

# Complexité Algorithmique

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



*27 février 2026*

# Table des matières

<b>1</b>	<b>Machines de Turing.</b>	<b>4</b>
1.1	Définitions. . . . .	4
1.2	Non déterminisme. . . . .	7
1.3	NP-complétude. . . . .	8
<b>2</b>	<b>Circuits booléens.</b>	<b>10</b>
2.1	Définitions. . . . .	10
2.2	Simulation de machines de Turing par les circuits. . . .	11
2.3	Premiers problèmes NP-complets. . . . .	14
<b>3</b>	<b>Complexité en espace.</b>	<b>16</b>
3.1	Définitions et premières propriétés. . . . .	16
3.2	Hierarchie en espace. . . . .	19
3.3	Complexité en espace non-déterministe. . . . .	21
3.4	Formules booléennes quantifiées, PSPACE. . . . .	22
3.5	Problème PATH et théorème de Savitch. . . . .	25
<b>4</b>	<b>Oracles et fonctions de conseil.</b>	<b>29</b>
4.1	Relativisation. . . . .	30
4.2	Fonctions de conseil. . . . .	33

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 une 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  $(\Gamma, Q, \delta)$  où

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

■ **Remarque 1.1.** On fixe

- ▷ un *ruban d'entrée* (généralement supposé en lecture unique-ment),
- ▷ un *ruban de sortie* (généralement supposé en écriture unique-ment),
- ▷ un état initial  $q_{\text{initial}}$  ;
- ▷ un état final  $q_{\text{final}}$  ;

- ▷ un symbole blanc  $\square \in \Gamma$  ;
- ▷ un symbole de départ  $\triangleright \in \Gamma$  ;
- ▷ un alphabet d'entrée  $\Sigma \subseteq \Gamma \setminus \{\triangleright, \square\}$ .<sup>1</sup>

Au départ,  $M$  est dans l'état  $q_{\text{initial}}$ , 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  $\mathbb{1}_L : \Sigma^* \rightarrow \{\emptyset, 1\} \subseteq \Sigma$ . On peut aussi avoir un état  $q_{\text{accepte}}$  acceptant et un état  $q_{\text{rejet}}$  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  $\Gamma = \{\emptyset, 1, 2, 3, \square, \triangleright\}$ , c'est-à-dire un alphabet plus gros. Ce modèle est équivalent avec l'association

$$\emptyset \leftrightarrow \emptyset\emptyset \quad 1 \leftrightarrow \emptyset 1 \quad 2 \leftrightarrow 1\emptyset \quad 3 \leftrightarrow 11.$$

☞ **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  $f(n) \geq n + 1$ .<sup>2</sup>

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

2. Il faut, au moins, lire l'entrée.

▮ **Définition 1.5.** La classe  $\mathbf{P}$  est l'ensemble des langages reconnaissables en temps polynomial :

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

▮ **Théorème 1.1.** Si  $L \in \mathbf{DTIME}(f(n))$  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  $\langle x, y \rangle = 1^{|x|}0xy$ .

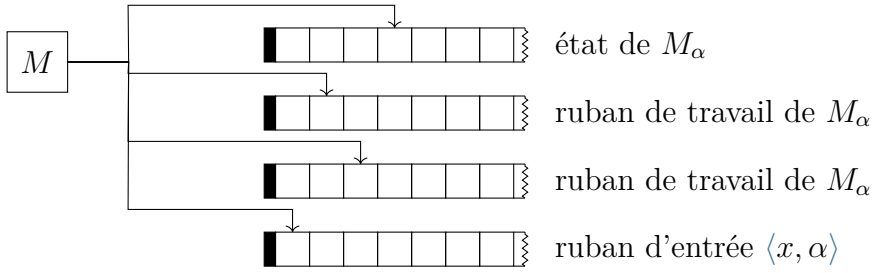
▮ **Définition 1.7.** Une *machine universelle*  $U$  prend en entrée des couples  $\langle x, \alpha \rangle$  (où  $\alpha$  est le code d'une machine  $M_\alpha$  et  $x \in \Sigma^*$ ) et simule la machine  $M_\alpha$  sur l'entrée  $x$ . Autrement dit, pour tout  $x$  et tout  $\alpha$ , on a

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

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

▮ **Preuve.**

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



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

$M_\alpha$  en temps linéaire. En effet, on utilise quatre rubans (figure 1.1) :

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

Pour faire une étape de calcul de  $M_\alpha$ , 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  $M_\beta$  à deux rubans équivalente et on peut simuler  $M_\beta$  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  $(\Gamma, Q, \delta)$  où

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

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

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

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

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

## 1.3 $\text{NP}$ -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  $x$ ,

$$x \in A \quad \Longleftrightarrow \quad f(x) \in B.$$

On note alors  $A \leq_{\mathcal{P}} B$  ou  $A \leq_M B$ .<sup>3</sup>

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

---

3. Personnellement, je noterai parfois  $f : A \leq_{\mathcal{P}} B$  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  $\phi$   
**Sortie.** La formule  $\phi$  est-elle satisfiable ?

