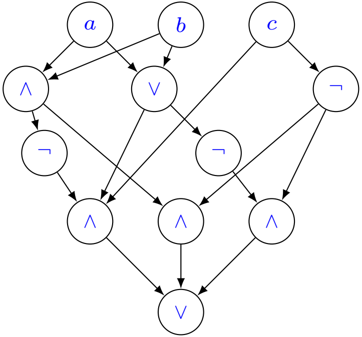
Boolean Circuits: a quick introduction

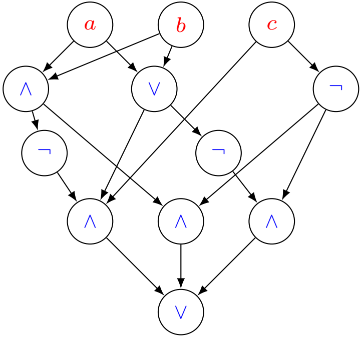
Charles Paperman, University of Lille

June 2020

A Boolean circuit

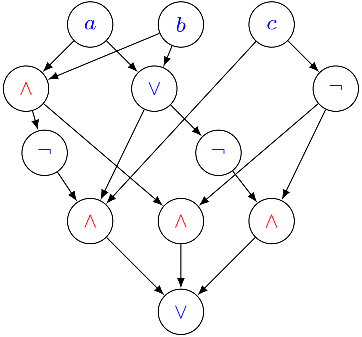


A Boolean circuit



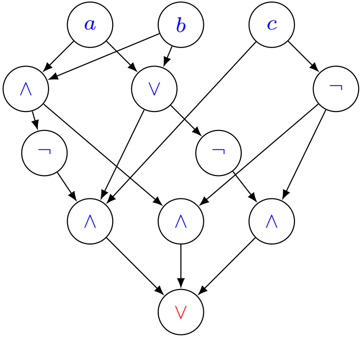
$a, b, c \in \{0, 1\}$

A Boolean circuit



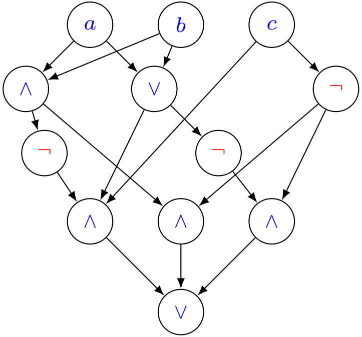
$\wedge \equiv$ All inputs must be one

A Boolean circuit



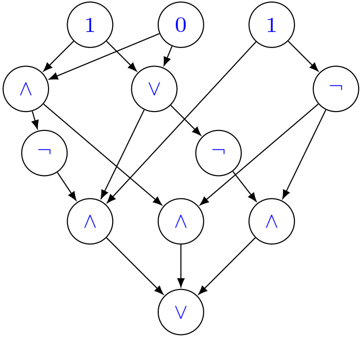
$\vee \equiv$ Some input must be one

A Boolean circuit

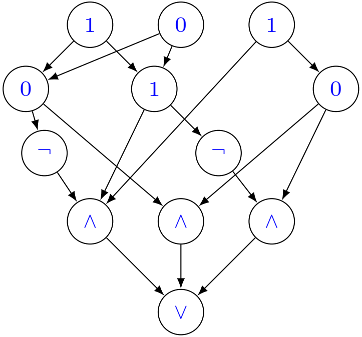


$\neg \equiv$ Negates its input

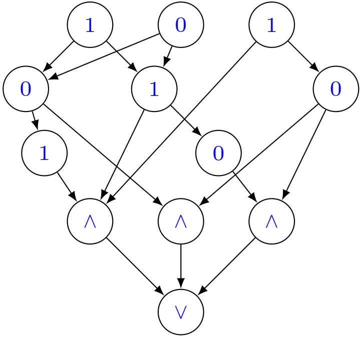
A Boolean circuit



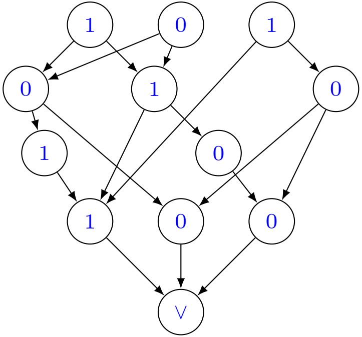
A Boolean circuit



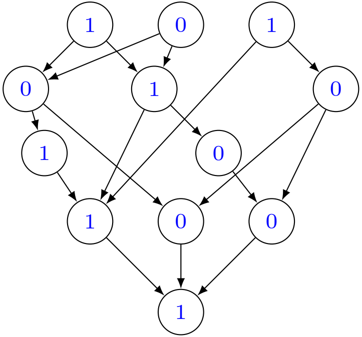
A Boolean circuit



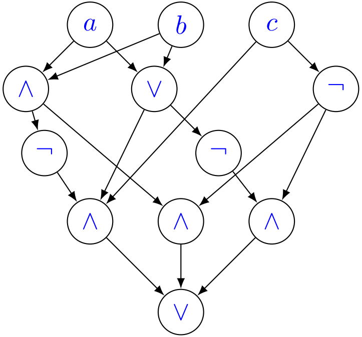
A Boolean circuit



A Boolean circuit



A Boolean circuit



Evaluate to 1 iff $a \oplus b = c$

Historical notes

1854 Simple mathematical reasoning requires few (simple) operations (Bool)

1936 Formalization of Turing Machine (Turing)

1937 Boolean algebra provides an abstraction of digital circuits design (Shannon)

1948 First transistor at Bell labs (John Bardeen, Walter Brattain, and William Shockley)

