

Complexité Algorithmique

*Basé sur le cours de Pascal KOIRAN
Notes prises par Hugo SALOU*



ENS DE LYON

10 février 2026

Table des matières

1	Machines de Turing.	4
1.1	Définitions.	4
1.2	Non déterminisme.	7
1.3	NP-complétude.	8
2	Circuits booléens.	10
2.1	Définitions.	10
2.2	Simulation de machines de Turing par les circuits.	11
2.3	Premiers problèmes NP-complets.	14
3	Complexité en espace.	16
3.1	Définitions et premières propriétés.	16
3.2	Hiérarchie en espace.	19
3.3	Formules booléennes quantifiées.	22

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, \dots, 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 ;
- ▷ $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\leftarrow, \rightarrow, I\}^k$ 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 $x \in \Sigma^*$ est l'entrée.

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

On dit qu'un langage $L \subseteq \Sigma^*$ est reconnu si on peut calculer la fonction $\mathbb{1}_L : \Sigma^* \rightarrow \{0, 1\} \subseteq \Sigma$. 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 $\Gamma = \{0, 1, 2, 3, \square, \triangleright\}$, c'est-à-dire un alphabet plus gros. Ce modèle est équivalent avec l'association

$$0 \leftrightarrow 00 \quad 1 \leftrightarrow 01 \quad 2 \leftrightarrow 10 \quad 3 \leftrightarrow 11.$$

Définition 1.3. Un langage $L \subseteq \Sigma^*$ 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 $\text{DTIME}(f(n))$ s'il existe une machine (à plusieurs rubans) qui reconnaît L en temps $O(f(n))$. On supposera toujours avoir $f(n) \geq n + 1$.²

1. Généralement, on prendra $\Sigma \subseteq \{0, 1\}$.

2. Il faut, au moins, lire l'entrée.

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

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

Théorème 1.1. Si $L \in \text{DTIME}(f(n))$ alors L peut être reconnu en temps $O(f(n)^2)$ 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 $O(T(n) \log T(n))$ telle que $M(x) = M'(x)$ pour toute entrée $x \in \Sigma^*$.

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

Définition 1.7. Une *machine universelle* U prend en entrée des couples $\langle x, \alpha \rangle$ (où α est le code d'une machine M_α et $x \in \Sigma^*$) 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 M_α en temps linéaire. En effet, on utilise quatre rubans (figure 1.1) :

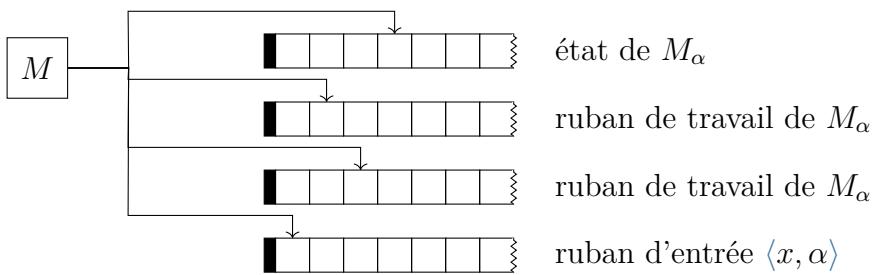


Figure 1.1 | Construction de U dans le cas à deux rubans

- 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

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

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

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

Définition 1.9. On note $\text{NTIME}(f(n))$ l'ensemble des langages acceptés par une machine de Turing non-déterministe fonctionnant en temps $O(f(n))$ 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,

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

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

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

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 $f : \Sigma^* \rightarrow \Sigma^*$ calculable en temps polynomial telle que, pour toute entrée x ,

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

On note alors $A \leq_{\text{P}} B$ ou $A \leq_{\text{M}} B$.³

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

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

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

- ▷ $A \in \text{NP}$;
- ▷ A est **NP-dur**, c'est-à-dire $B \leq_P A$ pour tout $B \in \text{NP}$.

