A brief introduction to **computational complexity**

Arnaud Casteigts
LaBRI, Université de Bordeaux

January 17, 2023

Séminaire du LOMA
(Laboratoire Ondes et Matière d'Aquitaine)

Computational complexity?

Study of the computational resources required for solving a given **problem**.

Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

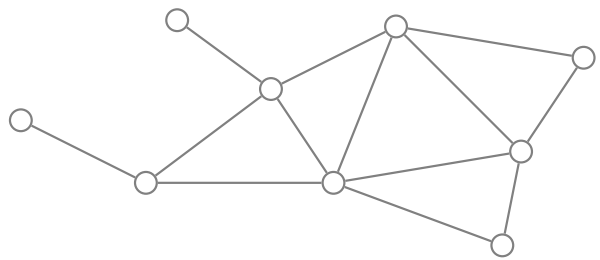
- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...

Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...

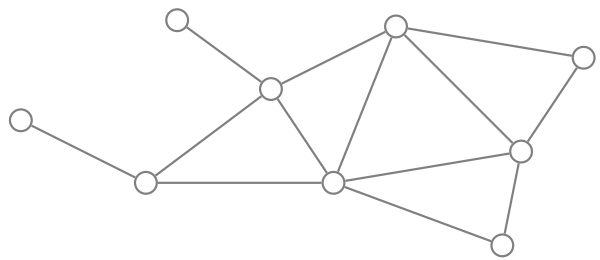


Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...



Formally, a **problem** is a **relation** between two domains: the **input** domain and the **output** domain.

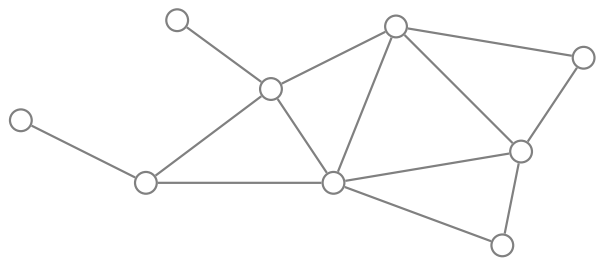
Equivalently, it is a **function** from the input domain to the output domain.

Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...



Formally, a **problem** is a **relation** between two domains: the **input** domain and the **output** domain.

Equivalently, it is a **function** from the input domain to the output domain.

Since computers (so far...) are discrete machines, both domains can be represented as sets of words over the $\{0, 1\}$ alphabet. Thus, a problem is a function from binary words to binary words:

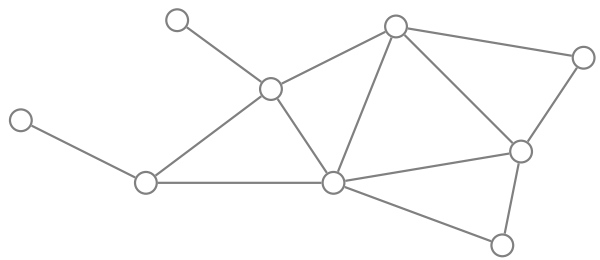
$$\text{Problem } \Pi : \{0,1\}^* \to \{0,1\}^*$$

Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...



Formally, a **problem** is a **relation** between two domains: the **input** domain and the **output** domain.

Equivalently, it is a **function** from the input domain to the output domain.

Since computers (so far...) are discrete machines, both domains can be represented as sets of words over the $\{0, 1\}$ alphabet. Thus, a problem is a function from binary words to binary words:

$$\text{Problem } \Pi : \{0,1\}^* \to \{0,1\}^*$$

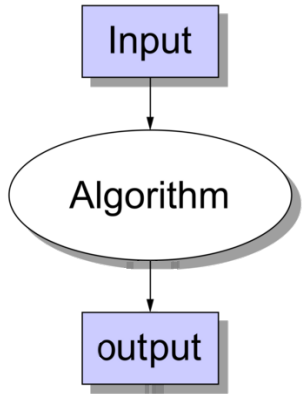
An algorithm **solves** Π if it computes $\Pi(x)$ for any x in $\{0, 1\}^*$ (or in a prescribed subset of $\{0, 1\}^*$).