1958 First integrated circuit (Robert Noyce and Jack Kilby)

1965 Moore's Law

1971 The 4004 Intel processor (2250 transistors)

2019 AMD Ryzen 9 processor (10 billions transistors)

Boolean algebra

$X_n = \{x_1, \dots, x_n\}$ a set of Boolean variables (taking values in $\{0, 1\}$).

Basic operations:

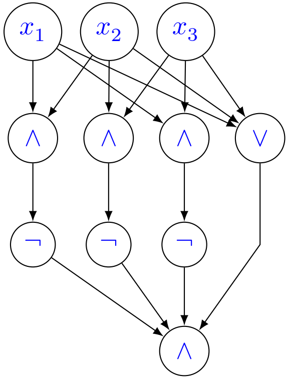
- $\wedge$ (AND)
- $\vee$ (OR)
- $\oplus$ (XOR)
- $\neg$ (NOT)

Boolean algebra

$X_n = \{x_1, \dots, x_n\}$ a set of Boolean variables (taking values in $\{0, 1\}$).

Basic operations:

- $\wedge$ (AND)
- $\vee$ (OR)
- $\oplus$ (XOR)
- $\neg$ (NOT)

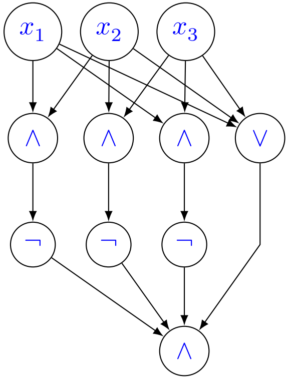


Boolean algebra

$X_n = \{x_1, \dots, x_n\}$ a set of Boolean variables (taking values in $\{0, 1\}$).

Basic operations:

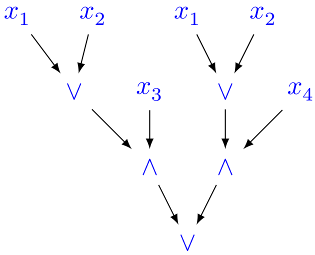
- $\wedge$ (AND)
- $\vee$ (OR)
- $\oplus$ (XOR)
- $\neg$ (NOT)



Compute $\{0, 1\}^3 \cap 0^*10^*$.

Boolean computation: Terms and Circuits

Terms



Boolean computation: Terms and Circuits

Terms

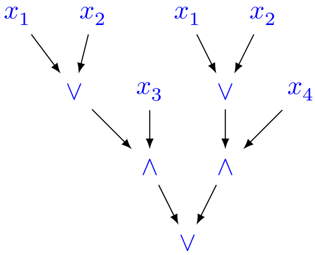
$$\begin{tikzcd}
x_1 \arrow[dr] & x_2 \arrow[d] & & x_1 \arrow[dr] & & x_2 \arrow[dl] & \\
& \vee \arrow[dr] & x_3 \arrow[d] & & \vee \arrow[d] & & x_4 \arrow[dll] \\
& & \wedge \arrow[dr] & & \wedge \arrow[dl] & & \\
& & & \vee & & &
\end{tikzcd}$$

