# Calculabilité et Complexité
## (11X008)

Arnaud Casteigts

Bachelor en sciences informatiques,
Université de Genève

# Traitement de l'information

$$\texttt{Entrée} \longrightarrow \boxed{\text{Programme}} \longrightarrow \texttt{Sortie}$$

## Exemples

- Factorisation de nombres :

$$120 \longrightarrow \boxed{\text{Factoriser}} \longrightarrow 2^3 \cdot 3 \cdot 5$$

- Reconnaissance d'image :

$$\longrightarrow \boxed{\text{Chat ?}} \longrightarrow \text{Oui}$$

- Tri de données :

$$(4, 9, 2, 13, 0) \longrightarrow \boxed{\text{Trier}} \longrightarrow (0, 2, 4, 9, 13)$$

## Plus généralement

$$001011100110010 \longrightarrow \boxed{\text{Programme}} \longrightarrow 1100110$$

Un programme prend en entrée une suite de **symboles** et produit en sortie une autre suite de **symboles** (auquelles on donne un sens). Il réalise un **traitement**.

Ce traitement peut être vu comme une fonction $F$ de $\{0,1\}^*$ vers $\{0,1\}^*$.

# Types de problèmes

- **Décision :** $\{0,1\}^* \to \{0,1\}$
  Ex : est-ce que la photo représente un chat ?
  Ex : y a-t-il un chemin de A vers B ?
- **Recherche :** $\{0,1\}^* \to \{0,1\}^*$
  Ex : trouver un chat
  Ex : donner un chemin de A vers B
- **Dénombrement :** $\{0,1\}^* \to \mathbb{N}$
  Ex : combien y-a-t'il de chats ?
  Ex : combien y-a-t'il de chemins de A vers B
- **Optimisation :** $\{0,1\}^* \to \{0,1\}^*$
  Ex : trouver le chat le plus mignon.
  Ex : trouver le plus court chemin de A vers B

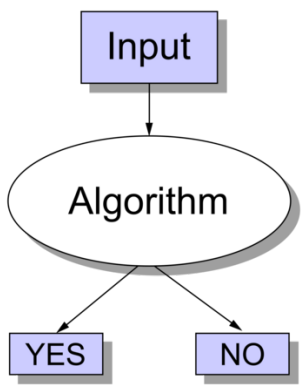
## Problème de décision (focus)

Fonctions $F$ du type $\{0,1\}^* \to \{0,1\}$ (réponse OUI ou NON)

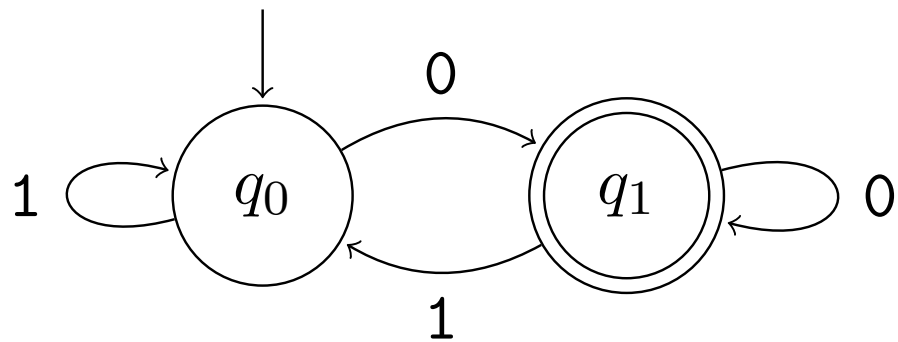
Ensemble des **instances** $x \in \{0,1\}^*$ telles que $F(x) = 1$