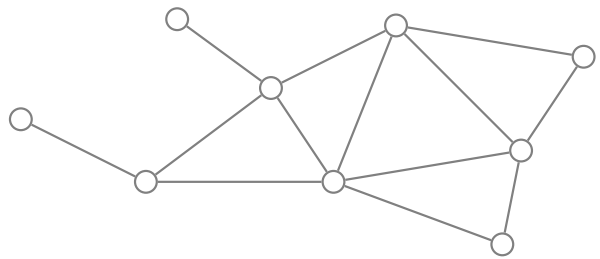
Computational complexity?

Study of the computational resources required for solving a given **problem**.

Examples of problems

- Given an integer, decide if it is prime
- Given a set of integers, compute their average
- Given a set of integers, sort them in ascending order
- Given a map, find the shortest route from a to b
- Given a set constraints, find a solution that satisfies them all
- Given a graph, decide if...
- ... it is connected
- ... it admits a 3-coloring
- ... it contains a clique of size 5
- ...





Formally, a **problem** is a **relation** between two domains: the **input** domain and the **output** domain.

Equivalently, it is a **function** from the input domain to the output domain.

Since computers (so far...) are discrete machines, both domains can be represented as sets of words over the $\{0,1\}$ alphabet. Thus, a problem is a function from binary words to binary words:

$$\text{Problem } \Pi : \{0,1\}^* \to \{0,1\}^*$$

An algorithm **solves** Π if it computes $\Pi(x)$ for any x in $\{0,1\}^*$ (or in a prescribed subset of $\{0,1\}^*$).

Types of problems

Different types of problems (examples based on coloring of a graph G)

- **Decision:** Does G admit a k-coloring (fir given k)?
- **Search:** Find a k-coloring of G (for a given k).
- **Counting:** How many k-colorings does G admit (for a given k)?
- **Optimisation:** What is the smallest k such that G admits a k-coloring.

Types of problems

Different types of problems (examples based on coloring of a graph G)

- **Decision:** Does G admit a k-coloring (fir given k)?
- **Search:** Find a k-coloring of G (for a given k).
- **Counting:** How many k-colorings does G admit (for a given k)?
- **Optimisation:** What is the smallest k such that G admits a k-coloring.

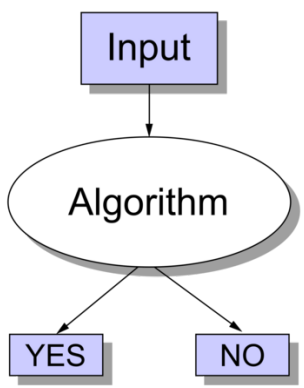
Decision problem (focus)

Special case where $\Pi : \{0,1\}^* \to \{0,1\}$ (the answer is YES or NO)

$x \in \{0,1\}^*$ is a **positive instance** if $\Pi(x) = 1$ (resp. **negative** if 0)

Positive instances define a "**formal language**" (ie a **set of words**) $L \subseteq \{0,1\}^*$

Point of view:
Solving a decision problem goes down to deciding if a given word is in L



What questions?

Goal of computational complexity:

$\rightarrow$ classify the problems according to the resources needed for solving them.

What type of resources?

- Time (= number of operations)
- Space (= memory)
- Randomness (number of random bits)
- Non-determinism
- ...

What questions?

Goal of computational complexity:

$\rightarrow$ classify the problems according to the resources needed for solving them.

What type of resources?

- Time (= number of operations)
- Space (= memory)
- Randomness (number of random bits)
- Non-determinism
- ...

Asymptotic point of view

- What matters is how these quantities **scale** with the **size** of the input, noted $\boldsymbol{n}$.
- Notations $O(\cdot), \Omega(\cdot), \Theta(\cdot), o(\cdot), \omega(\cdot)$

  Intuition $\leq \quad \geq \quad = \quad < \quad >$ for $n \to \infty$, up to constant factors
- Examples:

Constant	$O(1)$	Quadratic	$O(n^2)$
Logarithmic	$O(\log n)$	Polynomial	$O(n^c) = n^{O(1)} = poly(n)$
Linear	$O(n)$	Exponential	$O(2^n)$ ou $O(2^{poly(n)})$
Quasi-linear	$O(n \log n)$	Factorial	$O(n^n)$

Space and time

Unless otherwise mentioned, let us focus on **decision** problems

Generic classes

- $\text{TIME}(f(n))$: Set of problems solvable in time $O(f(n))$.
- $\text{SPACE}(f(n))$: Set of problems solvable in space $O(f(n))$.