Equivalent to

$((x_1 \text{ or } x_2) \text{ and } x_3) \text{ or } ((x_1 \text{ and } x_2) \text{ or } x_4)$

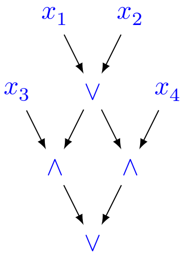
Boolean computation: Terms and Circuits

Terms



Equivalent to
$((x_1 \text{ or } x_2) \text{ and } x_3) \text{ or } ((x_1 \text{ and } x_2) \text{ or } x_4)$

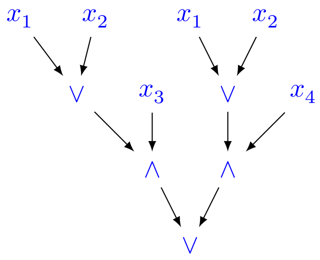
Circuits



Equivalent to
$(y \text{ and } x_3) \text{ or } (y \text{ or } x_4)$ with $y = x_1 \text{ or } x_2$.

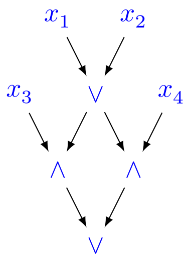
Boolean computation: Terms and Circuits

Terms



Equivalent to
$((x_1 \text{ or } x_2) \text{ and } x_3) \text{ or } ((x_1 \text{ and } x_2) \text{ or } x_4)$

Circuits



Equivalent to
$(y \text{ and } x_3) \text{ or } (y \text{ or } x_4)$ with $y = x_1 \text{ or } x_2$.

Circuits are factorized trees.

Small exercises

Exercice 1.

Any functions $f : \{0,1\}^n \to \{0,1\}$ can be computed by Boolean terms.

Small exercises

Exercice 1.

Any functions $f : \{0,1\}^n \to \{0,1\}$ can be computed by Boolean terms.

$$\bigvee_{f(u)=1} \bigwedge_{u_i=1} x_i \wedge \bigwedge_{u_i=0} \neg x_i$$

Small exercises

Exercice 1.

Any functions $f : \{0,1\}^n \to \{0,1\}$ can be computed by Boolean terms.

$$\bigvee_{f(u)=1} \bigwedge_{u_i=1} x_i \wedge \bigwedge_{u_i=0} \neg x_i$$

Exercice 2.

Any function computed by a Boolean circuit can be computed by a Boolean term.

Small exercises

Exercice 1.

Any functions $f : \{0,1\}^n \to \{0,1\}$ can be computed by Boolean terms.

$$\bigvee_{f(u)=1} \bigwedge_{u_i=1} x_i \wedge \bigwedge_{u_i=0} \neg x_i$$

Exercice 2.

Any function computed by a Boolean circuit can be computed by a Boolean term.

Simply by unfolding the circuits.

Shannon master Thesis

A Symbolic Analysis of Relay and Switching Circuits. Shannon, MIT 1937.

Main contribution:

1. The arrangement of Electrical switch can be improved by using Boolean algebra reasoning.
2. Electrical switch can be used to execute Boolean circuits.

One of the first appearance of computational complexity!

Electronic and Boolean complexity

Electronic synthesis: process of turning an abstract specification of some computations into a working electronic design. Ultimately, the goal is to obtained the most efficient electronic layout.

What efficiency means?

- Size of the design
- Speed of stabilization
- Power consumption

All those electronic parameters can be translated into Boolean circuits parameters.

Electronic and Boolean complexity

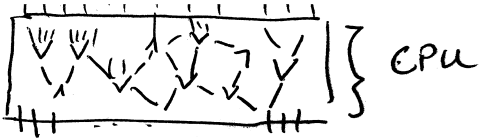
Electronic synthesis: process of turning an abstract specification of some computations into a working electronic design. Ultimately, the goal is to obtained the most efficient electronic layout.

What efficiency means?