$\to$ **Langage formel** $L$

Résoudre un problème de décision $\equiv$ reconnaître un langage

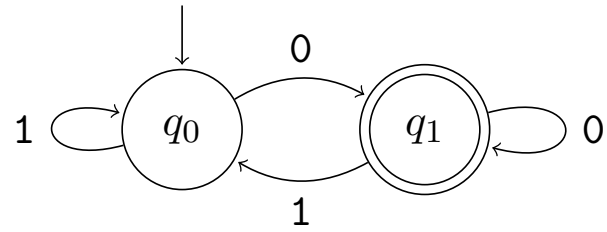


# Partie 1 : Calculabilité

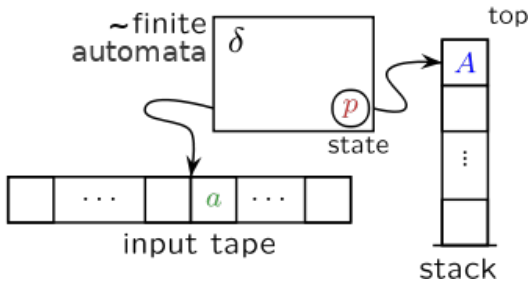


# Modèles de calcul et expressivité

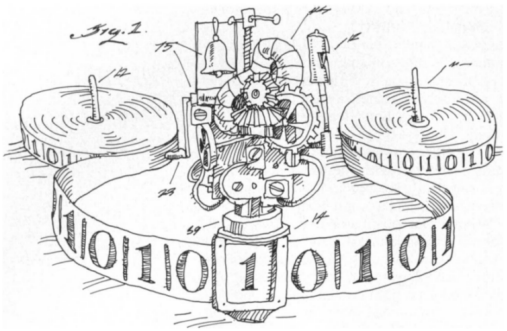
**Automate fini**




**Automate à pile**




**Machine de turing**



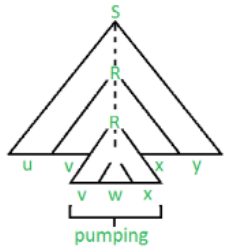
**Expressions régulières**

$$(a + b^*)^+ ca^*$$

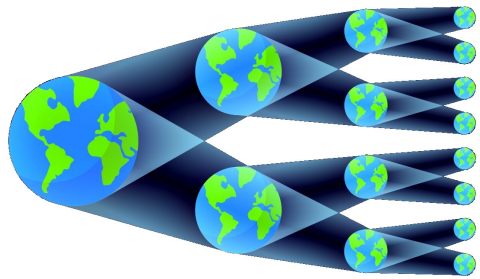
**Grammaires formelles**

$$S \to aSb$$
$$S \to \varepsilon$$

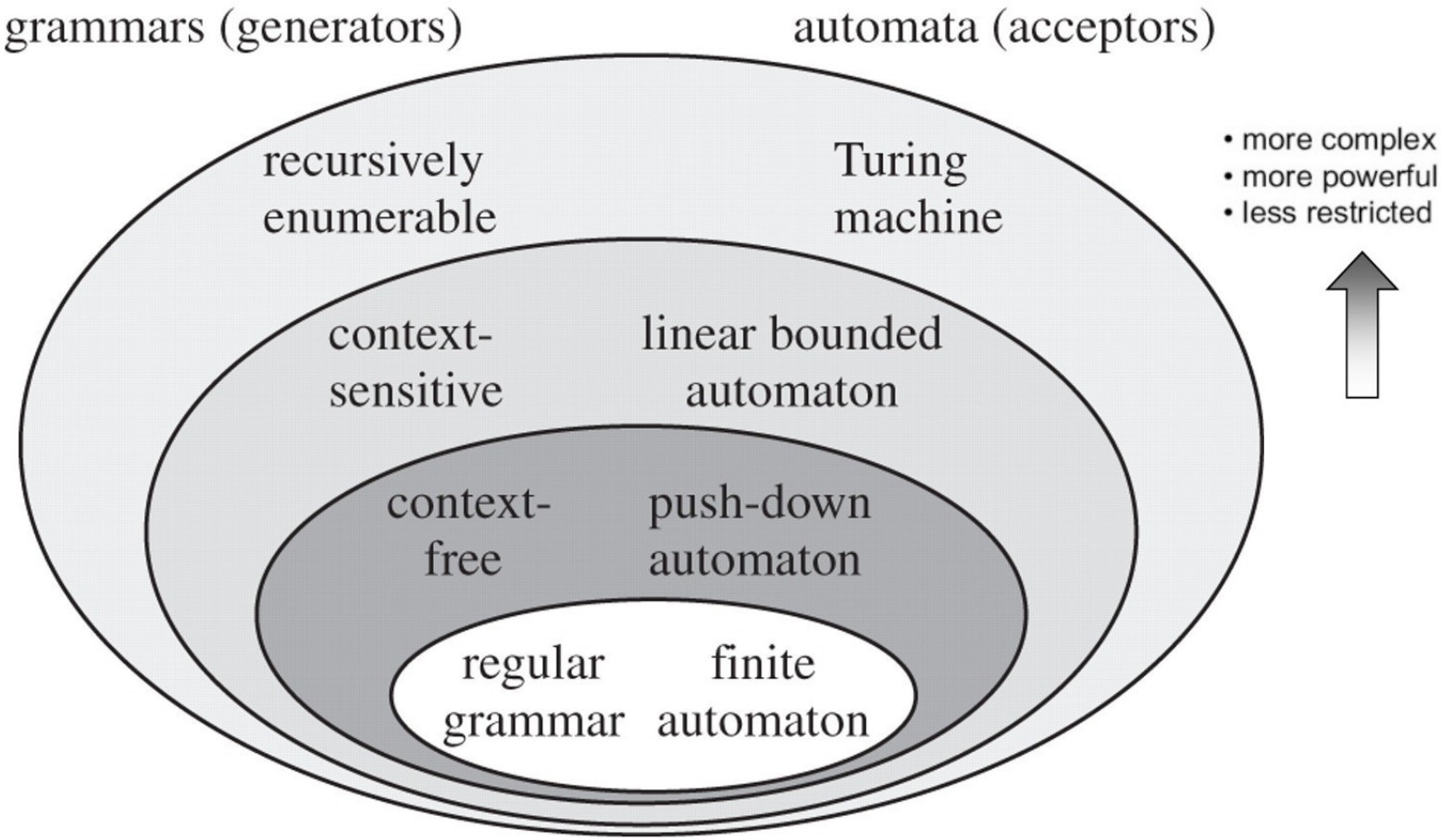
**Lemmes d'itérations**



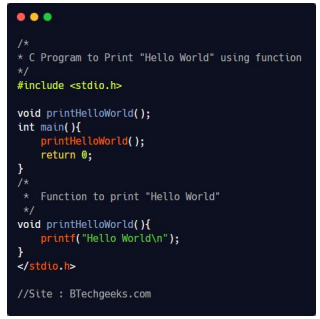

**Non-déterminisme**

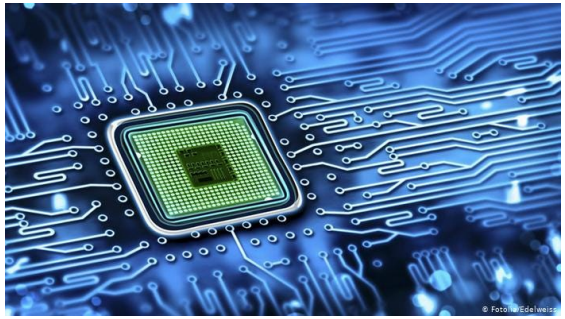


# Familles de langages correspondantes

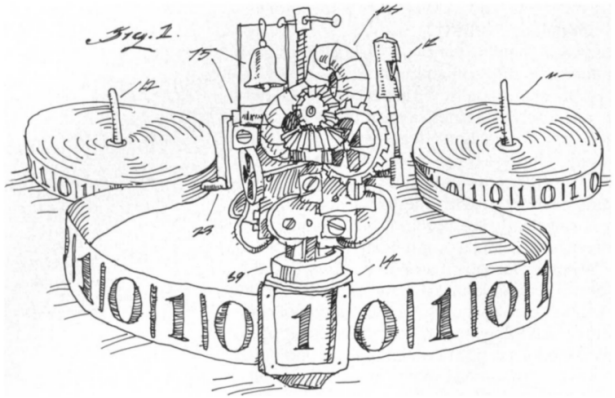


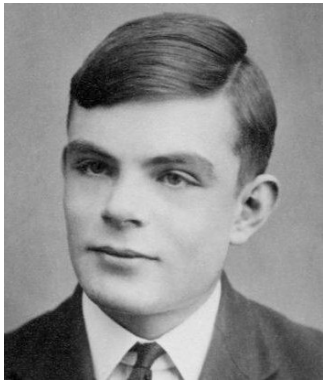
# Hypothèse de Church-Turing

## Machine de Turing



$\lambda$

Tout algorithme peut être décrit par une machine de Turing.

$\rightarrow$ À partir de maintenant, on reste au niveau des machines de Turing $\leftarrow$