Space and time

Unless otherwise mentioned, let us focus on **decision** problems

Generic classes

- $\mathrm{TIME}(f(n))$: Set of problems solvable in time $O(f(n))$.
- $\mathrm{SPACE}(f(n))$: Set of problems solvable in space $O(f(n))$.

Main classes of complexity

L	:=	$\mathrm{SPACE}(\log n)$	Problems solvable in **logarithmic space**
P	:=	$\mathrm{TIME}(poly(n))$	Problems solvable in **polynomial time**
PSPACE	:=	$\mathrm{SPACE}(poly(n))$	Problems solvable in **polynomial space**
EXP	:=	$\mathrm{TIME}(2^{poly(n)})$	Problems solvable in **exponentiel time**

Space and time

Unless otherwise mentioned, let us focus on **decision** problems

Generic classes

- $\text{TIME}(f(n))$: Set of problems solvable in time $O(f(n))$.
- $\text{SPACE}(f(n))$: Set of problems solvable in space $O(f(n))$.

Main classes of complexity

L	$:=$	$\text{SPACE}(\log n)$	Problems solvable in **logarithmic space**
P	$:=$	$\text{TIME}(poly(n))$	Problems solvable in **polynomial time**
PSPACE	$:=$	$\text{SPACE}(poly(n))$	Problems solvable in **polynomial space**
EXP	$:=$	$\text{TIME}(2^{poly(n)})$	Problems solvable in **exponentiel time**

$$\text{L} \subseteq \text{P} \subseteq \text{PSPACE} \subseteq \text{EXP}$$

Space and time

Unless otherwise mentioned, let us focus on **decision** problems

Generic classes

- $\mathrm{TIME}(f(n))$: Set of problems solvable in time $O(f(n))$.
- $\mathrm{SPACE}(f(n))$: Set of problems solvable in space $O(f(n))$.

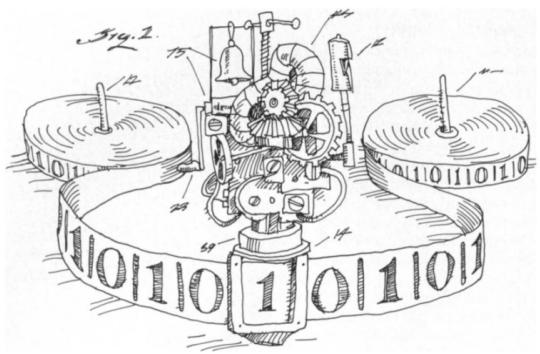
Main classes of complexity

L	:=	$\mathrm{SPACE}(\log n)$	Problems solvable in **logarithmic space**
P	:=	$\mathrm{TIME}(poly(n))$	Problems solvable in **polynomial time**
PSPACE	:=	$\mathrm{SPACE}(poly(n))$	Problems solvable in **polynomial space**
EXP	:=	$\mathrm{TIME}(2^{poly(n)})$	Problems solvable in **exponentiel time**

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

Machine model?

- Turing machines
- Intuition: “standard algorithm” OK
- Universality?



Space versus time (continued)

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

None of these inclusions is **known** to be strict (but all are **believed** to be so).

Space versus time (continued)

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

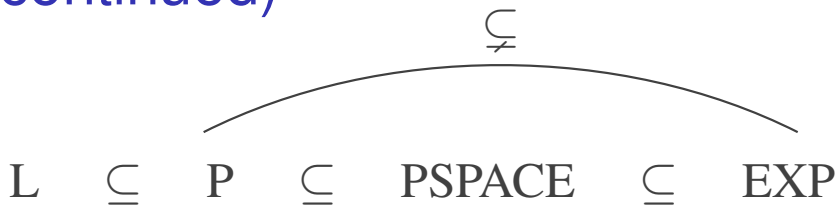
None of these inclusions is **known** to be strict (but all are **believed** to be so).