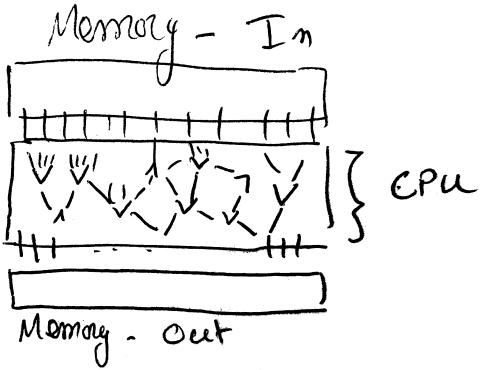
- Size of the design $\rightarrow$ number of gates
- Speed of stabilization $\rightarrow$ depth of the circuits
- Power consumption $\rightarrow$ number of wires

All those electronic parameters can be translated into Boolean circuits parameters.

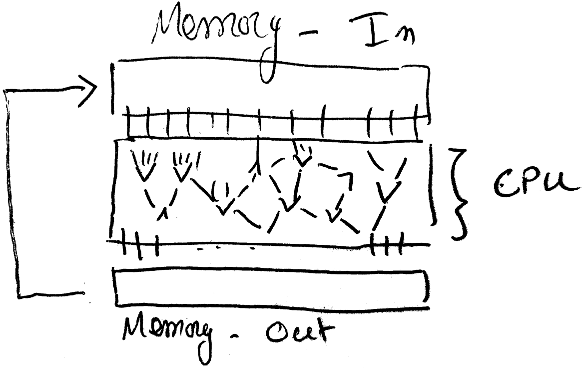
Classical complexity to Boolean realm



Classical complexity to Boolean realm

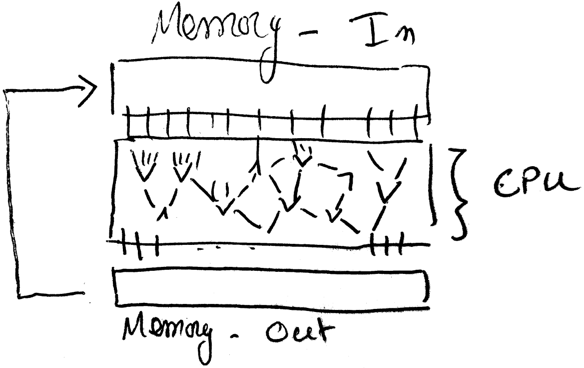


Classical complexity to Boolean realm



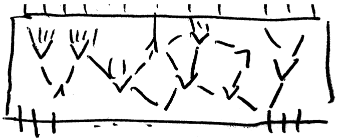
classical complexity focus on the number of loop and memory used by the algorithm.

Classical complexity to Boolean realm

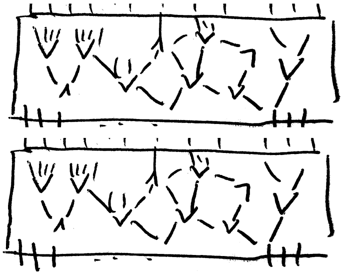


classical complexity focus on the number of loop and memory used by the algorithm.

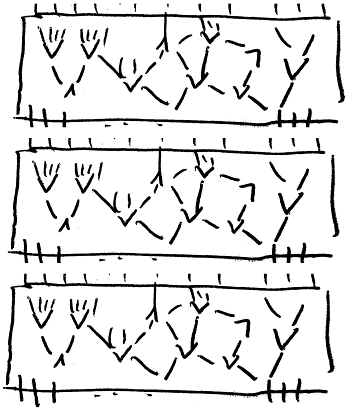
Classical complexity to Boolean realm



Classical complexity to Boolean realm



Classical complexity to Boolean realm



Summary

- Boolean complexity cares about Boolean circuits resources to perform computation
- Boolean complexity provides model-free notion of complexity
- Boolean complexity is linked to parallel complexity

Boolean Circuits Complexity

Remark.
Circuits have no memory model: they recognize finite subset of 2^{X_n}, for some n.

What Boolean complexity means when we care only of a finite subset?

An example: adding numbers

ADD_n is the function $\{0,1\}^{2n} \to \{0,1\}^{n+1}$ performing the addition on standard binary encoding of numbers on n-bits.

Exercice 3.

Using what you learn in middle school, prove that ADD_n is computable by a circuit with $\mathcal{O}(n)$ gates and $\mathcal{O}(n)$ depth.

An example: adding numbers

ADD_n is the function $\{0,1\}^{2n} \to \{0,1\}^{n+1}$ performing the addition on standard binary encoding of numbers on n-bits.