# Décidable vs reconnaissable

Une machine $M$ **décide** un langage $L$ si :

- $w \in L \implies M(L)$ accepte $w$.
- $w \notin L \implies M(L)$ rejette $w$.

Une machine $M$ **reconnaît** un langage $L$ si :

- $w \in L \implies M(L)$ accepte $w$.
- $w \notin L \implies M(L)$ rejette $w$ ou boucle à l'infini.

Langage **décidable** :

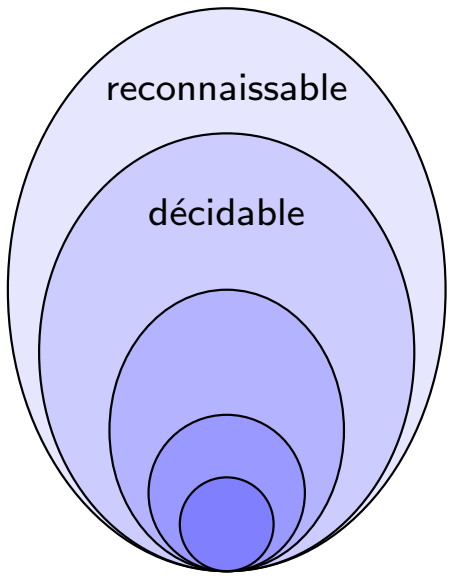
- Il existe une machine $M$ qui décide ce langage.

(synonymes : "récursif")

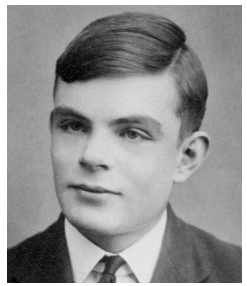
Langage **reconnaissable** :

- Il existe une machine $M$ qui reconnaît ce langage.

(synonymes : "récursivement énumérable" ou "semi-décidable")



# Patatras ! (Problème de l'arrêt)



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

*By* A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are,

Certains langages ne sont pas décidables !

Ex : langage $L = \{$(programme, entrée) | ce programme termine sur cette entrée$\}$.

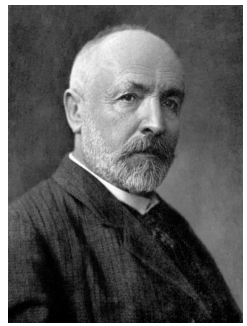
(... et beaucoup d'autres depuis)

Note : ce langage est quand même reconnaissable (pourquoi ?)

# Double patatras !

Certains langages ne sont même pas reconnaissables !

$$
\begin{array}{ll}
s_1 & = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\ldots \\
s_2 & = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\ldots \\
s_3 & = 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\ldots \\
s_4 & = 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\ldots \\
s_5 & = 1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\ldots \\
s_6 & = 0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\ldots \\
s_7 & = 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\ldots \\
s_8 & = 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\ldots \\
s_9 & = 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\ldots \\
s_{10} & = 1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\ldots \\
s_{11} & = 1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\ldots \\
\vdots & \quad \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\ddots
\end{array}
$$



(merci pour l'argument)

Argument **diagonal** inspiré de la preuve de Cantor montrant que les réels ont une cardinalité plus grande que les entiers.