Theorems and hierarchies (simplified)

Given two fonctions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- Time hierarchy:
  $f_1 = o(f_2 \log f_2) \implies \mathrm{TIME}(f_1) \subsetneq \mathrm{TIME}(f_2)$ (Stearns and Hartmanis'65)

Space versus time (continued)



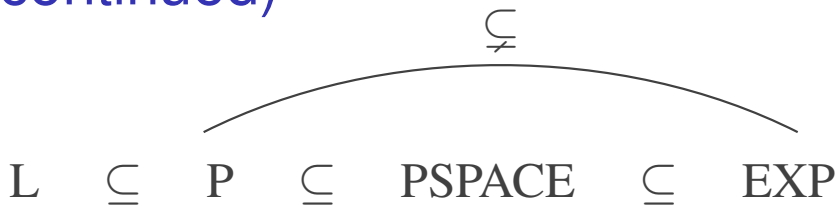
None of these inclusions is **known** to be strict (but all are **believed** to be so).

Theorems and hierarchies (simplified)

Given two fonctions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- Time hierarchy:
  $f_1 = o(f_2 \log f_2) \implies \text{TIME}(f_1) \subsetneq \text{TIME}(f_2)$ (Stearns and Hartmanis'65)

Space versus time (continued)



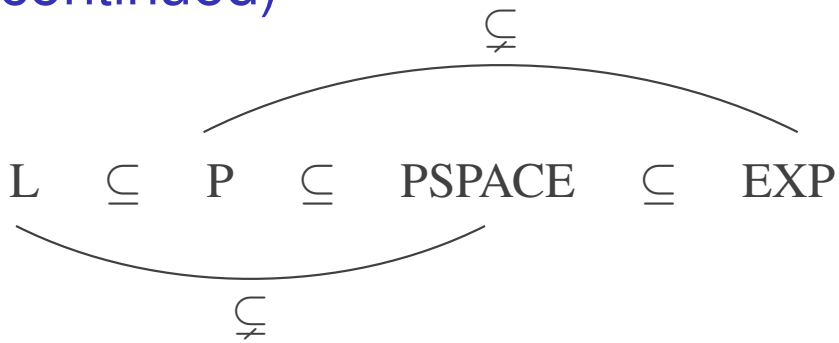
None of these inclusions is **known** to be strict (but all are **believed** to be so).

Theorems and hierarchies (simplified)

Given two fonctions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- Time hierarchy:
  $f_1 = o(f_2 \log f_2) \implies \text{TIME}(f_1) \subsetneq \text{TIME}(f_2)$ (Stearns and Hartmanis'65)
- Space hierarchy:
  $f_1 = o(f_2) \implies \text{SPACE}(f_1) \subsetneq \text{SPACE}(f_2)$

Space versus time (continued)



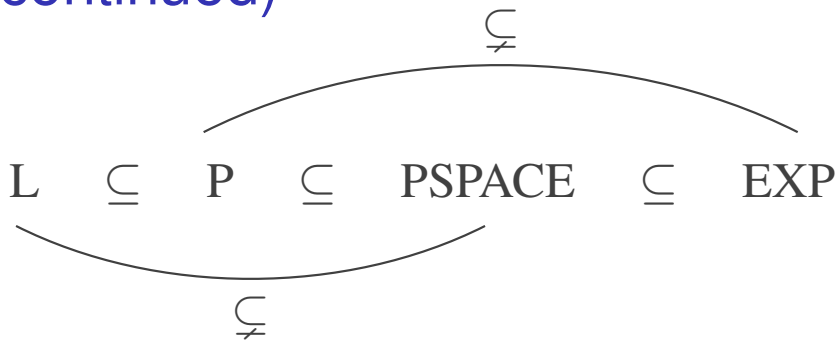
None of these inclusions is **known** to be strict (but all are **believed** to be so).