Exercice 3.

Using what you learn in middle school, prove that ADD_n is computable by a circuit with $\mathcal{O}(n)$ gates and $\mathcal{O}(n)$ depth.

Simply unroll the middle school classical algorithm.

A weird example: middle of inputs

Middle_n is the function (language) $\{0,1\}^{2n+1} \to \{0,1\}$ that map to 1 all words in $\{0,1\}^n \times \{1\} \times \{0,1\}^n$.

Exercice 4.

Prove that Middle_n is computable by a circuit with only one gate.

A weird example: middle of inputs

Middle_n is the function (language) $\{0,1\}^{2n+1} \to \{0,1\}$ that map to 1 all words in $\{0,1\}^n \times \{1\} \times \{0,1\}^n$.

Exercice 4.

Prove that Middle_n is computable by a circuit with only one gate.

Select the input x_{n+1} as output.

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n\in\mathbb{N}}$ be positive integers,
and $(f_n)_{n\in\mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n\in\mathbb{N}}$ be positive integers,
and $(f_n)_{n\in\mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Remark.

It is a non uniform notion of complexity ...

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n\in\mathbb{N}}$ be positive integers,
and $(f_n)_{n\in\mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Remark.

It is a non uniform notion of complexity ... and it makes sense.

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n\in\mathbb{N}}$ be positive integers,
and $(f_n)_{n\in\mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Remark.

It is a non uniform notion of complexity ... and it makes sense.

Exercice 5.

Propose a constant-complexity sequence of functions which is not Turing decidable

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n \in \mathbb{N}}$ be positive integers,
and $(f_n)_{n \in \mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Remark.

It is a non uniform notion of complexity ... and it makes sense.

Exercice 5.

Propose a constant-complexity sequence of functions which is not Turing decidable

Boolean circuits complexity: a formal definition

Definition.

Let $(r_n)_{n\in\mathbb{N}}$ be positive integers,
and $(f_n)_{n\in\mathbb{N}}$ be functions s.t. $f_n : \{0,1\}^n \to \{0,1\}^{r_n}$.

For $g : \mathbb{N} \to \mathbb{N}$, we say that (f_n) has a circuit-size $\mathcal{O}(g(n))$ if their exists a circuits family (C_n) such that C_n computes f_n and $|C_n| \in \mathcal{O}(g(n))$.

Similarly we can define the circuit-depth complexity or alternative notion by changing the set of considered gates.

Remark.

It is a non uniform notion of complexity ... and it makes sense.

Exercice 5.

Propose a constant-complexity sequence of functions which is not Turing decidable

Note on uniformity of Boolean classes

It is possible to add constraint on the way circuits classes are produces to lift the non-uniformity.

$\rightarrow$ To know more see Sipser's Book: Introduction to the Theory of Computation (chapter 10. Section Uniform Boolean circuits)

Some important classes

- $\mathrm{P/poly}$: functions computable by polysize circuits.
- AC^i: functions computable by polysize circuits and $\mathcal{O}(log^i(n))$ depth.
- NC^i: functions computable by bounded arity polysize circuits and $\mathcal{O}(log^i(n))$ depth.
- $\mathrm{AC} = \bigcup_i \mathrm{AC}^i$ and $\mathrm{NC} = \bigcup_i \mathrm{NC}^i$

Some important classes

- $\mathrm{P/poly}$: functions computable by polysize circuits.
- AC^i: functions computable by polysize circuits and $\mathcal{O}(log^i(n))$ depth.
- NC^i: functions computable by bounded arity polysize circuits and $\mathcal{O}(log^i(n))$ depth.
- $\mathrm{AC} = \bigcup_i \mathrm{AC}^i$ and $\mathrm{NC} = \bigcup_i \mathrm{NC}^i$

Some facts on $\mathrm{P/poly}$

Exercice 6.

Prove that P is in $\mathrm{P/poly}$

Some facts on $\mathrm{P/poly}$

Exercice 6.

Prove that P is in $\mathrm{P/poly}$

The following facts can be found with references in the complexity Zoo.

- Can also be define as P-machine with polysize advices.
- If it does not contains NP, then $\mathrm{P} \neq \mathrm{NP}$
- If it does contains PSPACE, then PSPACE collapse to the $\Sigma_2 \cap \Pi_2$

Exercice 7.