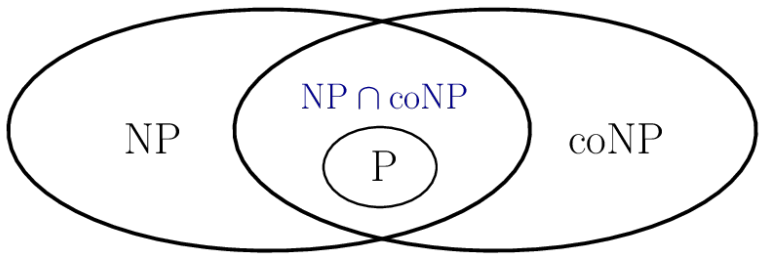
En fait,

- nombre de langages : cardinalité des réels
- nombre de machines : cardinalité des entiers

# Plan de la partie “Calculabilité”

1. Rappels :
   - Automates finis, automates à pile, machines de Turing
   - Non-déterminisme
2. Langages décidables et reconnaissables
   - Autres modèles équivalents aux machines de Turing
   - Propriétés des langages (complément, union, etc.)
   - Exemple de langages décidables / reconnaissables
3. Universalité et indécidabilité
   - Codage des machines de Turing
   - Machine de Turing universelle
   - Exemples de langages indécidables / non-reconnaissables
4. Théorème de Rice (+ divers)

# Partie 2 : Complexité



# Gödel's letter to Von Neumann (1956)

Princeton, 20/III. 1956

Lieber Herr v. Neumann!

Ich habe mit größtem Bedauern von Ihrer Erkrankung gehört. Die Nachricht kam mir ganz unerwartet. Morgenstern hatte mir zwar schon im Sommer von einem Schwächeanfall erzählt, den Sie einmal hatten, aber er meinte damals, dass keine größere Bedeutung beizumessen sei. Wie ich höre, haben Sie sich in den letzten Monaten einer radikalen Behandlung unterzogen und freue mich, dass diese den gewünschten Erfolg hatte. Es ist Ihnen jetzt wieder gut. Ich hoffe und wünsche Ihnen, dass Ihr Zustand sich bald noch weiter bessert und dass die neuesten Errungenschaften der Medizin, wenn möglich, zu einer vollständigen Heilung führen mögen.

Da Sie sich, wie ich höre, jetzt kräftig fühlen, möchte ich mir erlauben, Ihnen über ein mathematisches Problem zu schreiben, über das mich Ihre Ansicht sehr interessieren würde: Man kann offenbar leicht eine Turingmaschine konstruieren, welche von jeder Formel F des engeren Funktionalkalküls und jeder natürl. Zahl n zu entscheiden gestattet, ob F einen Beweis der Länge n hat [Länge = Anzahl der Symbole]. Sei $\psi(F,n)$ die Anzahl der Schritte, die die Maschine dazu benötigt und sei $\varphi(n) = \max_F \psi(F,n)$. Die Frage ist, wie rasch $\varphi(n)$ für eine optimale Maschine wächst. Man kann zeigen $\varphi(n) \geq Kn$. Wenn es wirklich eine Maschine mit ~~$\varphi(n) \sim Kn$~~ $\varphi(n) \sim Kn$ (oder auch nur $\sim Kn^2$) gäbe, hätte das Folgerungen von der größten Tragweite. Es würde nämlich offenbar bedeuten, dass man trotz der Unlösbarkeit des Entscheidungsproblems die Denkarbeit der Mathematiker bei ja-oder-nein Fragen vollständig durch Maschinen ersetzen könnte. ~~(Jetzt gestatten~~

# Gödel's letter to Von Neumann (1956)

I would like to allow myself to write you about a mathematical problem, of which your opinion would very much interest me. One can obviously construct a Turing machine, which for every formula $F$ in first order predicate logic and every natural number $n$, allows one to decide if there is a proof of $F$ of length $n$ (length = number of symbols). Let $\Psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \Psi(F, n)$.