Theorems and hierarchies (simplified)

Given two fonctions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- Time hierarchy:
  $f_1 = o(f_2 \log f_2) \implies \text{TIME}(f_1) \subsetneq \text{TIME}(f_2)$ (Stearns and Hartmanis'65)
- Space hierarchy:
  $f_1 = o(f_2) \implies \text{SPACE}(f_1) \subsetneq \text{SPACE}(f_2)$

Space versus time (continued)



None of these inclusions is **known** to be strict (but all are **believed** to be so).

Theorems and hierarchies (simplified)

Given two fonctions $f_1(n)$ and $f_2(n)$ (abbreviated as f_1 and f_2 below):

- Time hierarchy:
  $f_1 = o(f_2 \log f_2) \implies \text{TIME}(f_1) \subsetneq \text{TIME}(f_2)$ (Stearns and Hartmanis'65)
- Space hierarchy:
  $f_1 = o(f_2) \implies \text{SPACE}(f_1) \subsetneq \text{SPACE}(f_2)$

Space versus time?

- $\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)/\log(f(n)))$ (Hopcroft, Paul, Valiant'77)
  $\rightarrow$ Space is **strictly** better than time!

Non-determinism (NP and coNP)

Several equivalent definitions, the simplest are:

- NP: $\exists$ **short proof** when the answer is YES (positive certificate)
- coNP: $\exists$ **short proof** when the answer is NO (negative certificate)

Short proof = one that is **verifiable** in polynomial time

Exemple : 3-COLORATION $\in$ NP (certificat = the coloring itself)

Non-determinism (NP and coNP)

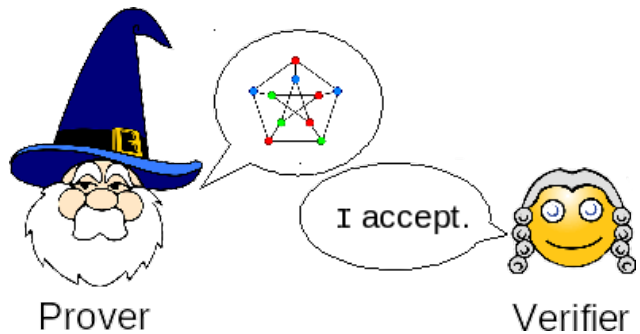
Several equivalent definitions, the simplest are:

NP: $\exists$ **short proof** when the answer is YES (positive certificate)

coNP: $\exists$ **short proof** when the answer is NO (negative certificate)

Short proof = one that is **verifiable** in polynomial time

Exemple : 3-COLORATION $\in$ NP (certificat = the coloring itself)



Non-determinism (NP and coNP)

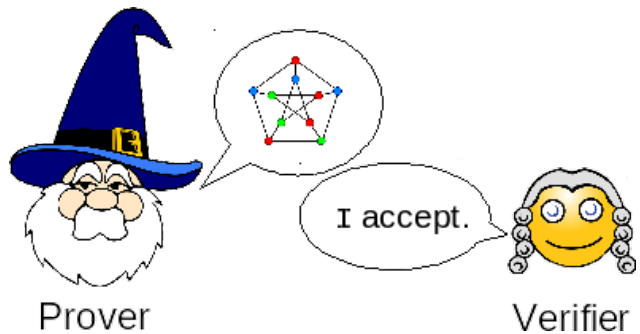
Several equivalent definitions, the simplest are:

NP: $\exists$ **short proof** when the answer is YES (positive certificate)

coNP: $\exists$ **short proof** when the answer is NO (negative certificate)

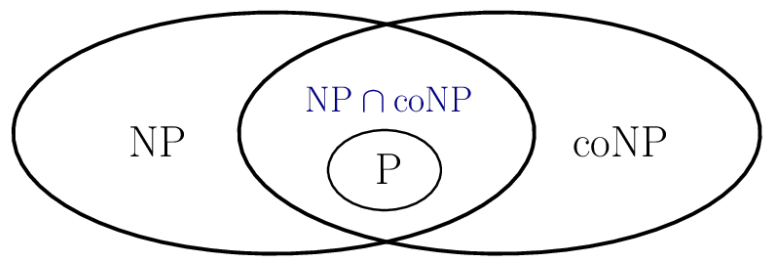
Short proof = one that is **verifiable** in polynomial time

