Complexité Algorithmique

Basé sur le cours de Pascal Koiran
Notes prises par Hugo Salou

ENS DE LYON

27 février 2026

Table des matières

Merci Amaury MAZOYER pour les 2.5 premiers chapitres (même si j'ai corrigé quelques coquilles).

1 Machines de Turing.

On travaille avec des machines à $k \geq 1$ rubans. Les rubans sont semi-infinis à droite et sur chaque ruban, on a une tête de lecture qui lit le contenu d'une case.

À chaque étape,

- la machine M lit les k caractères situés sous les têtes de lecture $(a_1, \ldots, a_k)$;
- en fonction de son état interne $q \in Q$, et des caractères lus, M remplace chaque a_i par a'_i, déplace les têtes de lectures (d'au plus un case à droite ou à gauche), et passe dans un état $q' \in Q$.

1.1 Définitions.

Définition 1.1. Formellement, une *machine de Turing* à k rubans est un triplet (Γ, Q, δ) où

- Γ est l'alphabet du ruban ;
- Q est un ensemble fini d'états ;
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{\leftarrow, \rightarrow, \mathtt{I}\}^k$ la fonction de transition.

Remarque 1.1. On fixe

- un *ruban d'entrée* (généralement supposé en lecture uniquement),
- un *ruban de sortie* (généralement supposé en écriture uniquement),
- un état initial q_{initial} ;
- un état final q_{final} ;

- un symbole blanc $\square \in \Gamma$;
- un symbole de départ $\triangleright \in \Gamma$;
- un alphabet d'entrée $\Sigma \subseteq \Gamma \setminus \{\triangleright, \square\}$. [1]

Au départ, M est dans l'état q_{initial}, le ruban d'entrée contient le mot infini $\triangleright x \square^\infty$ où $x \in \Sigma^\star$ est l'entrée.

Définition 1.2. On dit que M *calcule* la fonction $f : \Sigma^\star \to \Sigma^\star$ si, pour toute entrée $x \in \Sigma^\star$, le calcul de M termine et $f(x)$ est écrit sur le ruban de sortie.

On dit qu'un langage $L \subseteq \Sigma^\star$ est *reconnu* si on peut calculer la fonction $\mathbb{1}_L : \Sigma^\star \to \{\mathtt{0}, \mathtt{1}\} \subseteq \Sigma$. On peut aussi avoir un état q_{accepte} acceptant et un état q_{rejet} de rejet.

Remarque 1.2 (Variantes). On peut considérer un modèle avec des rubans bi-infinis. C'est un modèle équivalent (*c.f.* TD).

On peut considérer une machine avec $\Gamma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \square, \triangleright\}$, c'est-à-dire un alphabet plus gros. Ce modèle est équivalent avec l'association

$$\mathtt{0} \leftrightarrow \mathtt{00} \qquad \mathtt{1} \leftrightarrow \mathtt{01} \qquad \mathtt{2} \leftrightarrow \mathtt{10} \qquad \mathtt{3} \leftrightarrow \mathtt{11}.$$

Définition 1.3. Un langage $L \subseteq \Sigma^\star$ est *reconnu en temps* $T(n)$ par une machine M si

- M reconnaît L ;
- sur toute entrée x de taille n, la machine M s'arrête en au plus $T(n)$ étapes de calcul.

Définition 1.4. On dit que L est dans la classe $\mathrm{DTIME}(f(n))$ s'il existe une machine (à plusieurs rubans) qui reconnaît L en temps $\mathrm{O}(f(n))$. On supposera toujours avoir $f(n) \geq n + 1$. [2]

1. Généralement, on prendra $\Sigma \subseteq \{\mathtt{0}, \mathtt{1}\}$.
2. Il faut, au moins, lire l'entrée.

Définition 1.5. La classe P est l'ensemble des langages reconnaissables en temps polynomial :

$$\mathsf{P} = \bigcup_{\alpha \geq 1} \mathrm{DTIME}(n^\alpha).$$

Théorème 1.1. Si $L \in \mathrm{DTIME}(f(n))$ alors L peut être reconnu en temps $\mathrm{O}(f(n)^2)$ sur une machine à un ruban.

Théorème 1.2 (Simulation efficace). Pour toute machine M fonctionnant en temps $T(n)$, il existe une machine M' à deux rubans qui fonctionne en temps $\mathrm{O}(T(n) \log T(n))$ telle que $M(x) = M'(x)$ pour toute entrée $x \in \Sigma^\star$.

Définition 1.6. Étant donné x, y, on définit l'encodage du couple (x, y) comme le mot $\langle x, y \rangle = \mathtt{1}^{|x|}\mathtt{0}xy$.

Définition 1.7. Une *machine universelle* U prend en entrée des coules $\langle x, \alpha \rangle$ (où α est le code d'une machine M_α et $x \in \Sigma^\star$) et simule la machine M_α sur l'entrée x. Autrement dit, pour tout x et tout α, on a

$$U(\langle x, \alpha \rangle) = M_\alpha(x).$$

Théorème 1.3. Il existe une machine de Turing universelle U telle que, sur tout entrée $\langle x, \alpha \rangle$, si M_α s'arrête sur x en t étapes, alors la machine U s'arrête sur $\langle x, \alpha \rangle$ en au plus $c\, t \log t$, où c ne dépend que de x.

Preuve.

- ▷ *Cas d'une machine à deux rubans.* On montre d'abord que si M_α est une machine à deux rubans, alors U peut simuler

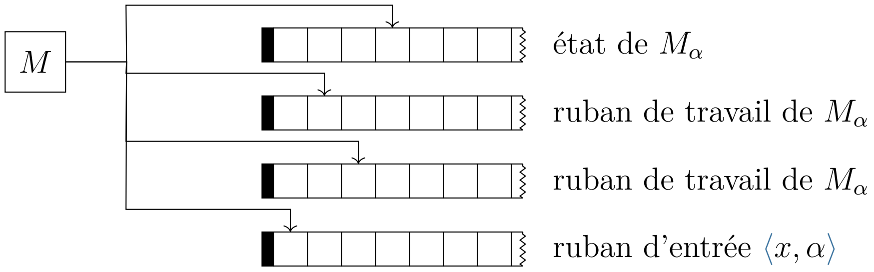

Figure 1.1 | *Construction de U dans le cas à deux rubans*

M_α en temps linéaire. En effet, on utilise quatre rubans (figure 1.1) :

- deux rubans pour stocker les deux rubans de M_α ;
- un ruban qui stocke l'état de M_α ;
- le ruban d'entrée qui contient $\langle x, \alpha \rangle$.

Pour faire une étape de calcul de M_α, la machine U doit déterminer $\delta(q, a, b)$, mais la complexité de cette opération est cachée dans la constante.

▷ *Cas général.* Par le théorème de simulation efficace, on peut construire une machine M_β à deux rubans équivalente et on peut simuler M_β en temps linéaire, et on obtient bien la complexité attendue.

□

1.2 Non déterminisme.

Définition 1.8. Une *machine de Turing non-déterministe* est un triplet (Γ, Q, δ) où

$$\delta : Q \times \Gamma^k \to \wp(Q \times \Gamma^k \times \{\leftarrow, \rightarrow, \mathtt{I}\}^k),$$

où l'on distingue deux états q_{accepte} et q_{rejet}.

Une entrée $x \in \Sigma^\star$ est *acceptée* s'il existe un chemin d'exécution acceptant sur l'entrée x.

Définition 1.9. On note $\mathrm{NTIME}(f(n))$ l'ensemble des langages acceptés par une machine de Turing non-déterministe fonctionnant en temps $\mathrm{O}(f(n))$ sur toute entrée de taille n et tout chemin de calcul.

Définition 1.10. Un langage L est dans NP s'il existe une machine non-déterministe M fonctionnant en temps polynomial tel que L est l'ensemble des entrées acceptées par M. Ainsi,

$$\mathsf{NP} = \bigcup_{\alpha \geq 1} \mathrm{NTIME}(n^\alpha).$$

Théorème 1.4 (Définition alternative de NP)**.** Un langage L est dans NP s'il existe un polynôme p et $A \in \mathsf{P}$ tel que, pour toute entrée $x \in \{\mathtt{0}, \mathtt{1}\}^\star$,

$$x \in L \qquad \iff \qquad \exists y \in \{\mathtt{0}, \mathtt{1}\}^\star, \quad \langle x, y \rangle \in A.$$

On dit que y *certifie* que x est dans L.