**The question is how fast $\varphi(n)$ grows for an optimal machine**. (...) If there really were a machine with $\varphi(n) \sim n$ (or even $\sim n^2$), this would have consequences of the greatest importance. Namely, it would mean that (...) the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine.

(...) It would be interesting to know, for instance, the situation concerning the determination of primality of a number and how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search.

# Complexité algorithmique ?

**Ressources** nécessaires pour résoudre un problème / décider un langage.

## Quel type de ressources ?

- Temps (nombre d'opérations)
- Espace (quantité de mémoire utilisée)
- Impact du non-déterminisme ?
- Impact de l'aléa ?
- ...

## Point de vue **asymptotique**

- **Évolution** de ces quantités en fonction de la **taille** $n$ de l'entrée, quand $n \to \infty$
- Notations $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ (ignore les facteurs constants et les termes dominés)

  Intuition $\leq \quad \geq \quad =$ Ex : $3n^2 + 5n + 4 = \Theta(n^2)$
- Quelques adjectifs :

| | | | |
|---|---|---|---|
| Constant | $O(1)$ | Quadratique | $O(n^2)$ |
| Logarithmique | $O(\log n)$ | Exponentiel | $O(2^n)$ |
| Linéaire | $O(n)$ | Factoriel | $O(n!)$ |
| Quasi-linéaire | $O(n \log n)$ | Polynomial | $O(n^c) = n^{O(1)}$ |

# L'espace et le temps

## Classes génériques pour les machine de Turing **déterministe**

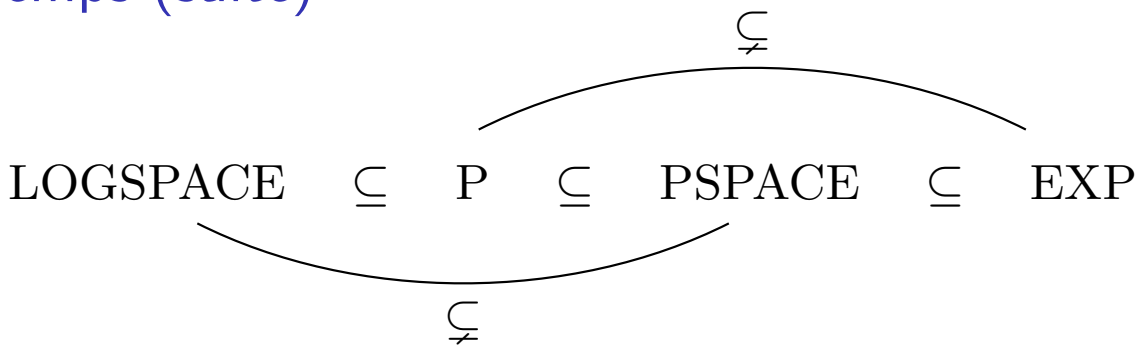
- $\mathrm{TIME}(f(n))$ : Langages que l'on peut décider en temps $O(f(n))$.
- $\mathrm{SPACE}(f(n))$ : Langages que l'on peut décider en espace $O(f(n))$.

## Exemples incontournables

| Définition | En langage courant | Nom commun |
|---|---|---|
| $\mathrm{SPACE}(\log n)$ | Langages décidables en **espace logarithmique** | LOGSPACE |
| $\mathrm{TIME}(n^{O(1)})$ | Langages décidables en **temps polynomial** | P |
| $\mathrm{SPACE}(n^{O(1)})$ | Langages décidables en **espace polynomial** | PSPACE |
| $\mathrm{TIME}(2^n)$ | Langages décidables en **temps exponentiel** | EXP |

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

# L'espace et le temps (suite)




## Théorèmes de hiérarchie (très simplifiés)

Pour deux fonctions $f(n)$ et $g(n)$ abrégées en $f$ et $g$ :

- Hiérarchie en temps :
  $f$ plus petit que $g \implies \mathrm{TIME}(f) \subsetneq \mathrm{TIME}(g)$ (Stearns et Hartmanis'65, puis d'autres)
  $\rightarrow$ "Avec plus de temps, on résoud plus de problèmes"