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

$$C \leq_P B \leq_P A.$$

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 DAG¹ dont les sommets sont étiquetés et de degré entrant 0, 1 ou 2 :

- ▷ les sommets de degré entrant 2 sont étiquetés par \wedge ou \vee ;
- ▷ 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, \dots, 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, \dots, x_n . Étant donné une *valuation* $a \in \{0, 1\}^n$, 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

$$\text{val}(x_i) := a_i, \quad \text{val}(0) := 0, \quad \text{val}(1) := 1;$$

- ▷ pour les autres portes, on pose

- $\text{val}(\alpha \vee \beta) = \text{val}(\alpha) \vee \text{val}(\beta)$,
- $\text{val}(\alpha \wedge \beta) = \text{val}(\alpha) \wedge \text{val}(\beta)$,
- $\text{val}(\neg \alpha) = \text{not val}(\alpha)$.

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 $\text{val}(C) := \text{val}(\alpha)$.

On peut donc dire que C calcule une fonction $f : \{0, 1\}^n \rightarrow \{0, 1\}$ 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}^s$.

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 $\text{VALCIRC} \in \mathbf{P}$ où

VALCIRC | **Entrée.** Une circuit booléen C et $a \in \{0, 1\}^n$
Sortie. Est-ce que $\text{val}(C) = 1$ 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 $O(T(n)^2)$.

Remarque 2.2. On peut donner la borne de $O(T(n) \log T(n))$ au lieu de $O(T(n)^2)$, 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)

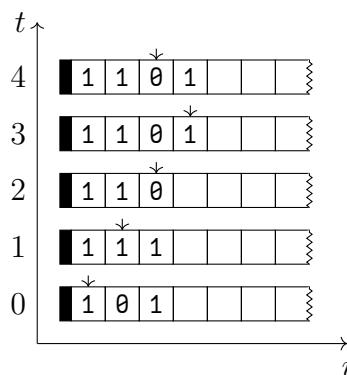


Figure 2.1 | Exemple de diagramme espace-temps

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 . Définissons 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

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

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

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

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 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 $L \subseteq \{0, 1\}^*$ est dans \mathbf{P} si et seulement si L est reconnu par une famille uniforme de circuits booléens de taille polynomiale.

Preuve. ▷ « \implies ». Soit $L \in \mathbf{P}$. 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.

- ▷ « \iff ». 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).

□

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** $\in \text{NP}$ avec a le certificat et **VALCIRC** le problème de vérification.
- ▷ D'autre part, soit $L \in \text{NP}$. Il existe une machine de Turing non-déterministe M fonctionnant en temps polynomial 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 $O(T(n)^2)\dots$ mais ce n'est pas suffisant : il faut modéliser le non-déterminisme.

On ajoute des variables d'entrée supplémentaires $y_1, \dots, y_{T(n)}$ 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 $x \in L$.

□

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** ∈ **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 la 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.

□

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 utilisées sur les rubans de travail¹ 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 $\text{DSPACE}(s(n))$ l'ensemble des langages reconnus par une machine de Turing déterministe fonctionnant en espace $O(s(n))$.

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 $O(n)$. 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 $O(\log n)$. 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 $t : \mathbb{N} \rightarrow \mathbb{N}$ est *constructible en espace* s'il existe une machine de Turing qui, sur l'entrée 1^n calcule $1^{t(n)}$ en espace $O(t(n))$.

Proposition 3.1. On a l'inclusion

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

Preuve. Supposons f constructible en espace (*).

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

- 1 : Calculer $f(n)$ en espace $O(f(n))$
- 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 $t = 1, 2, \dots$ 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

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

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 $N = 2^{O(s(n))}$. On compte le nombre de configurations distinctes possibles d'une machine M sur l'entrée x de taille n . Une configuration est donnée² 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 $\text{DSPACE}(1) \subseteq \text{DTIME}(1)$? Non! En effet, retourner le dernier caractère de l'entrée se fait en espace $O(1)$ mais pas en temps $O(1)$.

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

Corollaire 3.1. On a les inclusions

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

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

Théorème 3.1. Si deux fonctions $f, g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ sont calculables en espace $s(n) \geq \log n$ alors leur composée $g \circ f$ est

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).

calculable en espace $O(s(|x|)) + s(|f(x)|)$.

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 $O(s(|x|))$.

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 $f(x)$. \square

On utilise ensuite l'algorithme suivant :

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

\square

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

3.2 Hiérarchie en espace.

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

Corollaire 3.3. On a les inclusions strictes suivantes :

$$\underbrace{\text{DSPACE}(\log n)}_{\mathsf{L}} \subsetneq \text{DSPACE}(n) \subsetneq \text{DSPACE}(n^2) \subsetneq \cdots \subsetneq \text{PSPACE}.$$

Note 3.1. Il manque une partie du cours ici... mais, promis, je la recopie bientôt.