Exemple : 3-COLORATION $\in$ NP (certificat = the coloring itself)



Exercise: categorize the following problems

- Is a given graph connected?
- Is a given number prime? is it composite?
- Does this number admit a factor $\leq k$?
- Are these two graphs isomorphic?
- Does a given graph admit a clique of size $n/2$?
- Does a given $n \times n$ chess board admit a winning strategy for player 1?
- Does this program always terminate?



Non-determinism (continued)

More generally

- $\mathrm{NTIME}(f(n))$: $\exists$ positive certificates verifiable in time $O(f(n))$.
- $\mathrm{NSPACE}(f(n))$: $\exists$ positive certificates verifiable in space $O(f(n))$.
- $\mathrm{coNTIME}$ and $\mathrm{coNSPACE}$: same for negative certificates.

Non-determinism (continued)

More generally

- NTIME($f(n)$): $\exists$ positive certificates verifiable in time $O(f(n))$.
- NSPACE($f(n)$): $\exists$ positive certificates verifiable in space $O(f(n))$.
- coNTIME and coNSPACE: same for negative certificates.

Main classes (same as before, but non-deterministic version)

$$
\begin{array}{lcl}
\text{NL} & := & \text{NSPACE}(\log n) \\
\text{NP} & := & \text{NTIME}(poly(n)) \\
\text{NPSPACE} & := & \text{NSPACE}(poly(n)) \\
\text{NEXP} & := & \text{NTIME}(2^{poly(n)})
\end{array}
$$

Non-determinism (continued)

More generally

- $\mathrm{NTIME}(f(n))$: $\exists$ positive certificates verifiable in time $O(f(n))$.
- $\mathrm{NSPACE}(f(n))$: $\exists$ positive certificates verifiable in space $O(f(n))$.
- coNTIME and coNSPACE: same for negative certificates.

Main classes (same as before, but non-deterministic version)

$$
\begin{array}{lcl}
\mathrm{NL} & := & \mathrm{NSPACE}(\log n) \\
\mathrm{NP} & := & \mathrm{NTIME}(poly(n)) \\
\mathrm{NPSPACE} & := & \mathrm{NSPACE}(poly(n)) \\
\mathrm{NEXP} & := & \mathrm{NTIME}(2^{poly(n)})
\end{array}
$$

Relations

- $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$ (algo = verifier with an empty certificate)
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch'70)
- $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$ (Immerman–Szelepcsényi'87)
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ (Certificate enumeration)

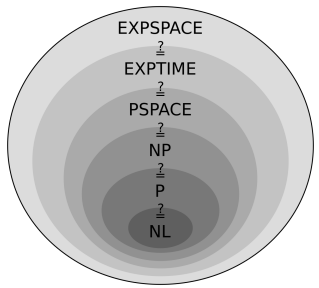
Non-determinism (continued)

More generally

- $\mathrm{NTIME}(f(n))$: $\exists$ positive certificates verifiable in time $O(f(n))$.
- $\mathrm{NSPACE}(f(n))$: $\exists$ positive certificates verifiable in space $O(f(n))$.
- $\mathrm{coNTIME}$ and $\mathrm{coNSPACE}$: same for negative certificates.

Main classes (same as before, but non-deterministic version)

$$
\begin{array}{lcl}
\mathrm{NL} & := & \mathrm{NSPACE}(\log n) \\
\mathrm{NP} & := & \mathrm{NTIME}(poly(n)) \\
\mathrm{NPSPACE} & := & \mathrm{NSPACE}(poly(n)) \\
\mathrm{NEXP} & := & \mathrm{NTIME}(2^{poly(n)})
\end{array}
$$