Proves that some function/language are not in $\mathrm{P/poly}$

Some facts on P/poly

Exercice 6.

Prove that P is in P/poly

The following facts can be found with references in the complexity Zoo.

- Can also be define as P-machine with polysize advices.
- If it does not contains NP, then $\mathrm{P} \neq \mathrm{NP}$
- If it does contains PSPACE, then PSPACE collapse to the $\Sigma_2 \cap \Pi_2$

Exercice 7.

Proves that some function/language are not in P/poly

Some facts on P/poly

Exercice 6.

Prove that P is in P/poly

The following facts can be found with references in the complexity Zoo.

- Can also be define as P-machine with polysize advices.
- If it does not contains NP, then $\mathrm{P} \neq \mathrm{NP}$
- If it does contains PSPACE, then PSPACE collapse to the $\Sigma_2 \cap \Pi_2$

Exercice 7.

Proves that some function/language are not in P/poly

Some facts on the NC hierarchy

- Worst name for a complexity class (Nick Class).
- Known as the classes of parallel computations.
- Strictness is a long standing open question.
- Not much is known ...

Exercice 8.

Prove that $\mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$. Deduce that $\mathrm{AC} = \mathrm{NC}$ and that Boolean matrix multiplication is in NC^2

Some facts on the NC hierarchy

- Worst name for a complexity class (Nick Class).
- Known as the classes of parallel computations.
- Strictness is a long standing open question.
- Not much is known ...

Exercice 8.

Prove that $\mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$. Deduce that $\mathrm{AC} = \mathrm{NC}$ and that Boolean matrix multiplication is in NC^2

Exercice 9.

Prove that NC^0 does not compute the language 1^*.

Some facts on the NC hierarchy

- Worst name for a complexity class (Nick Class).
- Known as the classes of parallel computations.
- Strictness is a long standing open question.
- Not much is known ...

Exercice 8.

Prove that $\mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$. Deduce that $\mathrm{AC} = \mathrm{NC}$ and that Boolean matrix multiplication is in NC^2

Exercice 9.

Prove that NC^0 does not compute the language 1^*.

Some facts about NC^1

Unlike other, a lot is known about NC^1.

- The class of divides and conquers algorithms
- Contains all regular languages (and much more)
- Regular languages are actually complete for NC^1.
- Look at Barrington's Theorem.

Exercice 10.

Prove that Integer multiplication is in NC^1

Some facts about NC^1

Unlike other, a lot is known about NC^1.

- The class of divides and conquers algorithms
- Contains all regular languages (and much more)
- Regular languages are actually complete for NC^1.
- Look at Barrington's Theorem.

Exercice 10.

Prove that Integer multiplication is in NC^1

Exercice 11.

Prove that Dyck languages are in NC^1

Some facts about NC^1

Unlike other, a lot is known about NC^1.

- The class of divides and conquers algorithms
- Contains all regular languages (and much more)
- Regular languages are actually complete for NC^1.
- Look at Barrington's Theorem.

Exercice 10.

Prove that Integer multiplication is in NC^1

Exercice 11.

Prove that Dyck languages are in NC^1

Some facts about AC^0

- The class of highly efficiently parallelizable computation.
- Match with reasonnable fragment of SQL (see Conjunctive query).
- One of the few actual class with actual lower-bound known: $\mathrm{Parity} \notin \mathrm{AC}^0$ is a classical result (see Arora ans Barak book).

Some facts about AC^0

- The class of highly efficiently parallelizable computation.
- Match with reasonnable fragment of SQL (see Conjunctive query).
- One of the few actual class with actual lower-bound known: $\mathrm{Parity} \notin \mathrm{AC}^0$ is a classical result (see Arora ans Barak book).

Exercice 12.

Prove that (ADD_n) is in AC^0

Some facts about AC^0

- The class of highly efficiently parallelizable computation.
- Match with reasonnable fragment of SQL (see Conjunctive query).
- One of the few actual class with actual lower-bound known: $\mathrm{Parity} \notin \mathrm{AC}^0$ is a classical result (see Arora ans Barak book).

Exercice 12.

Prove that (ADD_n) is in AC^0

Proving that (ADD_n) is not computable by linear size circuits of AC^0 is a long standing open problem