \to M$ that maps w to δ_w.
  $\to$ it is a morphism.
- Let $P = \{f \mid f(i) \in F\}$.
  $\to$ recognizes L.

- $\mu : \Sigma^* \to M$ such that $L = \mu^{-1}(P)$.
- Construct $\mathcal{A}$ with:
  - $Q = M$
  - $\delta_a(x) = x \cdot \mu(a)$
  - $i = 1$
  - $F = P$
- Invariant: $\delta_w(i) = \mu(w)$.

$\to$ This is the transition monoid of $\mathcal{A}$. It makes more structure visible.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,

$$xuy \in L \Leftrightarrow xvy \in L.$$

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,

$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,

$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity. Its syntactic relation has two classes:
- words with an even number of 1.
- words with an odd number of 1.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity. Its syntactic relation has two classes:
- words with an even number of 1.
- words with an odd number of 1.

Its syntactic monoid is the group $\mathbb{Z}/2\mathbb{Z}$.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity. Its syntactic relation has two classes:
- words with an even number of 1.
- words with an odd number of 1.

Its syntactic monoid is the group $\mathbb{Z}/2\mathbb{Z}$.

Example
Consider the language of words with a 1.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity. Its syntactic relation has two classes:
- words with an even number of 1.
- words with an odd number of 1.

Its syntactic monoid is the group $\mathbb{Z}/2\mathbb{Z}$.

Example
Consider the language of words with a 1.
Its syntactic relation has two classes:
- words with a 1.
- words without a 1.

A canonical monoid

We want to associates a distinguished monoid to every regular language.

Definition
The syntactic relation of L is the relation on Σ^* defined by $u \sim_L v$ iff for all x, y,
$$xuy \in L \Leftrightarrow xvy \in L.$$

$\rightarrow$ an equivalence relation.
$\rightarrow$ Meaning: we can replace u by v anywhere without changing membership in L.

Definition
The syntactic monoid M_L of L is the set of equivalence classes of $\sim_L$ equiped with: for $C_1, C_2 \in M_L$, and $u \in C_1$ and $v \in C_2$, $C_1 \cdot C_2$ is the class of uv.

$\rightarrow$ We have to check that this is well defined: the class $C_1 \cdot C_2$ does not depend on the choice of u and v.

Claim
If $u \sim_L u'$ and $v \sim_L v'$, then $uv \sim_L u'v'$.

Proof
- Let x, y such that $xuvy \in L$.
- Equivalent to $xu'vy \in L$ by $u \sim_L u'$.
- Equivalent to $xu'v'y \in L$ by $v \sim_L v'$.

Example
Consider Parity. Its syntactic relation has two classes:
- words with an even number of 1.
- words with an odd number of 1.

Its syntactic monoid is the group $\mathbb{Z}/2\mathbb{Z}$.

Example
Consider the language of words with a 1.
Its syntactic relation has two classes:
- words with a 1.
- words without a 1.

Its syntactic monoid is $(\{0, 1\}, \vee)$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof
- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow M_L$ is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.
  - For all y, we have
    $xuy \in L \Leftrightarrow xvy \in L$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow M_L$ is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.
  - For all y, we have
    $xuy \in L \Leftrightarrow xvy \in L$.
  - Thus, $\delta_{uy}(p) \in F \Leftrightarrow \delta_{vy}(p) \in F$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.
  - For all y, we have
    $xuy \in L \Leftrightarrow xvy \in L$.
  - Thus, $\delta_{uy}(p) \in F \Leftrightarrow \delta_{vy}(p) \in F$.
  - Thus,
    $\delta_y(\delta_u(p)) \in F \Leftrightarrow \delta_y(\delta_v(p)) \in F$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.
  - For all y, we have
    $xuy \in L \Leftrightarrow xvy \in L$.
  - Thus, $\delta_{uy}(p) \in F \Leftrightarrow \delta_{vy}(p) \in F$.
  - Thus,
    $\delta_y(\delta_u(p)) \in F \Leftrightarrow \delta_y(\delta_v(p)) \in F$.
  - By minimality, $\delta_u(p) = \delta_v(p)$.

Another view on the syntactic monoic

Claim
The syntactic monoid is the transition monoid of the minimal automaton of L.