- Hiérarchie en espace :
  $f$ plus petit que $g \implies \mathrm{SPACE}(f) \subsetneq \mathrm{SPACE}(g)$
  $\rightarrow$ "Avec plus d'espace, on résoud plus de problèmes"

## Espace versus temps ?

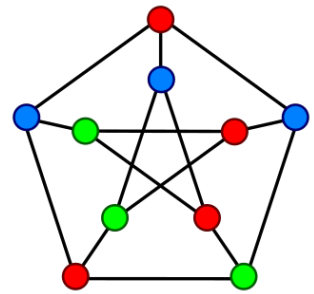
- $\mathrm{TIME}(f) \subseteq \mathrm{SPACE}(f / \log(f))$ (Hopcroft, Paul, Valiant'77)
  $\rightarrow$ L'espace est **strictement** plus fort que le temps !

# Non-déterminisme

P : langages décidables en **temps polynomial** par une machine de Turing **déterministe**

NP : langages décidables en **temps polynomial** par une machine de Turing **non-déterministe**

Exemple : langage $L = \{\text{graphes qui admettent une 3-coloration}\}$.



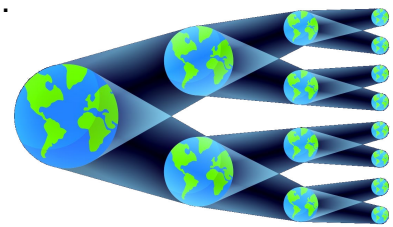
Une machine $M$ décidant le langage $L$ prend en entrée la description d'un graphe (disons $w \in \{0,1\}^*$), et accepte si $w \in L$, sinon rejette.

**Machine déterministe** (algorithme standard).

$\rightarrow$ Tester toutes les colorations. Accepter si on en trouve une (rejeter sinon).

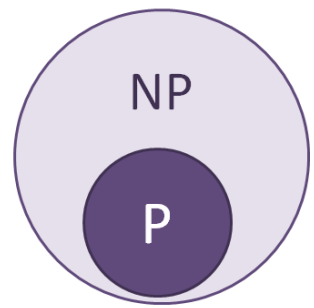
**Machine non-déterministe** (peut tester plusieurs choix de manière non-déterministe)

$\rightarrow$ Chaque branchement explore un choix différent pour le nœud courant.
Quand tous les nœuds sont coloriés, on **vérifie** si la coloration est bonne.
(la machine accepte si et seulement si au moins une branche accepte)



**D'où cet autre point de vue :**

- $L \in P \simeq$ On peut **trouver** la réponse rapidement
- $L \in NP \simeq$ On peut **vérifier** une réponse rapidement




La question du millénaire :

Est-ce que P = NP ?

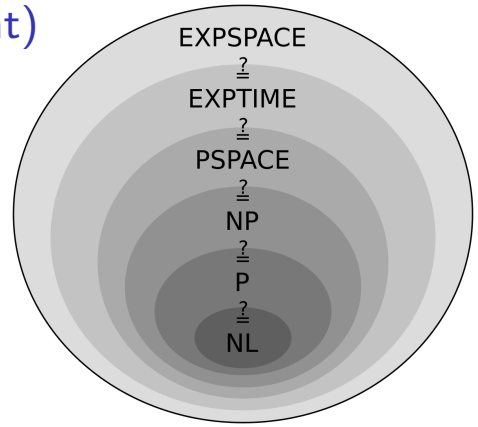
# Non-déterminisme (suite) (on ne verra pas tout)

## Plus généralement

- $\mathrm{NTIME}(f(n))$ : Il existe une preuve vérifiable en **temps** $O(f(n))$ que $w \in L$.
- $\mathrm{NSPACE}(f(n))$ : Il existe une preuve vérifiable en **espace** $O(f(n))$ que $w \in L$.
- $\mathrm{coNTIME}$ et $\mathrm{coNSPACE}$ : idem mais pour vérifier que $w \notin L$.