Relations

- $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$ (algo = verifier with an empty certificate)
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch'70)
- $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$ (Immerman–Szelepcsényi'87)
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ (Certificate enumeration)
- $\mathrm{L} \stackrel{?}{=} \mathrm{NL} \stackrel{?}{=} \mathrm{P} \stackrel{?}{=} \mathrm{NP} \stackrel{?}{=} \mathrm{PSPACE} \stackrel{?}{=} \mathrm{EXP} \stackrel{?}{=} \mathrm{NEXP} \stackrel{?}{=} \mathrm{EXPSPACE}$ (Inclusions OK)

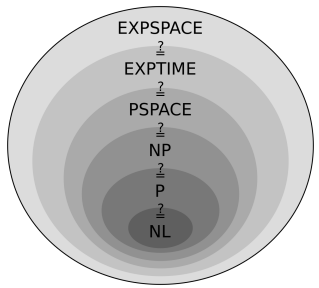
Non-determinism (continued)

More generally

- $\mathrm{NTIME}(f(n))$: $\exists$ positive certificates verifiable in time $O(f(n))$.
- $\mathrm{NSPACE}(f(n))$: $\exists$ positive certificates verifiable in space $O(f(n))$.
- $\mathrm{coNTIME}$ and $\mathrm{coNSPACE}$: same for negative certificates.

Main classes (same as before, but non-deterministic version)

$$
\begin{array}{lcl}
\mathrm{NL} & := & \mathrm{NSPACE}(\log n) \\
\mathrm{NP} & := & \mathrm{NTIME}(poly(n)) \\
\mathrm{NPSPACE} & := & \mathrm{NSPACE}(poly(n)) \\
\mathrm{NEXP} & := & \mathrm{NTIME}(2^{poly(n)})
\end{array}
$$



Relations

- $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$ (algo = verifier with an empty certificate)
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch'70)
- $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$ (Immerman–Szelepcsényi'87)
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ (Certificate enumeration)
- $\mathrm{L} \stackrel{?}{=} \mathrm{NL} \stackrel{?}{=} \mathrm{P} \stackrel{?}{=} \mathrm{NP} \stackrel{?}{=} \mathrm{PSPACE} \stackrel{?}{=} \mathrm{EXP} \stackrel{?}{=} \mathrm{NEXP} \stackrel{?}{=} \mathrm{EXPSPACE}$ (Inclusions OK)
- Padding arguments: $\mathrm{P} = \mathrm{NP} \implies \mathrm{EXP} = \mathrm{NEXP}$ (et d'autres)

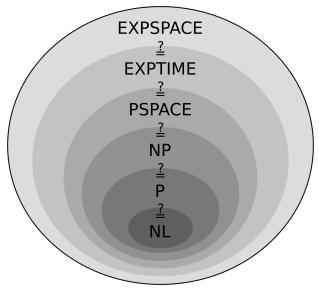
Non-determinism (continued)

More generally

- $\mathrm{NTIME}(f(n))$: $\exists$ positive certificates verifiable in time $O(f(n))$.
- $\mathrm{NSPACE}(f(n))$: $\exists$ positive certificates verifiable in space $O(f(n))$.
- $\mathrm{coNTIME}$ and $\mathrm{coNSPACE}$: same for negative certificates.

Main classes (same as before, but non-deterministic version)

$$
\begin{array}{lcl}
\mathrm{NL} & := & \mathrm{NSPACE}(\log n) \\
\mathrm{NP} & := & \mathrm{NTIME}(poly(n)) \\
\mathrm{NPSPACE} & := & \mathrm{NSPACE}(poly(n)) \\
\mathrm{NEXP} & := & \mathrm{NTIME}(2^{poly(n)})
\end{array}
$$



Relations

- $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$ (algo = verifier with an empty certificate)
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch'70)
- $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$ (Immerman–Szelepcsényi'87)
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ (Certificate enumeration)
- $\mathrm{L} \stackrel{?}{=} \mathrm{NL} \stackrel{?}{=} \mathrm{P} \stackrel{?}{=} \mathrm{NP} \stackrel{?}{=} \mathrm{PSPACE} \stackrel{?}{=} \mathrm{EXP} \stackrel{?}{=} \mathrm{NEXP} \stackrel{?}{=} \mathrm{EXPSPACE}$ (Inclusions OK)
- Padding arguments: $\mathrm{P} = \mathrm{NP} \implies \mathrm{EXP} = \mathrm{NEXP}$ (et d'autres)
- $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ **Does "easy to check" implies "easy to compute"?**

Reductions and completeness

Polynomial reduction

A problem Π_1 **reduces** in *polynomial time* to a problem Π_2, noted $\Pi_1 \leq \Pi_2$, if...