1.3 NP-complétude.

Définition 1.11. On dit qu'un problème A *se réduit à* B *en temps polynomial* s'il existe une fonction $f : \Sigma^\star \to \Sigma^\star$ calculable en temps polynomial telle que, pour toute entrée x,

$$x \in A \qquad \iff \qquad f(x) \in B.$$

On note alors $A \leq_{\mathrm{P}} B$ ou $A \leq_{\mathrm{M}} B$.[3]

Remarque 1.3. Si $A \leq_{\mathrm{P}} B$ et $B \leq_{\mathrm{P}} C$ alors $A \leq_{\mathrm{P}} C$ par composition des réductions.

3. Personnellement, je noterai parfois $f : A \leq_{\mathrm{P}} B$ où f est une fonction de réduction.

Définition 1.12 (NP-complétude)**.** On dit que A est NP-*complet* dès lors que

- $A \in \mathsf{NP}$;
- A est NP-*dur*, c'est-à-dire $B \leq_{\mathrm{P}} A$ pour tout $B \in \mathsf{NP}$.

Remarque 1.4. Pour montrer qu'un problème $A \in \mathsf{NP}$ est NP-complet, il suffit de montrer que $B \leq_{\mathrm{P}} A$ où B est un problème NP-complet. En effet, on a alors, pour tout $C \in \mathsf{NP}$,

$$C \leq_{\mathrm{P}} B \leq_{\mathrm{P}} A.$$

Il suffit donc d'avoir un problème NP-complet comme « point de départ ». Généralement, on choisit Sat ou 3-Sat, mais dans ce cours on partira de CircuitSat (défini au prochain chapitre).

Sat	**Entrée.** Une formule booléenne ϕ **Sortie.** La formule ϕ est-elle satisfiable ?

3-Sat	**Entrée.** Une formule booléenne ϕ sous 3-CNF **Sortie.** La formule ϕ est-elle satisfiable ?

2 Circuits booléens.

2.1 Définitions.

Définition 2.1. Un *circuit booléen* est un DAG [1] dont les sommets sont étiquetés et de degré entrant 0, 1 ou 2 :

- les sommets de degré entrant 2 sont étiquetés par $\land$ ou $\lor$;
- les sommets de degré entrant 1 sont étiquetés par $\neg$;
- les sommets de degré 0 sont étiquetés par 0, 1 ou des variables booléennes $x_1, \ldots, x_n$.

Les sommets sont appelés *portes*, et les sommets de degré 0 sont des *portes d'entrées*.

Définition 2.2. Soit C un circuit booléen avec des variables d'entrée $x_1, \ldots, x_n$. Étant donné une *valuation* $a \in \{\texttt{0}, \texttt{1}\}^n$, pour chaque porte α de C, on définit la *valeur* prise par α sur l'entrée a par :

- pour les portes d'entrées, on pose
$$\operatorname{val}(x_i) := a_i, \quad \operatorname{val}(0) := \texttt{0}, \quad \operatorname{val}(1) := \texttt{1} \;;$$
- pour les autres portes, on pose
  - $\operatorname{val}(\alpha \lor \beta) = \operatorname{val}(\alpha) \;\texttt{||}\; \operatorname{val}(\beta)$,
  - $\operatorname{val}(\alpha \land \beta) = \operatorname{val}(\alpha) \;\texttt{\&\&}\; \operatorname{val}(\beta)$,
  - $\operatorname{val}(\neg\alpha) = \texttt{not}\; \operatorname{val}(\alpha)$.

1. *Directed Acyclic Graph*, un graphe orienté acyclique

Définition 2.3. Si C n'a qu'une seule porte α de degré sortant 0, on dit une α est *porte de sortie* de C, et on pose $\operatorname{val}(C) := \operatorname{val}(\alpha)$.

On peut donc dire que C calcule une fonction $f : \{\mathtt{0}, \mathtt{1}\}^n \to \{\mathtt{0}, \mathtt{1}\}$ lorsque $f(a)$ est la valeur prise par C sur l'entrée a.

Plus généralement, si C a s portes de sorties, le circuit C peut calculer une fonction $f : \{\mathtt{0}, \mathtt{1}\}^n \to \{\mathtt{0}, \mathtt{1}\}^s$.

Remarque 2.1. Toute fonction booléenne peut être calculée par un circuit : il suffit de considérer l'arbre de syntaxe de celle-ci.

Exercice 2.1. Donner une famille de fonctions booléennes « explicites » $(f_n)_{n\geq 1}$ qui ne sont pas calculables par des circuits de taille polynomiale.

Lemme 2.1. On a ValCirc $\in$ P où

ValCirc	**Entrée.** Une circuit booléen C et $a \in \{\mathtt{0}, \mathtt{1}\}^n$ **Sortie.** Est-ce que $\operatorname{val}(C) = \mathtt{1}$ sous l'entrée a ?

Preuve. On utilise l'algorithme suivant :

1 : Calculer un tri topologique de C (pour la boucle ci-dessous).
2 : **pour** toute porte α de C **faire**
3 : Évaluer α à l'aide des valeurs déjà calculées[2]
4 : **retourner** la valeur de la porte de sortie

□

2.2 Simulation de machines de Turing par les circuits.

2. Par l'ordre topologique, on sait que les entrées de α ont déjà été évaluées.

Proposition 2.1. Soit M une machine à un ruban fonctionnant en temps $T(n)$. On peut simuler M sur les entrées de taille n par un circuit de taille $\mathrm{O}(T(n)^2)$.

Remarque 2.2. On peut donner la borne de $\mathrm{O}(T(n)\log T(n))$ au lieu de $\mathrm{O}(T(n)^2)$, même pour des machines à plusieurs rubans.

Définition 2.4. Un *diagramme espace-temps* est un diagramme 2D représentant l'état d'un ruban de M au cours du temps (figure 2.1)

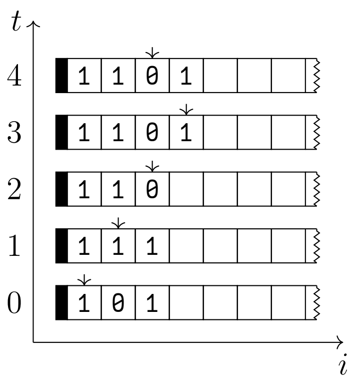

Figure 2.1 | *Exemple de diagramme espace-temps*

Preuve (Simulation par circuit)**.** Considérons un diagramme de taille $p \times p$, où $p = T(n) + 1$. Par la localité du calcul, le contenu de la cellule (i, t) ne dépend que de trois cellules sur le ruban du dessous : celle directement en dessous, celle en dessous et à gauche, et celle en dessous et à droite.

Définissons une variable $\ell_{a,i,t}$ qui dit « à l'instant t, la case i contient la lettre a » pour tout a, i, t. Définissions aussi $q_{r,i,t}$ indiquant « à l'instant t la machine est dans l'état r et la tête de lecture est sur la case i ».

Comme indiqué, on peut déterminer la valeur de ces variables à partir des valeurs de ces variables avec $(i-1, t-1)$, $(i, t-1)$ et $(i+1, t-1)$. Ceci nous permet d'en déduire une fonction booléenne

$$f : \{\texttt{0}, \texttt{1}\}^{3p} \to \{\texttt{0}, \texttt{1}\}^p.$$

Cette fonction ne dépendant que des transitions de M, on en déduit un circuit C calculant f.

En réalisant $\mathrm{O}(T(n)^2)$ copies de C, on obtient le circuit final. L'entrée est acceptée si $\bigvee_{i=0}^{T(n)} q_{q_\text{accepte},i,T(n)}$ a pour valeur $\texttt{1}$. □

Définition 2.5 (Famille de circuits uniforme)**.** Soit $(C_n)_{n\geq 1}$ une famille de circuits sur n variables. Cette famille est *uniforme* s'il existe une machine de Turing qui, sur l'entrée $\texttt{1}^n$, calcule une description complète de C_n en temps polynomial. C'est-à-dire, pour chaque porte α de C_n, elle calcule

- ▷ le type de porte ;
- ▷ si c'est une porte d'entrée, son étiquette ;
- ▷ si ce n'est pas une porte d'entrée, les numéros des portes en entrée de α.

Remarque 2.3. Si $(C_n)_{n\geq 1}$ est une famille uniforme, alors C_n est de taille polynomiale en n car sa description complète est écrite en temps polynomial.

Théorème 2.1. Un langage $L \subseteq \{\texttt{0}, \texttt{1}\}^\star$ est dans P si et seulement si L est reconnu par une famille uniforme de circuits booléens de taille polynomiale.