$\rightarrow$ M_L is finite and recognizes L.

Proof

- $\mathcal{A} = (Q, \delta, i, F)$ minimal DFA for L.
  $\rightarrow$ satisifes $p = q$ whenever
  $\delta_x(p) \in F \Leftrightarrow \delta_x(q) \in F$ for all x.
- We need: $u \sim_L v$ iff $\delta_u = \delta_v$.
- $\Leftarrow$: if $\delta_u = \delta_v$.
  - For x, y, assume $xuy \in L$.
  - Thus $\delta_{xuy}(i) \in F$.
  - Thus $\delta_{xvy}(i) \in F$.
  - Thus $xvy \in L$.
  - Hence $u \sim_L v$.
- $\Rightarrow$: if $u \sim_L v$.
  - Let $p \in Q$: by minimality, there is x such that $\delta_x(i) = p$.
  - For all y, we have
    $xuy \in L \Leftrightarrow xvy \in L$.
  - Thus, $\delta_{uy}(p) \in F \Leftrightarrow \delta_{vy}(p) \in F$.
  - Thus,
    $\delta_y(\delta_u(p)) \in F \Leftrightarrow \delta_y(\delta_v(p)) \in F$.
  - By minimality, $\delta_u(p) = \delta_v(p)$.
  - Hence, $\delta_u = \delta_v$.

$\rightarrow$ Also gives an algorithm to compute M_L.

An example

The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

An example

The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.
$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

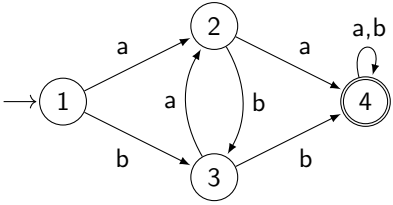
An example

The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton



Syntactic monoid

	1	2	3	4

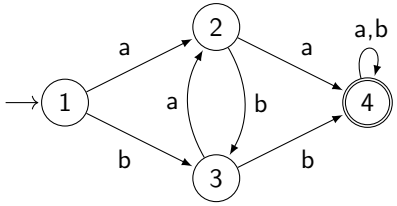
An example

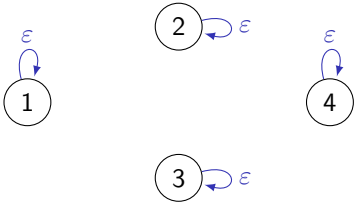
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4

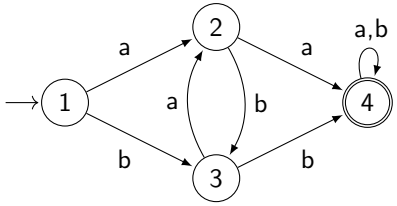
An example

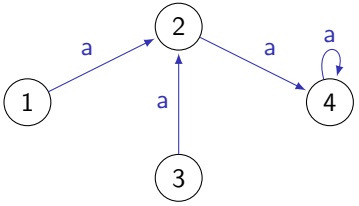
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4

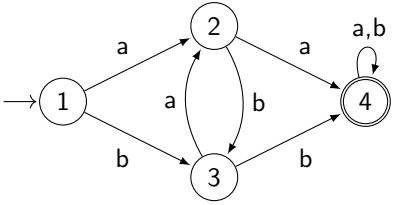
An example

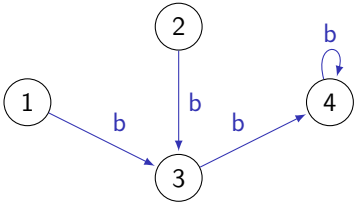
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4

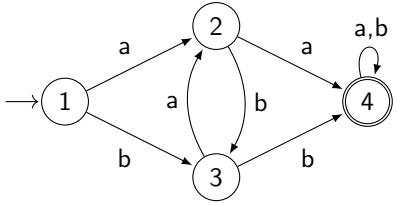
An example

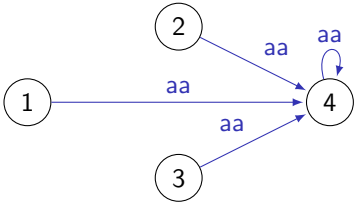
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4