Reductions and completeness

Polynomial reduction

A problem Π_1 **reduces** in *polynomial time* to a problem Π_2, noted $\Pi_1 \leq \Pi_2$, if...

- (transformation-based) ... for all instance x_1 of Π_1, one can build in polynomial time an instance x_2 of Π_2 such that $\Pi_1(x_1) = \Pi_2(x_2)$.

Reductions and completeness

Polynomial reduction

A problem Π_1 **reduces** in *polynomial time* to a problem Π_2, noted $\Pi_1 \leq \Pi_2$, if...

- (transformation-based) ... for all instance x_1 of Π_1, one can build in polynomial time an instance x_2 of Π_2 such that $\Pi_1(x_1) = \Pi_2(x_2)$.
- (oracle-based) ... there exists an algorithm that solves Π_1 in polynomial time given access to **an oracle** for Π_2 (ie a machine that answers questions for Π_2 in unit time).

In other words, Π_2 is “at least as hard” as Π_1.

Reductions and completeness

Polynomial reduction

A problem Π_1 **reduces** in *polynomial time* to a problem Π_2, noted $\Pi_1 \leq \Pi_2$, if...

- (transformation-based) ... for all instance x_1 of Π_1, one can build in polynomial time an instance x_2 of Π_2 such that $\Pi_1(x_1) = \Pi_2(x_2)$.
- (oracle-based) ... there exists an algorithm that solves Π_1 in polynomial time given access to **an oracle** for Π_2 (ie a machine that answers questions for Π_2 in unit time).

In other words, Π_2 is “at least as hard” as Π_1.

Hardness and completeness

For a class $\mathcal{C}$ of problems and a type of reduction:

- $\mathcal{C}$-hard: at least as hard as any problem in $\mathcal{C}$
- $\mathcal{C}$-complete: both $\mathcal{C}$-hard and in $\mathcal{C}$

Reductions and completeness

Polynomial reduction

A problem Π_1 **reduces** in *polynomial time* to a problem Π_2, noted $\Pi_1 \leq \Pi_2$, if...

- (transformation-based) ... for all instance x_1 of Π_1, one can build in polynomial time an instance x_2 of Π_2 such that $\Pi_1(x_1) = \Pi_2(x_2)$.
- (oracle-based) ... there exists an algorithm that solves Π_1 in polynomial time given access to **an oracle** for Π_2 (ie a machine that answers questions for Π_2 in unit time).

In other words, Π_2 is “at least as hard” as Π_1.

Hardness and completeness

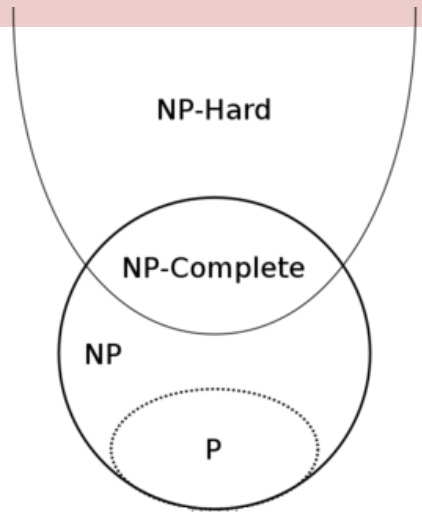
For a class $\mathcal{C}$ of problems and a type of reduction:

- $\mathcal{C}$-hard: at least as hard as any problem in $\mathcal{C}$
- $\mathcal{C}$-complete: both $\mathcal{C}$-hard and in $\mathcal{C}$

Ex: NP-complete = in NP and NP-hard

(here, *transformation-based* and *in polynomial time*)

SAT $\in$ NP-complet (Cook, Levin’71)



(what people believe)

Thanks!