Preuve.

- ▷ « $\implies$ ». Soit $L \in \mathsf{P}$. Dans la preuve précédente, on construit une famille de circuits taille polynomiale. Elle est uniforme car il suffit d'itérer sur i et sur t, pour calculer les valeurs de $\ell_{a,i,t}$ et $q_{r,i,t}$. Ceci se fait en temps polynomial.

- « $\Longleftarrow$ ». Pour tester si $x \in \{0,1\}^n$ est dans L, on construit C_n (en temps polynomial), puis on évalue C_n sur x (qui se fait en temps polynomial).

$\square$

2.3 Premiers problèmes NP-complets.

Théorème 2.2. Le problème CircuitSat est NP-complet où

CircuitSat	**Entrée.** Une circuit booléen C avec n variables **Sortie.** Existe-t-il a tel que $C(a) = 1$?

Preuve.

- D'une part, CircuitSat $\in$ NP avec a le certificat et ValCirc le problème de vérification.
- D'autre part, soit $L \in$ NP. Il existe une machine de Turing non-déterministe M fonctionnant en temps polynômial et qui reconnaît L. Soit $x \in \{0,1\}^n$. On doit construire en temps polynomial un circuit C_x qui est satisfiable ssi $x \in L$.

  On peut simuler M sur l'entrée x par un circuit de taille $\mathrm{O}(T(n)^2)$... mais ce n'est pas suffisant : il faut modéliser le non-déterminisme.

  On ajoute des variables d'entrée supplémentaires $y_1, \ldots, y_{T(n)}$ qui modélisent les choix non-déterministes de la machine M.

  On en déduit C_x (construit en temps polynomial) qui simule M sur l'entrée x avec les $(y_i)_i$ pour les choix non-déterministes.

  On a que C_x est satisfiable si et seulement s'il existe une suite de choix non-déterministes (les valeurs des y_i) telle que M accepte l'entrée x, c'est-à-dire ssi $x \in L$.

$\square$

Théorème 2.3 (Cook-Levin)**.** Le problème 3-Sat est NP-complet où

3-Sat	**Entrée.** Une formule booléenne ϕ sous 3-CNF **Sortie.** La formule ϕ est-elle satisfiable ?

Preuve.

- On a que 3-Sat $\in$ NP car on peut utiliser la valuation comme certificat.
- On fait une réduction de CircuitSat à 3-Sat. Soit C un circuit booléen avec des variables $x_1, \dots, x_n$. Pour chaque porte α, on crée une variable z_α qui représente la valeur prise par le porte α. On utilise l'identité $(P \Rightarrow Q) \Leftrightarrow \bar{P} \vee Q$ pour construire les clauses qui contraignent les variables z_α à respecter le fonctionnement des portes. Par exemple :
  - si $\alpha = \beta \wedge \gamma$, on ajoute les clauses
  $$\underbrace{(\bar{z}_\beta \vee \bar{z}_\gamma \vee z_\alpha)}_{\beta \wedge \gamma \Rightarrow \alpha}, \quad \underbrace{(z_\beta \vee \bar{z}_\alpha)}_{\alpha \Rightarrow \beta} \quad \text{et} \quad \underbrace{(z_\gamma \vee \bar{z}_\alpha)}_{\gamma \Rightarrow \alpha} ;$$
  - si $\alpha = \beta \vee \gamma$, on ajoute les clauses
  $$\underbrace{(z_\beta \vee z_\gamma \vee \bar{z}_\alpha)}_{\alpha \Rightarrow \beta \vee \gamma}, \quad \underbrace{(\bar{z}_\beta \vee z_\alpha)}_{\beta \Rightarrow \alpha} \quad \text{et} \quad \underbrace{(\bar{z}_\gamma \vee z_\alpha)}_{\gamma \Rightarrow \alpha} ;$$
  - si $\alpha = \neg\beta$, on ajoute les clauses
  $$\underbrace{(\bar{z}_\beta \vee \bar{z}_\alpha)}_{\alpha \Rightarrow \bar{\beta}} \quad \text{et} \quad \underbrace{(z_\beta \vee z_\alpha)}_{\bar{\alpha} \Rightarrow \beta} .$$

  Enfin, on ajoute la clause $z_{\alpha_{\text{sortie}}}$ pour forcer la porte de sortie à être vraie.

$\square$

3 Complexité en espace.

3.1 Définitions et premières propriétés.

Définition 3.1. L'*espace utilisé* par une machine de Turing déterministe sur une entrée x est le nombre de cases distinctes utilisés sur les rubans de travail [1] au cours de son calcul sur x.

On dit que M fonctionne en espace $s(n)$ si M s'arrête sur toutes ses entrées et utilise un espace au plus $s(n)$ sur toute entrée de taille n.

On note $\mathrm{DSPACE}(s(n))$ l'ensemble des langages reconnus par une machine de Turing déterministe fonctionnant en espace $\mathrm{O}(s(n))$.

Remarque 3.1. On supposera que, pour le ruban d'entrée, la tête de lecture ne dépassera jamais la fin de l'entrée.

Exemple 3.1.

- ▷ Un algorithme naïf pour Sat utilise un espace en $\mathrm{O}(n)$. En effet, il suffit d'énumérer toutes les valuations possibles avec un compteur binaire de taille n, puis de vérifier si une de ces valuations satisfait la formule.
- ▷ L'addition de deux entiers de taille n peut s'effectuer en espace $\mathrm{O}(\log n)$. En effet, il suffit de stocker les positions des bits en cours d'addition et la retenue.

---

1. Pas le ruban d'entrée, ni le ruban de sortie, ceci permet de parler de machine utilisant un espace logarithmique, bien que la taille de l'entrée soit n.

Définition 3.2. On dit que $t : \mathbb{N} \to \mathbb{N}$ est *constructible en espace* s'il existe une machine de Turing qui, sur l'entrée $\mathtt{1}^n$ calcule $\mathtt{1}^{t(n)}$ en espace $\mathrm{O}(t(n))$.

Proposition 3.1. On a l'inclusion

$$\mathrm{NTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n)).$$

Preuve. Supposons f constructible en espace $(*)$.

Soit $L \in \mathrm{NTIME}(f(n))$ et M une machine non-déterministe reconnaissant L en temps au plus $c\, f(n)$. On code un chemin de calcul de M par $y \in [\![0, R-1]\!]^{c\, f(n)}$ où R désigne le nombre de choix possibles à chaque étape (il dépend de M).

1 : Calculer $f(n)$ en espace $\mathrm{O}(f(n))$
2 : **pour tout** y de taille $c\, f(n)$ **faire**
3 :   Simuler M sur x en suivant les choix donnés par y
4 :   **si** la simulation accepte **alors**
5 :   **Accepter**
6 : **Rejeter**

On a un algorithme en espace $f(n)$ qui teste l'appartenance au langage L.

Sans l'hypothèse $(*)$, on peut obtenir la même inclusion avec une légère modification de l'argument. On fait fonctionner le même algorithme pour des chemins de calcul de longueur $t = 1, 2, \ldots$ jusqu'à ce que la simulation de M sur l'entrée de x s'arrête (ce qui arrive forcément si $x \in L$). On s'arrête pour $t = c\, f(n)$ au plus. $\square$

Proposition 3.2. On a l'inclusion

$$\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathrm{O}(s(n))}),$$

dès lors que $s(n) \geq \log n$.

Preuve. Soit N tel que, si un calcul prend un temps plus long que N, alors il boucle. Nous allons montrer que $N = 2^{\mathrm{O}(s(n))}$. On compte le nombre de configurations distinctes possibles d'une machine M sur l'entrée x de taille n. Une configuration est donnée[2] par :

- l'état courant (il y a au plus $|Q|$ possibilités) ;
- la position de la tête de lecture sur le ruban d'entrée (il y a au plus n possibilités) ;
- le contenu des cases utilisées sur les rubans de travail (il y a au plus $|\Gamma|^{s(n)}$ possibilités) ;
- la position des têtes de lecture sur les rubans de travail (il y a au plus $s(n)^k$ possibilités où k est le nombre de rubans de travail).

D'où la borne annoncée. □

Remarque 3.2. A-t-on $\mathrm{DSPACE}(1) \subseteq \mathrm{DTIME}(1)$? Non ! En effet, retourner le dernier caractère de l'entrée se fait en espace $\mathrm{O}(1)$ mais pas en temps $\mathrm{O}(1)$.

Définition 3.3. On définit $\mathsf{L} = \mathrm{DSPACE}(\log n)$.