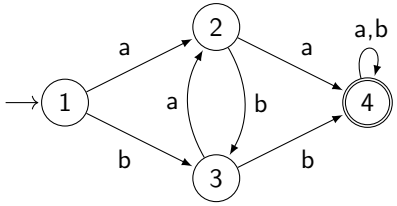
An example

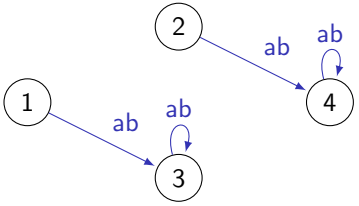
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4

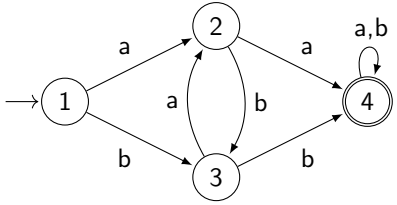
An example

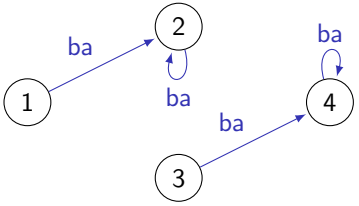
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

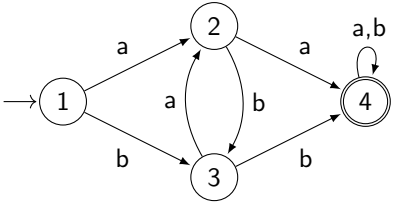
An example

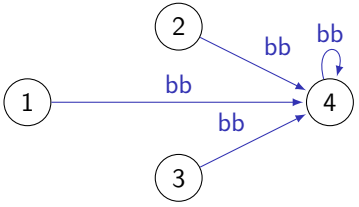
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\delta_b \cdot \delta_b \quad = \quad \delta_{aa}$$

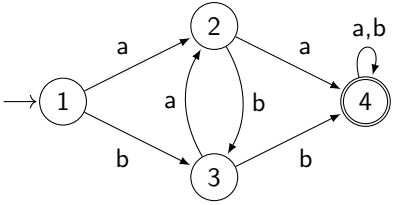
An example

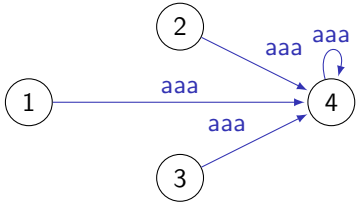
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\begin{aligned} \delta_b \cdot \delta_b &= \delta_{aa} \\ \delta_{aa} \cdot \delta_a &= \delta_{aa} \end{aligned}$$

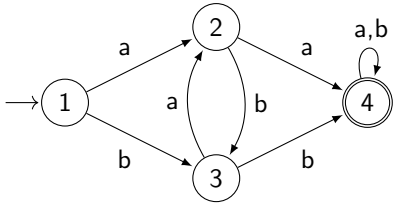
An example

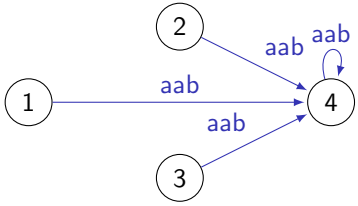
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\begin{aligned} \delta_b \cdot \delta_b &= \delta_{aa} \\ \delta_{aa} \cdot \delta_a &= \delta_{aa} \\ \delta_{aa} \cdot \delta_b &= \delta_{aa} \end{aligned}$$

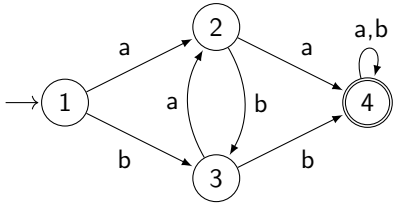
An example

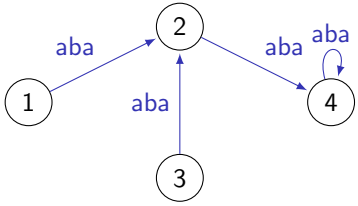
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\begin{aligned} \delta_b \cdot \delta_b &= \delta_{aa} \\ \delta_{aa} \cdot \delta_a &= \delta_{aa} \\ \delta_{aa} \cdot \delta_b &= \delta_{aa} \\ \delta_{ab} \cdot \delta_a &= \delta_a \end{aligned}$$