## Classes non-déterministes incontournables (les mêmes qu'avant)

$$
\begin{array}{lcl}
\mathrm{NL} & := & \mathrm{NSPACE}(\log n) \\
\mathrm{NP} & := & \mathrm{NTIME}(n^{O(1)}) \\
\mathrm{NPSPACE} & := & \mathrm{NSPACE}(n^{O(1)}) \\
\mathrm{NEXP} & := & \mathrm{NTIME}(2^n)
\end{array}
$$




## Relations

- $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{coNP}$ (algo = vérifieur avec certificat vide)
- $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (théorème de Savitch'70)
- $\mathrm{NSPACE}(f(n)) = \mathrm{coNSPACE}(f(n))$ (thérorème d'Immerman–Szelepcsényi'87)
- $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ (énumération des certificats)
- $\mathrm{LOGSPACE} \stackrel{?}{=} \mathrm{NL} \stackrel{?}{=} \mathrm{P} \stackrel{?}{=} \mathrm{NP} \stackrel{?}{=} \mathrm{PSPACE} \stackrel{?}{=} \mathrm{EXP} \stackrel{?}{=} \mathrm{NEXP} \stackrel{?}{=} \mathrm{EXPSPACE}$
- Arguments de remplissage (padding) : $\mathrm{P} = \mathrm{NP} \implies \mathrm{EXP} = \mathrm{NEXP}$ (et d'autres)

# NP-complétude

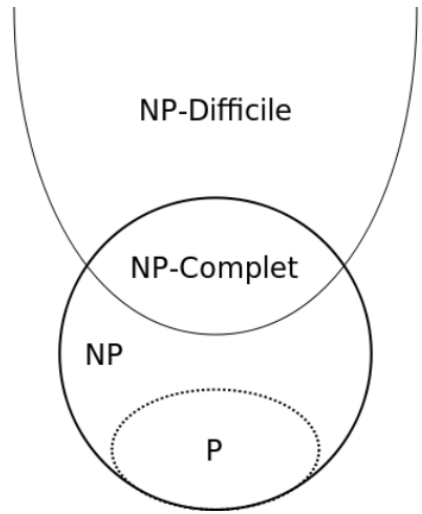
## Réduction

Un langage $L_1$ **se réduit** en *temps polynomial* à un autre langage $L_2$, si pour tout mot $x_1$ de $L_1$, on peut construire en temps polynomial une instance $x_2$ de $L_2$ telle que $x_1 \in L_1 \iff x_2 \in L_2$.

$\rightarrow$ On peut utiliser $L_2$ pour résoudre $L_1$ rapidement : il est "au moins aussi difficile".

## Difficulté et complétude

- NP-difficile : au moins aussi difficile que tout langage dans NP
- NP-complet : à la fois NP-difficile et dans NP




## Exemples

- SAT est NP-complet (Cook, Levin'71)
- 3-Coloration, Clique, Set Cover, Voyageur de commerce,
- ...

Trouvez un algorithme rapide pour l'un de ces problèmes $\implies$ P = NP.

# Plan de la partie “Complexité”

1. Notions de bases
   - Techniques sur les machines de Turing (compression de bandes, accélération linéaire)
   - Complexité en temps, en espace (classes TIME, SPACE, NTIME, NSPACE)
2. Relations entre l'espace et le temps
3. Classes de complexité P, NP, PSPACE, NPSPACE
4. NP-complétude
   - Le problème SAT est NP-complet
   - Quelques problèmes NP-complets
     (3-SAT, Clique, Vertex cover, Chemin hamiltonien, Couplage 3D)
5. Autres problèmes NP-complets
   - Exact cover, Cycle Hamitonien, Partition, Sac à dos, Arbre couvrant borné, Isomorphisme de sous-graphe, Voyageur de commerce
6. Compléments
   - Autre classes de complexité : co-NP, LOGSPACE, IP, AM, BQP (quantique)
   - Théorème de Ladner
   - Comment gagner 1 million de dollars :-)