**3-SAT** | **Entrée.** Une formule booléenne  $\phi$  sous 3-CNF  
**Sortie.** La formule  $\phi$  est-elle satisfiable ?

# 2 Circuits booléens.

## 2.1 Définitions.

■ **Définition 2.1.** Un *circuit booléen* est un DAG<sup>1</sup> 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  $\alpha$  de  $C$ , on définit la *valeur* prise par  $\alpha$  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
  - $\text{val}(\alpha \vee \beta) = \text{val}(\alpha) \vee \text{val}(\beta)$ ,
  - $\text{val}(\alpha \wedge \beta) = \text{val}(\alpha) \wedge \text{val}(\beta)$ ,
  - $\text{val}(\neg \alpha) = \text{not } \text{val}(\alpha)$ .

---

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

▮ **Définition 2.3.** Si  $C$  n'a qu'une seule porte  $\alpha$  de degré sortant 0, on dit que  $\alpha$  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

$\text{VALCIRC}$	<p><b>Entrée.</b> Un circuit booléen <math>C</math> et <math>a \in \{0, 1\}^n</math></p> <p><b>Sortie.</b> Est-ce que <math>\text{val}(C) = 1</math> sous l'entrée <math>a</math> ?</p>
------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

▮ **Preuve.** On utilise l'algorithme suivant :

- 1 : Calculer un tri topologique de  $C$  (pour la boucle ci-dessous).
- 2 : **pour** toute porte  $\alpha$  de  $C$  **faire**
- 3 :     $\lfloor$  Évaluer  $\alpha$  à l'aide des valeurs déjà calculées<sup>2</sup>
- 4 : **retourner** la valeur de la porte de sortie

□

## 2.2 Simulation de machines de Turing par les circuits.

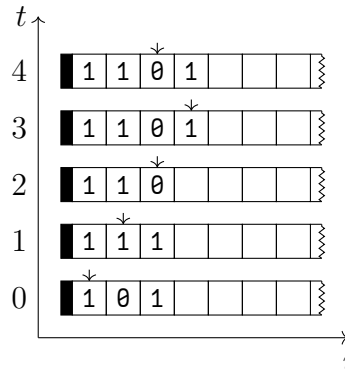
---

2. Par l'ordre topologique, on sait que les entrées de  $\alpha$  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  $\alpha$  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  $\alpha$ .

**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 **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.

- ▷ «  $\Leftarrow$  ». 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	<b>Entrée.</b> Une circuit booléen $C$ avec $n$ variables <b>Sortie.</b> Existe-t-il $a$ tel que $C(a) = 1$ ?
------------	------------------------------------------------------------------------------------------------------------------

▮ **Preuve.**

- ▷ D'une part, CIRCUITSAT  $\in$  NP avec  $a$  le certificat et VAL-CIRC 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  $O(T(n)^2)$ ... mais ce n'est pas suffisant : il faut modéliser le non-déterminisme.

On ajoute des variables d'entrée supplémentaires  $y_1, \dots, y_{T(n)}$  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  $\phi$  sous 3-CNF  
**Sortie.** La formule  $\phi$  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  $\alpha$ , on crée une variable  $z_\alpha$  qui représente la valeur prise par le porte  $\alpha$ . On utilise l'identité  $(P \Rightarrow Q) \Leftrightarrow \bar{P} \vee Q$  pour construire les clauses qui contraignent les variables  $z_\alpha$  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 utilisés sur les rubans de travail<sup>1</sup> 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  $\text{SAT}$  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 donnée<sup>2</sup> 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  $\text{L} = \text{DSPACE}(\log n)$ .

▮ **Corollaire 3.1.** On a les inclusions

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

où  $\text{PSPACE}$  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  $f, g : \{0,1\}^* \rightarrow \{0,1\}^*$  sont calculables en espace  $s(n) \geq \log n$  alors leur composée  $g \circ f$  est 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  $f(x)$ . □