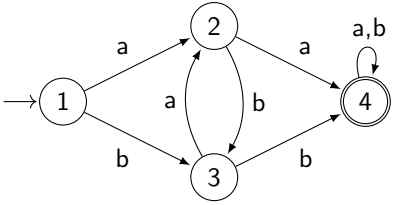
An example

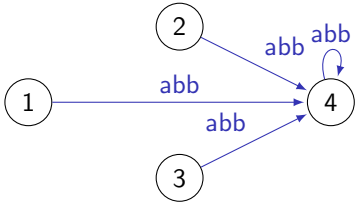
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\begin{aligned} \delta_b \cdot \delta_b &= \delta_{aa} \\ \delta_{aa} \cdot \delta_a &= \delta_{aa} \\ \delta_{aa} \cdot \delta_b &= \delta_{aa} \\ \delta_{ab} \cdot \delta_a &= \delta_a \\ \delta_{ab} \cdot \delta_b &= \delta_{aa} \end{aligned}$$

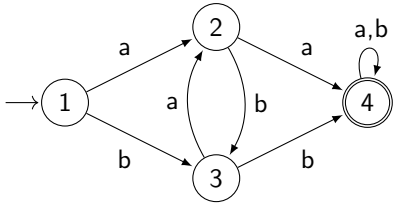
An example

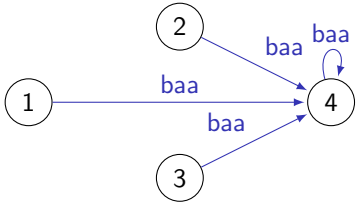
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$\begin{aligned}
\delta_b \cdot \delta_b &= \delta_{aa} \\
\delta_{aa} \cdot \delta_a &= \delta_{aa} \\
\delta_{aa} \cdot \delta_b &= \delta_{aa} \\
\delta_{ab} \cdot \delta_a &= \delta_a \\
\delta_{ab} \cdot \delta_b &= \delta_{aa} \\
\delta_{ba} \cdot \delta_a &= \delta_{aa}
\end{aligned}$$

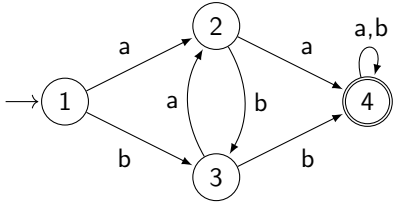
An example

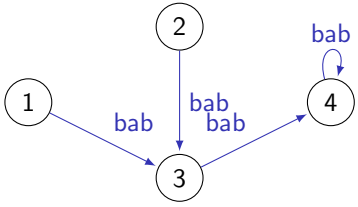
The syntactic monoid is generated by δ_a and δ_b: every element can be obtained are a product of these two.

$\rightarrow$ We can only describe the multiplication by δ_a and δ_b.

Computation of the syntactic monoid of $(a+b)^*(aa+bb)(a+b)^*$.

Minimal automaton





Syntactic monoid

	1	2	3	4
δ_ε	1	2	3	4
δ_a	2	4	2	4
δ_b	3	3	4	4
δ_{aa}	4	4	4	4
δ_{ab}	3	4	3	4
δ_{ba}	2	2	4	4

$$
\begin{aligned}
\delta_b \cdot \delta_b &= \delta_{aa} \\
\delta_{aa} \cdot \delta_a &= \delta_{aa} \\
\delta_{aa} \cdot \delta_b &= \delta_{aa} \\
\delta_{ab} \cdot \delta_a &= \delta_a \\
\delta_{ab} \cdot \delta_b &= \delta_{aa} \\
\delta_{ba} \cdot \delta_a &= \delta_{aa} \\
\delta_{ba} \cdot \delta_b &= \delta_b
\end{aligned}
$$

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

$\rightarrow$ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof

- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

$\rightarrow$ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
$\rightarrow$ We denote it x^ω.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

$\rightarrow$ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
$\rightarrow$ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

$\rightarrow$ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

$\rightarrow$ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
$\rightarrow$ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

$\rightarrow$ Intuition: M cannot count.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

→ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

→ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
→ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

→ Intuition: M cannot count.
→ A language L is aperiodic if M_L is.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

→ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

→ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
→ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

→ Intuition: M cannot count.
→ A language L is aperiodic if M_L is.