Corollaire 3.1. On a les inclusions

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE},$$

où PSPACE est l'ensemble des problèmes définis en espace polynomial (défini plus tard). □

2. On oublie la position de la tête de lecture sur le ruban de sortie, vu qu'elle n'influe pas le calcul (car écriture uniquement).

Théorème 3.1. Si deux fonctions $f, g : \{\mathtt{0}, \mathtt{1}\}^\star \to \{\mathtt{0}, \mathtt{1}\}^\star$ sont calculables en espace $s(n) \geq \log n$ alors leur composée $g \circ f$ est calculable en espace $\mathrm{O}(s(|x|)) + s(|f(x)|)$.

Preuve. L'idée est de calculer un bit de $f(x)$ uniquement quand on en a besoin pour calculer $g(f(x))$.

> **Lemme 3.1.** Étant donné i, on peut calculer le i-ème bit de $f(x)$ en espace $\mathrm{O}(s(|x|))$.
>
> **Preuve.** Faire fonctionner M_f (la machine calculant f en espace logarithmique) sans écrire sur le ruban de sortie, mais simplement en comptant le nombre de bits que l'on voulait écrire. Dès lors que i est atteint, on renvoie le bit courant : c'est le i-ème bit de $f(x)$. □

On utilise ensuite l'algorithme suivant :

$i \leftarrow 0$
tant que M_g ne s'est pas arrêtée **faire**
    Calculer le i-ème bit de $f(x)$ (par le lemme)
    L'écrire sur le ruban d'entrée de M_g
    Effectuer un pas de calcul de M_g
    si la tête $\rightarrow$ sur le ruban d'entrée de M_g **alors**
        $i \leftarrow i + 1$
    sinon si la tête $\leftarrow$ sur le ruban d'entrée de M_g **alors**
        $i \leftarrow i - 1$

□

Corollaire 3.2. Si f et g sont calculables en espace $\mathrm{O}(\log n)$ alors $g \circ f$ est calculable en espace $\mathrm{O}(\log n)$. □

3.2 Hiérarchie en espace.

Théorème 3.2. Si s est constructible en espace et $s(n) \geq \log n$ alors il existe un langage reconnaissable en espace $\mathrm{O}(s(n))$ mais pas en espace $\mathrm{o}(s(n))$.

Preuve. L'idée est de faire une preuve par diagonalisation. On construit une machine D qui fonctionne en espace $\mathrm{O}(s(n))$ et qui reconnaît un langage A différent de tous les langages reconnus en espace $\mathrm{o}(s(n))$. Pour cela, D simule une machine tournant en espace au plus $\mathrm{o}(s(n))$ et on a que $D(\langle M\rangle)$ accepte ssi $M(\langle M\rangle)$ rejette (et inversement).

Lemme 3.2. Si $L \in \mathrm{DSPACE}(s(n))$ alors L est reconnaissable en $\mathrm{O}(s(n))$ par une machine à un ruban de travail dont l'alphabet est $\{\mathtt{0}, \mathtt{1}, \square, \triangleright\}$. $\square$

Lemme 3.3. Il existe une machine de Turing universelle U qui prend en entrée $\langle M, x\rangle$ avec $x \in \{\mathtt{0}, \mathtt{1}\}^\star$ et M une machine à un ruban de travail et dont l'alphabet est $\{\mathtt{0}, \mathtt{1}, \square, \triangleright\}$, et qui simule M sur l'entrée x en espace $\mathrm{O}(s(n))$ si M fonctionne en espace $s(n)$. On peut même supposer que U ne possède qu'un seul ruban de travail. $\square$

À l'aide de ces deux lemmes, on donne l'algorithme suivant (machine D).

Lire l'entrée $w \in \{\mathtt{0}, \mathtt{1}\}^\star$ (notons n sa taille)
Calculer $s(n)$ en espace $\mathrm{O}(s(n))$
Réserver $s(n)$ cases sur le ruban de travail
si w n'est pas de la forme $\langle M\rangle\mathtt{1}\mathtt{0}^\star$ [3] **alors**
  Rejeter
Simuler M sur l'entrée w
si on dépasse $2^{2\cdot s(n)}$ étapes ou $s(n)$ cases mémoires **alors**
  Rejeter
si M accepte **alors Rejeter**
sinon Accepter

On a que la machine D s'arrête sur toute entrée et fonctionne en espace $\mathrm{O}(s(n))$.

De plus, si B est un langage décidable en $\mathrm{o}(s(n))$ par une machine M, alors B est différent du langage de A. En effet, D simule M en espace $\mathrm{o}(s(n))$.

- ▷ Ainsi, il existe n_0 tel que, pour $n \geq n_0$, D fera une simulation en espace strictement plus petit que s(n), donc ne manquera pas d'espace.
- ▷ Aussi, comme M fonctionne en temps $2^{\mathrm{o}(s(n))}$ alors la simulation sera en temps strictement plus petit que $2^{s(n)}$ pour $n \geq n_1$ pour un certain n_1, donc ne manquera pas de temps.

En posant $n_2 := \max(n_0, n_1)$, on a que sur l'entrée $\langle M \rangle \mathtt{1}\mathtt{0}^{n_2}$, la simulation de M se fera jusqu'au bout et D donne une réponse différente de M sur la même entrée. D'où les deux langages B et A sont différents.

□

Corollaire 3.3. On a les inclusions strictes suivantes :

$$\mathsf{L} \subsetneq \mathrm{DSPACE}(n) \subsetneq \mathrm{DSPACE}(n^2) \subsetneq \cdots \subsetneq \mathsf{PSPACE}.$$

3.3 Complexité en espace non-déterministe.

Définition 3.4. Une machine non-déterministe M fonctionne en espace au plus $s(n)$ si, sur toute entrée de taille n, chaque chemin de calcul utilise un espace au plus $s(n)$ (sur les rubans de travail).

On note $\mathrm{NSPACE}(s(n))$ l'ensemble des langages reconnus par une machine de Turing non-déterministe en espace $\mathrm{O}(s(n))$.

Définition 3.5. On définit $\mathsf{NL} := \mathrm{NSPACE}(\log n)$.

3. Pour une certaine machine M

On a prouvé précédemment que $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{s(n)})$ (lorsque l'on a que $s(n) \geq \log n$). On peut améliorer cette inclusion avec $\mathrm{NSPACE}(s(n))$, comme $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{NSPACE}(s(n))$.

Proposition 3.3. On a

$$\mathrm{NSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathrm{O}(s(n))}),$$

pour $s(n) \geq \log n$.

Preuve. On considère G_x le ***graphe*** (orienté) ***des configurations*** de M sur l'entrée x de taille n, où $c_1 c_2 \in \mathrm{E}(G_x)$ ssi M peut passer de la configuration c_1 à la configuration c_2 en un seul pas de calcul.

Lemme 3.4. Le graphe G_x a $2^{\mathrm{O}(s(n))}$ sommets, et on peut le construire en espace $\mathrm{O}(s(n))$. □

On explore G_x à partir de la configuration initiale c_{initiale} et on accepte si on atteint une configuration d'acceptation. □

Remarque 3.3. On n'a pas utilisé l'hypothèse que M s'arrête sur toutes ses entrées.

3.4 Formules booléennes quantifiés, PSPACE.

Définition 3.6. On définit

$$\mathrm{PSPACE} := \bigcup_{k \geq 1} \mathrm{DSPACE}(n^k).$$

Théorème 3.3 (Savitch)**.** On a $\mathrm{NSPACE}(s(n)) \subseteq \mathrm{DSPACE}(s(n)^2)$ si $s(n) \geq \log n$. [4]

Remarque 3.4. On peut aussi définir la classe NPSPACE mais cette classe est égale à PSPACE par Savitch. On a donc

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \underset{(*)}{\subseteq} \mathsf{PSPACE} \underset{(**)}{\subseteq} \overbrace{\mathsf{EXPTIME}}^{\bigcup_{k\geq 1} \mathrm{DTIME}(2^{n^k})},$$

où

- ▷ $(*)$ est vraie car $\mathrm{NTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$;
- ▷ $(**)$ est vraie car $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathrm{O}(s(n))})$.

On sait, de plus, que $\mathsf{P} \subsetneq \mathsf{EXPTIME}$ par la hiérarchie en temps (variante (vue en TD) de la hiérarchie en espace vue avant). Et, que $\mathsf{L} \subsetneq \mathsf{PSPACE}$ par le théorème de Savitch et la hiérarchie en espace.

On autorise des quantificateurs universels et existentiels aux formules vues précédemment.