Remarque 3.3 (Rappels). On a que

- ▷ $\text{NSPACE}(s(n)) \subseteq \text{DTIME}(2^{O(s(n))})$
- ▷ et $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$ (Savitch)

lorsque $s(n) \geq \log n$. On a

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE},$$

et avec Savitch et la hiérarchie en espace, on a

$$\text{NL} \subsetneq \text{PSPACE}.$$

Définition 3.4. On définit

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

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

$$\text{P} \subseteq \text{NP} \stackrel{(*)}{\subseteq} \text{PSPACE} \stackrel{(**)}{\subseteq} \text{EXPTIME} = \bigcup_{k \geq 1} \text{DTIME}(2^{n^k}),$$

où

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

On sait, de plus, que $\text{P} \subsetneq \text{EXPTIME}$ par la hiérarchie en temps (variante (vue en TD) de la hiérarchie en espace vue avant).

3.3 Formules booléennes quantifiées.

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

Exemple 3.2. Les formules

- ▷ $\forall x \exists y \left(\overbrace{(x \vee y) \wedge (\bar{x} \vee \bar{y})}^{c'est\;x\;xor\;y} \right)$
- ▷ $\exists y \forall x \left((x \vee y) \wedge (\bar{x} \vee \bar{y}) \right)$

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 3.5. On définit le problème **QBF** comme

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

Proposition 3.3. On a $\text{QBF} \in \text{PSPACE}$.

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 3.3. 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 $A \leq_P \text{QBF}$.

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 $\phi_{c \cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$.³

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 :

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

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 c, c_3, c_4 ne sont pas des variables mais des n -uplets de taille $O(n^k)$ variables booléennes. La taille de ϕ_{t+1} augmente de $O(n^k)$ par rapport à la taille de ϕ_t . Au final, on est de taille $O(n^{2k})$.⁴ □

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

Théorème 3.4 (Savitch). On a $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$ si $s(n) \geq \log n$.⁵ □

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

3. 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.

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

5. 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 que $\text{PATH} \in \text{DSPACE}(\log^2 n)$, où

PATH | **Entrée.** Un graphe G orienté, et $s, t \in V(G)$

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 $A \in \text{NL}$ et $x \in \{\theta, 1\}^n$, on a

$$x \in A \iff f(x) \in \text{PATH},$$

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

Preuve (de la proposition). On donne un algorithme $\text{path}(G, u, v, i)$ en espace $O(\log^2 n)$ 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 $\text{path}(G, s, t, \lceil \log n \rceil)$.

```

1 : Procédure path( $G, u, v, i$ )
2 :   si  $i = 0$  alors
3 :     si  $uv \in E(G)$  ou  $u = v$  alors Accepter
4 :     sinon Rejeter
5 :   pour tout sommet  $w \in V(G)$  faire
6 :     si path( $G, u, w, i - 1$ ) et path( $G, w, v, i - 1$ ) alors
7 :       Accepter
8 :   Rejeter

```

Pour la correction, on montre si $\text{path}(G, u, v, i)$ 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'hérité, 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 $\text{path}(G, u, v, i)$ accepte, car il suffit de choisir w comme sommet milieu du chemin.

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

Preuve (du théorème de Savitch). Supposons que A peut être résolu par une machine non-déterministe M en espace $O(s(n))$ où $s(n)$ est constructible en espace. Soit $x \in \{0, 1\}^n$ 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 3.2. Le graphe G_x a $2^{O(s(n))}$ sommets et peut être construit en espace $O(s(n))$. \square

On a que

$$x \in A \iff (G_x, s, t) \in \text{PATH},$$

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 $O(s(n))$. Décider si $(G_x, s, t) \in \text{PATH}$ se fait en espace $O(\log^2 |G_x|)$ donc $O(s(n)^2)$. \square

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 $O(s_g(|x|)) + s_f(|g(x)|)$. \square

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

Pour cela, on essaye $s(n) = 1, 2, \dots$ 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).

La prochaine fois, on s'intéresse aux oracles.