Example
Parity and $\mathbb{Z}/2\mathbb{Z}$ are not aperiodic: $1^\omega = 0$ and $1^{\omega+1} = 1$.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

→ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

→ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
→ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

→ Intuition: M cannot count.
→ A language L is aperiodic if M_L is.

Example
Parity and $\mathbb{Z}/2\mathbb{Z}$ are not aperiodic: $1^\omega = 0$ and $1^{\omega+1} = 1$.

→ For any $m \in \mathbb{N}$, Mod_m is also not aperiodic.

Idempotents

Definition
An idempotent is an element $x \in M$ such that $x^2 = x$.

→ Capture cycles in an automaton.

We can find idempotents easily:

Claim
For every $x \in M$, there is an $i \in \mathbb{N}$ such that x^i is idempotent.

Proof
- Take the sequence of x^i for any i.
- Pigeonhole: there is some $x^i = x^{i+p}$.
- For all $j \geq i$, $x^j = x^{j+p}$.
- For $j = kp$, $x^{kp} = x^{kp+p}$.
- x^{kp} is idempotent.

→ It is unique: if x^i and x^j are idempotents then $x^i = x^{ij} = x^j$.
→ We denote it x^ω.

The case $p = 1$ is particularly interesting.

Definition
A monoid M is aperiodic if for every $x \in M$, we have

$$x^{\omega+1} = x^\omega$$

→ Intuition: M cannot count.
→ A language L is aperiodic if M_L is.

Example
Parity and $\mathbb{Z}/2\mathbb{Z}$ are not aperiodic: $1^\omega = 0$ and $1^{\omega+1} = 1$.

→ For any $m \in \mathbb{N}$, Mod_m is also not aperiodic.

Example
$\Sigma^* a \Sigma^*$ and $(\{0,1\}, \vee)$ are aperiodic: $0^\omega = 0^{\omega+1} = 0$ and $1^\omega = 1^{\omega+1} = 1$.

The regular languages of AC^0

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
$$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* =$

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.
- $a^* =$

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.
- $a^* = (\emptyset^c b \emptyset^c)^c$.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.
- $a^* = (\emptyset^c b \emptyset^c)^c$.
- $(ab)^* =$

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1, i] \wedge C_2[i+1, n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.
- $a^* = (\emptyset^c b \emptyset^c)^c$.
- $(ab)^* = (b\emptyset^c + \emptyset^c a + \emptyset^c aa\emptyset^c + \emptyset^c bb\emptyset^c)^c$.

Star-free languages

Claim

AC^0 languages are close under Boolean operations (union, intersection, complement).

Proof

- A $\vee$-gate for union.
- A $\wedge$-gate for intersection.
- A $\neg$-gate for complement.

Claim

AC^0 languages are close under concatenation.

Proof

- Circuits C_1 and C_2 for L_1 and L_2.
- Guess the split position.
- $L_1 \cdot L_2$ computed by:
  $$\bigvee_{i=0}^{n} C_1[1,i] \wedge C_2[i+1,n].$$
- $2(n+1)$ circuits of polynomial size.

The problem is the Kleene star: Parity is the Kleene star of an AC^0 language.

Definition

A language is star-free if it is expressible with a regular expression with only:

- $\emptyset, \epsilon$ and a for $a \in \Sigma$,
- union,
- complement,
- concatenation.

$\rightarrow$ Complement needed to have infinite languages.

Examples

- $\Sigma^* = \emptyset^c$.
- $a^* = (\emptyset^c b \emptyset^c)^c$.
- $(ab)^* = (b\emptyset^c + \emptyset^c a + \emptyset^c aa\emptyset^c + \emptyset^c bb\emptyset^c)^c$.

Claim

$$\text{star-free} \subseteq AC^0$$

An algebraic characterization

For the converse, we are stuck... we need algebra!

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.

- union: for any u, v
- $ux^n v \in L_1 \cup L_2$ $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.
- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.

- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.
- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$
- $\Rightarrow ux^i u' \in L_1$ and $v' x^j v \in L_2$ with $i \geq n$ or $j \geq n$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.

- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.

- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$
- $\Rightarrow ux^i u' \in L_1$ and $v' x^j v \in L_2$ with $i \geq n$ or $j \geq n$.
- So $ux^{i+1} u' \in L_1$ or $v' x^{j+1} v \in L_2$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.

- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.

- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.
- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$
- $\Rightarrow ux^i u' \in L_1$ and $v' x^j v \in L_2$ with $i \geq n$ or $j \geq n$.
- So $ux^{i+1} u' \in L_1$ or $v' x^{j+1} v \in L_2$.
- Thus $ux^{2n+1} v \in L_1 L_2$.
- The equivalence is proved similarly.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.
- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$
- $\Rightarrow ux^i u' \in L_1$ and $v' x^j v \in L_2$ with $i \geq n$ or $j \geq n$.
- So $ux^{i+1} u' \in L_1$ or $v' x^{j+1} v \in L_2$.
- Thus $ux^{2n+1} v \in L_1 L_2$.
- The equivalence is proved similarly.
- $x^{2n} \sim_{L_1 L_2} x^{2n+1}$.

An algebraic characterization

For the converse, we are stuck... we need algebra!

Theorem (Schützenberger)

The star-free languages are precisely the aperiodic languages.

Proof ($\Rightarrow$)

- Induction on the expression.
- $M_\emptyset = \{1\}$.
- $M_\epsilon = (\{0,1\}, \vee)$.
- For $a \in \Sigma$, $M_a = \{0,1,2\}$ with $x \cdot y = \min(x+y, 2)$.
- For any L, $\sim_L = \sim_{L^c}$ thus $M_{L^c} = M_L$.
- For the last two case, let L_1 and L_2 be aperiodic.
- For $x \in \Sigma^*$, let n such that $x^n \sim_{L_1} x^{n+1}$ and $x^n \sim_{L_2} x^{n+1}$.
- union: for any u, v
- $ux^n v \in L_1 \cup L_2$
  $\Leftrightarrow ux^n v \in L_1$ or $ux^n v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1$ or $ux^{n+1} v \in L_2$
  $\Leftrightarrow ux^{n+1} v \in L_1 \cup L_2$
- $x^n \sim_{L_1 \cup L_2} x^{n+1}$.
- Concatenation: for any u, v
- $ux^{2n} v \in L_1 L_2$
- $\Rightarrow ux^i u' \in L_1$ and $v' x^j v \in L_2$ with $i \geq n$ or $j \geq n$.
- So $ux^{i+1} u' \in L_1$ or $v' x^{j+1} v \in L_2$.
- Thus $ux^{2n+1} v \in L_1 L_2$.
- The equivalence is proved similarly.
- $x^{2n} \sim_{L_1 L_2} x^{2n+1}$.

$\Leftarrow$ Much more complicated: need a structure theory of monoids (Green's theory).

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.
- Let q the smallest integer st $x^{\omega+q} = x^{\omega}$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.
- Let q the smallest integer st $x^{\omega+q} = x^{\omega}$.
- We want to reduce Mod_q to L.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.
- Let q the smallest integer st $x^{\omega+q} = x^{\omega}$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.
- Let q the smallest integer st $x^{\omega+q} = x^{\omega}$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^{\omega}$.
- Let q the smallest integer st $x^{\omega+q} = x^{\omega}$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^{\omega}v \in L$ and $ux^{\omega+1}v \notin L$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).
- Let f that send 0 to x^ω and 1 to $x^{\omega+1}$;

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).
- Let f that send 0 to x^ω and 1 to $x^{\omega+1}$;
- The reduction send w to $uf(w_0)\cdots f(w_n)v$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).
- Let f that send 0 to x^ω and 1 to $x^{\omega+1}$;
- The reduction send w to $uf(w_0)\cdots f(w_n)v$.
- This word is equivalent to $ux^{\omega+\mathrm{Parity}^c(w)}v$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).
- Let f that send 0 to x^ω and 1 to $x^{\omega+1}$;
- The reduction send w to $uf(w_0)\cdots f(w_n)v$.
- This word is equivalent to $ux^{\omega+\mathrm{Parity}^c(w)}v$.
- This is in L iff $\mathrm{Parity}(w) = 0$.