Exemple 3.2. Les formules

- ▷ $\forall x\, \exists y\, \big(\overbrace{(x \vee y) \wedge (\bar{x} \vee \bar{y})}^{\text{c'est } x \operatorname{xor} y}\big)$
- ▷ $\exists y\, \forall x\, \big((x \vee y) \wedge (\bar{x} \vee \bar{y})\big)$

sont des formules booléennes quantifiées. La première est vraie (avec $y = \bar{x}$) mais pas la seconde (avec $x = y$). On suppose que les quantificateurs sont tous en début de formule (forme prénexe).

Définition 3.7. On définit le problème QBF comme

QBF | **Entrée.** Une formule booléenne quantifiée F (close)
Sortie. Est-ce que F est vraie ?

4. Depuis sa preuve, on n'a jamais réussi à améliorer ce résultat, ni montrer qu'un facteur carré est nécessaire.

Proposition 3.4. On a QBF $\in$ PSPACE.

Preuve. On utilise l'algorithme suivant.

1. Si F ne contient pas de quantificateurs, accepter si F s'évalue à vrai et rejeter si F s'évalue à faux.
2. Si $F = \exists x\ G$, on décide récursivement avec $G[x := 0]$ et $G[x := 1]$. Si une de ces formules s'évalue à vrai, accepter sinon rejeter.
3. Si $F = \forall x\ G$, on décide récursivement avec $G[x := 0]$ et $G[x := 1]$. Si une de ces formules s'évalue à faux, rejeter sinon accepter.

Cet algorithme utilise un espace linéaire. □

Théorème 3.4. Le problème QBF est PSPACE-complet.

Preuve. Supposons que A est résolu en espace n^k par une machine M à un ruban. On va montrer que $A \leq_{\mathrm{P}}$ QBF.

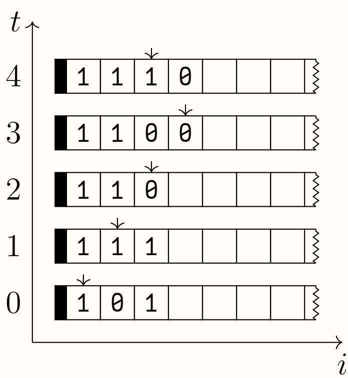

Figure 3.1 | *Exemple de diagramme espace-temps*

On considère le diagramme espace-temps de taille n^k en espace et $2^{c \cdot n^k}$ en temps. On ne peut pas simplement utiliser la même preuve que pour SAT car le diagramme est exponentiel.

On construit une formule $\phi_t(c_1, c_2)$ qui exprime qu'on peut aller de la configuration c_1 en la configuration c_2 en au plus 2^t étapes de calcul.

On a donc que M accepte si $\phi_{c\cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$.[5]

La formule $\phi_0(c_1, c_2)$ est de taille polynomiale (*c.f.* preuve du théorème de Cook).

Pour aller de ϕ_t à ϕ_{t+1}, on cherche la configuration du milieu. On pourrait utiliser $(*) := \exists c\, \phi_t(c_1, c) \wedge \phi_t(c, c_2)$ qui est correcte mais trop couteuse (taille exponentielle). On choisit plutôt :

$$\exists c\, \forall c_3\, \forall c_4\, \big[(c_3 = c_1 \wedge c_4 = c) \vee (c_3 = c \wedge c_4 = c_2)\big] \Rightarrow \phi_t(c_3, c_4).$$

Cette formule est logiquement équivalente à $(*)$ (en regardant les cas où $(c_3, c_4) = (c_1, c)$ et $(c_3, c_4) = (c, c_2)$). Il est important de se rappeler que que c, c_3, c_4 ne sont pas des variables mais des n-uplets de taille $\mathrm{O}(n^k)$ variables booléennes. La taille de ϕ_{t+1} augmente de $\mathrm{O}(n^k)$ par rapport à la taille de ϕ_t. Au final, on est de taille $\mathrm{O}(n^{2k})$.[6] □

Le problème QBF est « le » problème PSPACE-complet. En TD, on fera des réductions de certains problèmes à QBF.

3.5 Problème PATH et théorème de Savitch.

Corollaire 3.4. On a $\mathsf{NL} \subseteq \mathrm{DSPACE}(\log^2 n)$.

Proposition 3.5. On a que PATH $\in \mathrm{DSPACE}(\log^2 n)$, où

5. On peut considérer qu'il y a une unique configuration acceptante, quitte à transformer tout état acceptant en un état qui efface tout le ruban et remet la tête en position initiale.

6. Le passage quadratique est similaire au théorème de Savitch.

PATH	**Entrée.** Un graphe G orienté, et $s, t \in \mathrm{V}(G)$ **Sortie.** Existe-t-il un chemin de s à t dans G ?

Remarque 3.5. En TD, on a vu que PATH est NL-complet, et donc la proposition implique le corollaire. En effet, soit $A \in \mathsf{NL}$ et $x \in \{\mathtt{0}, \mathtt{1}\}^n$, on a

$$x \in A \iff f(x) \in \textsc{Path},$$

où $f : A \le_{\mathrm{L}} \textsc{Path}$ est la réduction en espace logarithmique (car le problème PATH est NL-complet). La construction de $f(x)$ se fait en espace $\mathrm{O}(\log n)$ et décider si $f(x) \in \textsc{Path}$ ou non peut se faire en espace $\mathrm{O}(\log^2 |f(x)|)$, or $|f(x)|$ est polynomial en $|x| = n$, d'où la borne annoncée.

Preuve (de la proposition)**.** On donne un algorithme $\mathsf{path}(G, u, v, i)$ en espace $\mathrm{O}(\log^2 n)$ qui décide s'il existe, dans G, un chemin de u à v de longueur au plus 2^i. On pourra donc résoudre PATH en appelant $\mathsf{path}(G, s, t, \lceil \log n \rceil)$.

```
Procédure path(G, u, v, i)
    si i = 0 alors
        si uv ∈ E(G) ou u = v alors Accepter
        sinon Rejeter
    pour tout sommet w ∈ V(G) faire
        si path(G, u, w, i − 1) et path(G, w, v, i − 1) alors
            Accepter
    Rejeter
```

Pour la correction, on montre si $\mathsf{path}(G, u, v, i)$ accepte alors il existe un chemin de longueur au plus 2^i par récurrence sur i.

- Pour le cas $i = 0$, c'est bon par le premier « **si** ».
- Pour l'hérédité, si on a un chemin de longueur au plus 2^{i-1} de u à w et un chemin de longueur au plus 2^{i-1} de w à v, on concatène ces chemins pour obtenir un chemin de u à v

de longueur au plus 2^i.

Réciproquement, s'il existe un chemin de u à v de longueur au plus 2^i, alors $\mathsf{path}(G, u, v, i)$ accepte, car il suffit de choisir w comme sommet milieu du chemin.

On a $\mathrm{O}(\log n)$ appels récursifs ; et à chaque appel, on doit mémoriser w ce qui demande $\mathrm{O}(\log n)$ bits. On en déduit une complexité en espace en $\mathrm{O}(\log^2 n)$. □

Preuve (du théorème de Savitch)**.** Supposons que A peut être résolu par une machine non-déterministe M en espace $\mathrm{O}(s(n))$ où $s(n)$ est constructible en espace. Soit $x \in \{\mathtt{0}, \mathtt{1}\}^n$ une instance de A.

On considère le graphe G_x des ***configurations potentielles*** de la machine M sur l'entrée x, c'est-à-dire l'ensemble des configurations avec x en entrée et au plus $c \cdot s(n)$ cases utilisées sur chaque ruban de travail.

Lemme 3.5. Le graphe G_x a $2^{\mathrm{O}(s(n))}$ sommets et peut être construit en espace $\mathrm{O}(s(n))$. □

On a que

$$x \in A \qquad \iff \qquad (G_x, s, t) \in \textsc{Path},$$

où s est la configuration initiale de M sur l'entrée x et t la configuration acceptante (qu'on supposera unique, *c.f.* preuve de la PSPACE-complétude de QBF). Par le lemme, on a que (G_x, s, t) se fait en espace $\mathrm{O}(s(n))$. Décider si $(G_x, s, t) \in \textsc{Path}$ se fait en espace $\mathrm{O}(\log^2 |G_x|)$ donc $\mathrm{O}(s(n)^2)$. □

Remarque 3.6. La preuve précédente utilise deux résultats

- ▷ le théorème de composition *amélioré* :

Théorème 3.5 (Composition)**.** Soient f et g deux fonctions calculables en espace $s_f(n)$ (*resp.* $s_g(n)$). On peut calculer la composée $(f \circ g)(x)$ en espace $\mathrm{O}(s_g(|x|)) + s_f(|g(x)|)$. □

- et le fait que l'on pourra supprimer l'hypothèse de constructibilité de $s(n)$:

  Pour cela, on essaye $s(n) = 1, 2, \ldots$ et on s'arrête à $s(n) = i$ si aucune configuration de taille $i + 1$ n'est accessible à partir de la configuration initiale sur l'entrée x (appel à l'algorithme pour Path).

Remarque 3.7. Le problème Path dans les graphes non-orientés est dans L ! C'est un résultat récent (2005).

4 Oracles et fonctions de conseil.

Une machine de Turing avec un oracle pour L peut décider en un pas de calcul si un mot donné appartient à L.

Définition 4.1. Une *machine de Turing à oracle* dispose d'un ruban particulier et de trois états particuliers : $q_?$, q_{oui} et q_{non}.

Soit $L \subseteq \Sigma^\star$. La machine M^L se comporte comme une machine de Turing normale, sauf quand elle entre dans l'état $q_?$. Dans ce cas, en notant u le mot sur le ruban de l'oracle,

- si $u \in L$ alors l'état suivant est q_{oui} ;
- si $u \notin L$ alors l'état suivant est q_{non}.

Remarque 4.1. On peut définir aussi des machines *non déterministes* à oracle.

Exemple 4.1.

1. Étant donnée une formule booléenne

$$\phi(x_1, \dots, x_n),$$

   on voudrait former une assignation qui satisfait ϕ s'il y en a. On peut le faire avec un oracle pour SAT avec l'algorithme de ***recherche préfixe*** :

$a \leftarrow \varepsilon$

tant que $|a| < n$ **faire**

On demande à l'oracle si $\phi(a, \mathtt{0}, x_{|a|+2}, \dots, x_n) \in$ Sat.

si l'oracle dit « oui » **alors** $a \leftarrow a\mathtt{0}$

sinon $a \leftarrow a\mathtt{1}$

retourner a

Si $\phi \in$ Sat alors l'algorithme renvoie une assignation qui satisfait ϕ. Sinon, on renvoie $\mathtt{1}^n$.

1. Soit $A \subseteq \Sigma^\star$. On voudrait reconnaitre

$$L_A := \{\mathtt{1}^n \mid A^{=n} \neq \emptyset\},$$

où $A^{=n} := \{w \in A \mid |w| = n\} = A \cap \Sigma^n$.

On peut le faire en temps polynomial avec une machine non-déterministe équipée d'un oracle pour A.

si $x \neq \mathtt{1}^{|x|}$ **alors Rejeter**

Choisir $y \in \Sigma^{|x|}$ de manière non-déterministe.

Demander à l'oracle si $y \in A$.

si l'oracle dit « oui » **alors**

Accepter

sinon

Rejeter

4.1 Relativisation.

On a prouvé les théorèmes de hiérarchie en utilisant la méthode de diagonalisation. On peut étudier les limites de cette méthode grâce à la ***relativisation***.

Théorème 4.1 (Baker, Gill, Solovay)**.** Il existe des oracles A et B tels que $\mathsf{P}^A = \mathsf{NP}^A$ et $\mathsf{P}^B \neq \mathsf{NP}^B$.

On prouvera ce théorème dans la suite de cette section.

Définition 4.2. Soit A un oracle.

On note P^A la classe des langages reconnaissable en temps polynomial par une machine déterministe avec un oracle pour A.

On note NP^A la classe des langages reconnaissable en temps polynomial par une machine non-déterministe avec un oracle pour A.

On étend cette notation avec $\mathscr{C}$ une classe de langages, on définit

$$\mathsf{P}^{\mathscr{C}} := \bigcup_{L \in \mathscr{C}} \mathsf{P}^L \qquad \mathsf{NP}^{\mathscr{C}} := \bigcup_{L \in \mathscr{C}} \mathsf{NP}^L.$$

On peut ainsi défini P^{NP}, $\mathsf{NP}^{\mathsf{NP}}$, *etc.*

Remarque 4.2. On a $\mathsf{P} \neq \mathsf{EXPTIME}$ par le théorème de hiérarchie en temps. La preuve relativise $\mathsf{P}^A \neq \mathsf{EXPTIME}^A$ pour tout oracle A. C'est une propriété des preuves par diagonalisation.

Remarque 4.3. L'interprétation du théorème de Baker-Gill-Solovay est que l'on ne peut pas résoudre « P *vs.* NP » par une méthode de diagonalisation.

La preuve que $\mathsf{IP} = \mathsf{PSPACE}$ ne se relativise pas : il existe A tel que $\mathsf{IP}^A \neq \mathsf{PSPACE}^A$.

Proposition 4.1. Il existe un oracle A tel que $\mathsf{P}^A = \mathsf{NP}^A$.

Preuve. On choisit $A = \mathrm{QBF}$, et on a bien $\mathsf{P}^{\mathrm{QBF}} = \mathsf{PSPACE}$.

- On a $\mathsf{PSPACE} \subseteq \mathsf{P}^{\mathrm{QBF}}$ par PSPACE-complétude de QBF.
- Supposons $L \in \mathsf{P}^{\mathrm{QBF}}$. On a $L \in \mathsf{PSPACE}$ car on peut répondre aux questions posées à l'oracle en espace polynomial puisque $\mathrm{QBF} \in \mathsf{PSPACE}$. D'où $\mathsf{P}^{\mathrm{QBF}} \subseteq \mathsf{PSPACE}$.

Et, on a $\mathsf{NP}^{\mathrm{QBF}} = \mathsf{PSPACE}$. On doit juste montrer que $\mathsf{NP}^{\mathrm{QBF}} \subseteq$

PSPACE. On énumère tous les chemins de calcul d'une machine NP. □

Proposition 4.2. Il existe un oracle A tel que $\mathsf{P}^A \neq \mathsf{NP}^A$.

Preuve. Pour tout A, on note $L_A := \{\mathtt{1}^n \mid A^{=n} \neq \emptyset\}$. On a que $L_A \in \mathsf{NP}^A$. On va construire A tel que $L_A \notin \mathsf{P}^A$.

Soit $(M_i)_{i\geq 1}$ une énumération des machines à oracle telle que M_i fonctionne en temps au plus $p_i(n)$ sur les entrées de taille n, pour un certain polynôme p_i.[1] On va construire A tel que L_A n'est pas reconnu par ${M_i}^A$. Construisons ce A par étapes, où l'étape i assurera que L_A n'est pas reconnu par ${M_i}^A$.

On part de $A = \emptyset$. À chaque étape, on peut ajouter des éléments dans A ou $\bar{A}$: pour un mot $w \in \Sigma^\star$, à l'étape i, il est soit dans A, soit dans $\bar{A}$, soit on ne sait pas encore. « À la fin » (étape ω), les mots non-décidés sont mis dans $\bar{A}$.

À l'étape i, on s'assure que ${M_i}^A$ donne la mauvaise réponse sur l'entrée $\mathtt{1}^{n_i}$. On choisit n_i de sorte que :

1. on ait $2^{n_i} > p_i(n_i)$ (ce qui est toujours vrai pour un n_i assez grand) ;
2. l'entier n_i est strictement supérieur à la longueur de tous les mots pour lesquels on a déjà fait une décision $(*)$.

On simule M_i sur l'entrée $\mathtt{1}^{n_i}$. Pour chaque requête à l'oracle, on donne une réponse consistante avec les décisions précédentes. Plus précisément, on répond « oui » uniquement pour les chaînes w qu'on a déjà décidé dans A (sinon, on ne répond « non » et w est placé dans $\bar{A}$). À la fin du calcul sur $\mathtt{1}^{n_i}$,

1. si M_i rejette, on décide de mettre dans A un mot de longueur n_i sur lequel M_i n'a fait aucune réponse (possible car $2^{n_i} > p_i(n_i)$) ;

2. si M_i accepte $\mathtt{1}^{n_i}$, on décide que $A^{=n_i} = \emptyset$, ce qui est consistant avec les décisions précédentes par $(*)$, la seconde propriété pour le choix de n_i.

En conclusion, ${M_i}^A(\mathtt{1}^{n_i})$ accepte ssi $A^{=n_i} = \emptyset$. Et donc ${M_i}^A$ se trompe sur $\mathtt{1}^{n_i}$. □

On en conclut le théorème de Baker-Gill-Solovay.