The theorem

A small technicality:

Definition

A language L has a neutral letter c if c can be added and removed anywhere without affecting membership in L.

Theorem (Barrington, Compton, Straubing, Thérien)

Let L be a regular language with a neutral letter, then

$$L \in \mathrm{AC}^0 \Leftrightarrow L \text{ is star-free.}$$

Proof

- L a regular language with a neutral letter that is not star-free.
- It is not aperiodic: there is $x \in M_L$ st $x^{\omega+1} \neq x^\omega$.
- Let q the smallest integer st $x^{\omega+q} = x^\omega$.
- We want to reduce Mod_q to L.
- $\mathrm{Mod}_q \notin \mathrm{AC}^0 \Rightarrow L \notin \mathrm{AC}^0$.
- We prove the case $q = 2$.
- Let u, v st (wlog.) $ux^\omega v \in L$ and $ux^{\omega+1}v \notin L$.
- Assume that x^ω and $x^{\omega+1}$ have same size (thanks to the neutral letter).
- Let f that send 0 to x^ω and 1 to $x^{\omega+1}$;
- The reduction send w to $uf(w_0)\cdots f(w_n)v$.
- This word is equivalent to $ux^{\omega+\mathrm{Parity}^c(w)}v$.
- This is in L iff $\mathrm{Parity}(w) = 0$.

Without a neutral letter, $(aa)^*$ is not star-free but is in AC^0.

Going further

We can identify the regular languages in a few other classes.

Going further

We can identify the regular languages in a few other classes.

Definition

A turtle program is a sequence (d_i, l_i) with $d_i \in \{\rightarrow, \leftarrow\}$ and $l_i \in \Sigma$.
The turle starts at position 1 and, for each instruction, moves on the direction d_i until it reaches a letter l_i. It fails if it does not find the letter at any point.

Going further

We can identify the regular languages in a few other classes.

Definition

A turtle program is a sequence (d_i, l_i) with $d_i \in \{\rightarrow, \leftarrow\}$ and $l_i \in \Sigma$.
The turle starts at position 1 and, for each instruction, moves on the direction d_i until it reaches a letter l_i. It fails if it does not find the letter at any point.

Theorem

The regular languages with a neutral letter of WLAC^0 are precisely the Boolean combination of turtle programs.

Going further

We can identify the regular languages in a few other classes.

Definition

A turtle program is a sequence (d_i, l_i) with $d_i \in \{\rightarrow, \leftarrow\}$ and $l_i \in \Sigma$.
The turle starts at position 1 and, for each instruction, moves on the direction d_i until it reaches a letter l_i. It fails if it does not find the letter at any point.

Theorem

The regular languages with a neutral letter of WLAC^0 are precisely the Boolean combination of turtle programs.

Definition

A subword language is a language L for which there exists u such that L is the set of words that have u as a subword.

Going further

We can identify the regular languages in a few other classes.

Definition

A turtle program is a sequence (d_i, l_i) with $d_i \in \{\rightarrow, \leftarrow\}$ and $l_i \in \Sigma$.
The turle starts at position 1 and, for each instruction, moves on the direction d_i until it reaches a letter l_i. It fails if it does not find the letter at any point.

Theorem

The regular languages with a neutral letter of WLAC^0 are precisely the Boolean combination of turtle programs.

Definition

A subword language is a language L for which there exists u such that L is the set of words that have u as a subword.

Theorem

The regular languages with a neutral letter that can be computed by k-DNFs with constant k are precisely the Boolean combination of subword languages.

Going further

We can identify the regular languages in a few other classes.

Definition

A turtle program is a sequence (d_i, l_i) with $d_i \in \{\rightarrow, \leftarrow\}$ and $l_i \in \Sigma$.
The turle starts at position 1 and, for each instruction, moves on the direction d_i until it reaches a letter l_i. It fails if it does not find the letter at any point.

Theorem

The regular languages with a neutral letter of WLAC^0 are precisely the Boolean combination of turtle programs.

Definition

A subword language is a language L for which there exists u such that L is the set of words that have u as a subword.

Theorem

The regular languages with a neutral letter that can be computed by k-DNFs with constant k are precisely the Boolean combination of subword languages.

$\rightarrow$ Can be extended to depth-3 but not depth-4 so far.