4.2 Fonctions de conseil.

Définition 4.3. Une *fonction de conseil* est une fonction $f : \mathbb{N} \to \{\mathtt{0},\mathtt{1}\}^\star$ où, sur l'entrée $x \in \{\mathtt{0},\mathtt{1}\}^n$ le « conseil » $f(n)$ est donné à la machine de Turing (en plus de l'entrée).

Définition 4.4. Soit $\mathscr{C}$ une classe de langages, et $\mathscr{F}$ une classe de fonctions de conseil. On définit $\mathscr{C}/\mathscr{F}$ l'ensemble des langages A tels qu'il existe $B \in \mathscr{C}$ et une fonction de conseil $f \in \mathscr{F}$ telle que :

$$\forall x \in \{\mathtt{0},\mathtt{1}\}^\star, \qquad x \in A \quad\iff\quad \langle x, f(|x|)\rangle \in B.$$

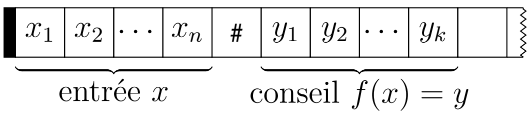

Figure 4.1 | *Ruban d'entrée d'une machine avec fonction de conseil*[2]

1. On énumère les machines polynomiales à horloge, c'est-à-dire que l'on énumère toutes les machines qui s'arrêtent en temps n^k pour tout $k \in \mathbb{N}$. On force l'arrêt au bout de n^k étapes de calculs. On parle de « *clocked Turing machines* » en anglais.

Exemple 4.2. On note

$$\mathsf{poly} := \left\{ f \;\middle|\; \begin{array}{l} \text{il existe } p \text{ un polynôme tel} \\ \text{que } |f(n)| \le p(n) \text{ pour tout } n \end{array} \right\},$$

et

$$\mathsf{log} := \left\{ f \;\middle|\; \begin{array}{l} \text{il existe } c \text{ une constante telle} \\ \text{que } |f(n)| \le c \cdot \log n \text{ pour tout } n \ge 1 \end{array} \right\}.$$

On obtient donc des classes P/poly, $\mathsf{NP}/\mathsf{poly}$, P/log, NP/log, *etc.*

Théorème 4.2. Pour un problème $A \subseteq \{\mathtt{0}, \mathtt{1}\}^\star$, les deux propriétés suivantes sont équivalentes :

1. $A \in \mathsf{P}/\mathsf{poly}$;
2. A peut être résolu par une famille de circuits booléens de taille polynomiale.

Preuve. Pour « 1. $\implies$ 2. », soit $A \in \mathsf{P}/\mathsf{poly}$, et $x \in \{\mathtt{0}, \mathtt{1}\}^n$ avec

$$x \in A \qquad \iff \qquad \langle x, f(n) \rangle \in B,$$

où $B \in \mathsf{P}$. On a que B peut être résolu par une famille de circuits booléens $(C_n)_{n \ge 1}$. Supposons que $\langle x, y \rangle = \mathtt{1}^{|x|}\mathtt{0}xy$. Avec $y = f(n)$, cette chaîne est de longueur $2n+1+|f(n)|$, et il suffit de considérer $C_{2n+1+|f(n)|}$.

Pour « 2. $\implies$ 1. », soit A résolu par une famille de circuits $(C_n)_{n\ge 1}$ de taille polynomiale. On donne comme conseil $f(n)$ une description du circuit C_n. On peut conclure que $A \in \mathsf{P}/\mathsf{poly}$ puisque le problème d'évaluation ValCirc est dans P. $\square$

2. où on construit $\langle \cdot, \cdot \rangle$ avec un séparateur `#`.

5 Hiérarchie polynomiale.

Définition 5.1. Étant donnée une classe de langages $\mathscr{C}$, on définit

$$\mathsf{co}\mathscr{C} := \{\, A \subseteq \Sigma^\star \mid \Sigma^\star \setminus A \in \mathscr{C} \,\}.$$

Définition 5.2. Les classes Σ_i^{P}, pour $i \geq 0$, sont définies par induction :

- ▷ $\Sigma_0^{\mathrm{P}} := \mathsf{P}$;
- ▷ $\Sigma_{i+1}^{\mathrm{P}} := \mathsf{NP}^{\Sigma_i^{\mathrm{P}}}$.

On pose $\mathsf{PH} := \bigcup_{i \geq 0} \Sigma_i^{\mathrm{P}}$.

On définit aussi $\Pi_i^{\mathrm{P}} = \mathsf{co}\Sigma_i^{\mathrm{P}}$ et $\Delta_i^{\mathrm{P}} := \mathsf{P}^{\Sigma_{i-1}^{\mathrm{P}}}$.

Exemple 5.1. On a

- ▷ $\Sigma_1^{\mathrm{P}} = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$,
- ▷ $\Pi_1^{\mathrm{P}} = \mathsf{coNP}$,
- ▷ $\Delta_2^{\mathrm{P}} = \mathsf{P}^{\mathsf{NP}}$,
- ▷ $\Sigma_2^{\mathrm{P}} = \mathsf{NP}^{\mathsf{NP}}$,
- ▷ *etc.*

En général, on a les inclusions suivantes :

$$\begin{tikzcd}[column sep=small]
 & \Sigma_1^{\mathrm{P}} = \mathsf{NP} \arrow[dr, phantom, sloped, "\subseteq"] & & \Sigma_2^{\mathrm{P}} \arrow[dr, phantom, sloped, "\subseteq"] & & \Sigma_3^{\mathrm{P}} \arrow[dr, phantom, sloped, "\subseteq"] & \\
\mathsf{P} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & \Delta_2^{\mathrm{P}} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & \Delta_3^{\mathrm{P}} \arrow[ur, phantom, sloped, "\subseteq"] \arrow[dr, phantom, sloped, "\subseteq"] & & \cdots \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \\
 & \Pi_1^{\mathrm{P}} = \mathsf{NP} \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi_2^{\mathrm{P}} \arrow[ur, phantom, sloped, "\subseteq"] & & \Pi_3^{\mathrm{P}} \arrow[ur, phantom, sloped, "\subseteq"] &
\end{tikzcd}$$

Remarque 5.1. On a que $\Delta_i^{\mathrm{P}} = \mathsf{P}^{\Sigma_{i-1}^{\mathrm{P}}} = \mathsf{P}^{\Pi_{i-1}^{\mathrm{P}}}$. Par exemple, on a que $\Delta_2^{\mathrm{P}} = \mathsf{P}^{\mathsf{NP}} = \mathsf{P}^{\mathsf{coNP}}$.

Les classes Δ_i^{P} sont closes par complément. Par exemple, l'inclusion $\Delta_i^{\mathrm{P}} \subseteq \Pi_i^{\mathrm{P}}$ découle de la clôture par complément et de l'inclusion $\Delta_i^{\mathrm{P}} \subseteq \Sigma_i^{\mathrm{P}}$.

On omet parfois l'exposant « P », mais attention, il existe une hiérarchie similaire (avec Σ_i, Π_i et Δ_i) en calculabilité.

Il ne reste que l'inclusion $\mathsf{PH} \subseteq \mathsf{PSPACE}$ à démontrer.

Proposition 5.1. On a $\mathsf{PH} \subseteq \mathsf{PSPACE}$.

Preuve. On montre par récurrence sur i que $\Sigma_i^{\mathrm{P}} \subseteq \mathsf{PSPACE}$.

- ▷ On a $\Sigma_0^{\mathrm{P}} = \mathsf{P} \subseteq \mathsf{PSPACE}$.
- ▷ Supposons $\Sigma_{i-1}^{\mathrm{P}} \subseteq \mathsf{PSPACE}$. On a

$$\Sigma_i^{\mathrm{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathrm{P}}} \subseteq \mathsf{NP}^{\mathsf{PSPACE}} = \mathsf{PSPACE},$$

car $\mathsf{NP}^{\mathrm{QBF}} = \mathsf{PSPACE}$.

□

Proposition 5.2. Si $\mathsf{P} = \mathsf{NP}$ alors $\mathsf{PH} = \mathsf{P}$.

Plus généralement, pour tout $i \geq 0$, si

$$\Sigma_i^{\mathrm{P}} = \Sigma_{i+1}^{\mathrm{P}},$$

alors $\mathsf{PH} = \Sigma_i^{\mathrm{P}}$. On dit alors que « *la hiérarchie polynomiale s'effondre au i-ème niveau* ».

Preuve. Supposons $\Sigma_i^{\mathrm{P}} = \Sigma_{i+1}^{\mathrm{P}}$.

On montre par récurrence que $\Sigma_j^{\mathrm{P}} = \Sigma_i^{\mathrm{P}}$ pour tout $j \geq i+1$. L'initialisation est vraie par hypothèse. L'étape de récurrence est : supposons $\Sigma_{j-1}^{\mathrm{P}} = \Sigma_i^{\mathrm{P}}$ alors

$$\Sigma_j^{\mathrm{P}} = \mathsf{NP}^{\Sigma_{j-1}^{\mathrm{P}}} = \mathsf{NP}^{\Sigma_i^{\mathrm{P}}} = \Sigma_{i+1}^{\mathrm{P}} = \Sigma_i^{\mathrm{P}}.$$

□

5.1 Caractérisation par quantificateurs.

Théorème 5.1. Un langage A est dans Σ_i^{P} si, et seulement si, il existe $B \in \mathsf{P}$ et un polynôme p tel que, pour tout $x \in \{\mathtt{0}, \mathtt{1}\}^\star$,

$$x \in A \iff \begin{pmatrix} \exists y_1 \in \{\mathtt{0},\mathtt{1}\}^{p(n)} \\ \forall y_2 \in \{\mathtt{0},\mathtt{1}\}^{p(n)} \\ \exists y_3 \in \{\mathtt{0},\mathtt{1}\}^{p(n)} \\ \vdots \\ \mathsf{Q}_i y_i \in \{\mathtt{0},\mathtt{1}\}^{p(n)} \\ \langle x, y_1, y_2, \ldots, y_i \rangle \in B \end{pmatrix},$$

avec une alternance de $\forall$ et de $\exists$. [1] □

Remarque 5.2.

1. On note ici $\mathsf{Q}_i := \exists$ si i est impair et $\mathsf{Q}_i := \forall$ si i est pair.

1. Cette caractérisation est similaire (c'est une généralisation) à la caractérisation de NP avec des certificats.
2. On peut quantifier sur des blocs de taille variables (des chaînes de tailles $p_1(n), p_2(n), \ldots, p_i(n)$). On peut aussi enchaîner plusieurs blocs existentiels sans augmenter le i (il suffit de concaténer les chaînes).
3. On pourrait aussi quantifier sur $y_k \in \{\mathtt{0}, \mathtt{1}\}^{\leq p(n)}$.
4. On a une caractérisation similaire pour la classe Π_i^{P} où on commence par « $\forall y_1 \in \{\mathtt{0}, \mathtt{1}\}^{p(n)}$ ».

Proposition 5.3. Si $\Sigma_i^{\mathrm{P}} = \Pi_i^{\mathrm{P}}$ alors on a que $\Sigma_i^{\mathrm{P}} = \mathsf{PH}$.

Preuve. Montrons $\Sigma_i^{\mathrm{P}} = \Sigma_{i+1}^{\mathrm{P}}$. Soit $A \in \Sigma_{i+1}^{\mathrm{P}}$. On a

$$x \in A \iff \exists y_1\, \forall y_2 \ldots \mathsf{Q}_{i+1} y_{i+1} \langle x, y_1, \ldots, y_{i+1} \in B \rangle,$$

avec $B \in \mathsf{P}$. Et, le langage

$$C := \{\, \langle x, y_1 \rangle \mid \forall y_2 \ldots \mathsf{Q}_{i+1} y_{i+1} \langle x, y_1, \ldots, y_{i+1} \in B \rangle \,\}$$

est dans Π_i^{P}, donc dans Σ_i^{P}. D'où, par caractérisation,

$$x \in A \iff \exists y_1 \underbrace{\exists z_1\, \forall z_2\, \ldots\, \mathsf{Q}_i z_i\, \langle x, y_1, z_1, \ldots, z_i \rangle \in D}_{\langle x, y_1 \rangle \in C},$$

avec $D \in \mathsf{P}$. Et ainsi A est un problème de Σ_i^{P} en combinant les deux « $\exists$ » (avec la remarque précédente). □

Remarque 5.3 (Propriétés supplémentaires).

1. La classe Σ_i^{P} est *close par réduction polynomiale*, c'est-à-dire si $B \in \Sigma_i^{\mathrm{P}}$ et $A \leq_{\mathrm{P}} B$ alors $A \in \Sigma_i^{\mathrm{P}}$.
2. Le problème de décision QBF-Σ_i^{P} est Σ_i^{P}-complet, où

QBF-Σ_i^{P}	**Entrée.** Une formule booléenne quantifiée F avec i quantificateurs et commençant par un bloc existentiel **Sortie.** Est-ce que F est vraie ?

3. De même, le problème de décision QBF-Π_i^{P} est Π_i^{P}-complet, où

QBF-Π_i^{P}	**Entrée.** Une formule booléenne quantifiée F avec i quantificateurs et commençant par un bloc universel **Sortie.** Est-ce que F est vraie ?

5.2 Théorème de Karp-Lipton.

Théorème 5.2 (Karp-Lipton). Si $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{poly}$, alors $\Sigma_2^{\mathsf{P}} = \Pi_2^{\mathsf{P}}$.

Définition 5.3. Un circuit booléen à s entrées *décide* SAT si, étant donnée une formule booléenne F de taille s, le circuit C décide si F est satisfiable.

La preuve de ce théorème repose sur deux lemmes.

Lemme 5.1. L'ensemble des (codages de) circuits qui décident SAT est dans coNP.

Preuve. On utilise le fait que SAT est ***auto-réductible***[2] : une formule booléenne $F(v_1, \ldots, v_n)$ est satisfiable si et seulement si l'une des deux formules booléennes

$$F(v_1, \ldots, v_{n-1}, \mathtt{0}) \quad \text{ou} \quad F(v_1, \ldots, v_{n-1}, \mathtt{1})$$

est satisfiable.

Un circuit C décide SAT ssi pour toute formule F de taille s

1. si F n'a pas de variable, alors $C(F) = \mathtt{1}$ ssi $F \equiv \mathtt{1}$;

2. si F dépend de $n \geq 1$ variables $v_1, \ldots, v_n$ alors $C(F) = \mathtt{1}$ ssi

$$C(F[v_n := \mathtt{0}]) = \mathtt{1} \quad \text{ou} \quad C(F[v_n := \mathtt{1}]) = \mathtt{1}.$$

Étant donnée F, les conditions ci-dessous peuvent être vérifiées en temps polynomial (car ValCirc est dans P).

Cette caractérisation commence par un « pour toute formule » et on considère ensuite un problème dans P, d'où le langage est bien dans coNP. □

Lemme 5.2. Si NP $\subseteq$ P/poly, alors Sat peut être décidé par une famille de circuits booléens de taille polynomiale. □

Preuve (du théorème de Karp-Lipton). On suppose avoir l'inclusion des classes NP $\subseteq$ P/poly. Il suffit de montrer que $\Pi_2^{\mathsf{P}} \subseteq \Sigma_2^{\mathsf{P}}$. En effet, avec ça on a que

$$\Sigma_2^{\mathsf{P}} = \mathsf{co}\Pi_2^{\mathsf{P}} \subseteq \mathsf{co}\Sigma_2^{\mathsf{P}} = \Pi_2^{\mathsf{P}}.$$

Il suffit de montrer que le problème de décision QBF-Π_2^{P} est dans Σ_2^{P}. Soit F une formule booléen de taille s, alors

$$\forall u \,\exists v \quad F(u, v),$$

est équivalente à

$$\exists C \,\forall u \quad C(F(u, \cdot)) = \mathtt{1} \quad \text{et} \quad C \text{ décide } \textsc{Sat},$$

où C est un circuit booléen avec s entrées. Il suffit de quantifier sur des circuits de taille polynomiale d'après le lemme 5.2. Ceci

2. *self-reductible* en anglais.

est équivalent à

$$\exists C\, \forall u \quad C(F(u,\cdot)) = \mathtt{1} \quad \text{et} \quad \forall y \in \{\mathtt{0},\mathtt{1}\}^{p(s)}\ \langle C, y\rangle \in A,$$

avec $A \in \mathsf{P}$ d'après le lemme 5.1 et la caractérisation de coNP. On en déduit que ceci est équivalent à

$$\exists C\, \forall u\, \forall y \quad \underbrace{C(F(u,\cdot)) = \mathtt{1} \quad \text{et} \quad \langle C, y\rangle \in A}_{\text{vérifiable en temps polynomial}},$$

et est donc vérifiable dans Σ_2^{P} grâce à la caractérisation par quantificateurs. $\square$