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 :      $i \leftarrow i - 1$ 

```

□

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

## 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  $O(s(n))$  mais pas en espace  $o(s(n))$ .

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

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

▮ **Lemme 3.3.** Il existe une machine de Turing universelle  $U$  qui prend en entrée  $\langle M, x \rangle$  avec  $x \in \{0, 1\}^*$  et  $M$  une machine à un ruban de travail et dont l'alphabet est  $\{0, 1, \square, \triangleright\}$ , et qui simule  $M$  sur l'entrée  $x$  en espace  $O(s(n))$  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  $D$ ).

- 1 : Lire l'entrée  $w \in \{0, 1\}^*$  (notons  $n$  sa taille)
- 2 : Calculer  $s(n)$  en espace  $O(s(n))$
- 3 : Réserver  $s(n)$  cases sur le ruban de travail
- 4 : **si**  $w$  n'est pas de la forme  $\langle M \rangle 10^{*3}$  **alors**
- 5 :    ▮ **Rejeter**
- 6 : Simuler  $M$  sur l'entrée  $w$
- 7 : **si** on dépasse  $2^{2 \cdot s(n)}$  étapes ou  $s(n)$  cases mémoires **alors**
- 8 :    ▮ **Rejeter**
- 9 : **si**  $M$  accepte **alors Rejeter**
- 10 : **sinon Accepter**

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

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

- ▷ 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  $2^{o(s(n))}$  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  $\langle M \rangle 10^{n_2}$ , 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 :

$$L \subsetneq DSPACE(n) \subsetneq DSPACE(n^2) \subsetneq \dots \subsetneq PSPACE.$$

### 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  $NSPACE(s(n))$  l'ensemble des langages reconnus par une machine de Turing non-déterministe en espace  $O(s(n))$ .

▮ **Définition 3.5.** On définit  $NL := NSPACE(\log n)$ .

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

▮ **Proposition 3.3.** On a

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

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  $c_1 c_2 \in E(G_x)$  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  $2^{O(s(n))}$  sommets, et on peut le construire en espace  $O(s(n))$ . □

On explore  $G_x$  à partir de la configuration initiale  $c_{\text{initiale}}$  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

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

▮ **Théorème 3.3 (Savitch).** On a  $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$  si  $s(n) \geq \log n$ .<sup>4</sup>

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

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

où

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

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

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

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

QBF	<p><b>Entrée.</b> Une formule booléenne quantifiée <math>F</math> (close)</p> <p><b>Sortie.</b> Est-ce que <math>F</math> est vraie ?</p>
-----	-------------------------------------------------------------------------------------------------------------------------------------------

---

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  $\text{QBF} \in \text{PSPACE}$ .

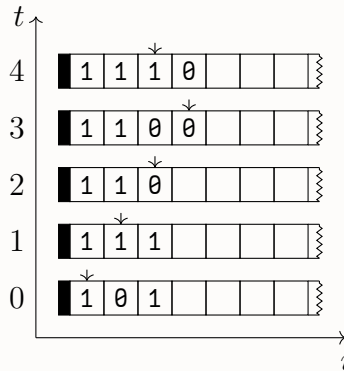
**Preuve.** On utilise l'algorithme suivant.

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.  $\square$

**Théorème 3.4.** Le problème  $\text{QBF}$  est  $\text{PSPACE}$ -complet.

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



**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  $\phi_{c \cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$ .<sup>5</sup>

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

Pour aller de  $\phi_t$  à  $\phi_{t+1}$ , 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  $\phi_{t+1}$  augmente de  $O(n^k)$  par rapport à la taille de  $\phi_t$ . Au final, on est de taille  $O(n^{2k})$ .<sup>6</sup>  $\square$

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  $\mathbf{NL} \subseteq \mathbf{DSPACE}(\log^2 n)$ .

▮ **Proposition 3.5.** On a que  $\mathbf{PATH} \in \mathbf{DSPACE}(\log^2 n)$ , 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  $s, t \in V(G)$   
**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  $A \in \mathbf{NL}$  et  $x \in \{0, 1\}^n$ , on a

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

où  $f : A \leq_L \mathbf{PATH}$  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  $f(x) \in \mathbf{PATH}$  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  $\text{path}(G, u, v, i)$ 
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  $\text{path}(G, u, w, i - 1)$  et  $\text{path}(G, w, v, i - 1)$  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'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  $\text{path}(G, u, v, i)$  accepte, car il suffit de choisir  $w$  comme sommet milieu du chemin.

On a  $O(\log n)$  appels récurifs ; 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 3.5.** Le graphe  $G_x$  a  $2^{O(s(n))}$  sommets et peut être construit en espace  $O(s(n))$ .  $\square$

On a que

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

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).

## 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 :  $q_?$ ,  $q_{\text{oui}}$  et  $q_{\text{non}}$ .

Soit  $L \subseteq \Gamma^*$ . 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  $q_{\text{oui}}$  ;
- ▷ si  $u \notin L$  alors l'état suivant est  $q_{\text{non}}$ .

▮ **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  $\phi$  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  $\phi(a, \emptyset, x_{|a|+2}, \dots, x_n) \in \text{SAT}$ .

**si** l'oracle dit « oui » **alors**  $a \leftarrow a\emptyset$

└ **sinon**  $a \leftarrow a1$

**retourner**  $a$

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

1. Soit  $A \subseteq \Sigma^*$ . On voudrait reconnaître

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

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

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

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

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

Demander à l'oracle si  $y \in L$ .

**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  $\text{P}^A = \text{NP}^A$  et  $\text{P}^B \neq \text{NP}^B$ .

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  $\mathcal{C}$  une classe de langages, on définit

$$P^{\mathcal{C}} := \bigcup_{L \in \mathcal{C}} P^L \quad NP^{\mathcal{C}} := \bigcup_{L \in \mathcal{C}} NP^L.$$

On peut ainsi définir  $P^{NP}$ ,  $NP^{NP}$ , etc.

☞ **Remarque 4.2.** On a  $P \neq EXPTIME$  par le théorème de hiérarchie en temps. La preuve relativise  $P^A \neq EXPTIME^A$  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  $IP = PSPACE$  ne se relativise pas : il existe  $A$  tel que  $IP^A \neq PSPACE^A$ .

☞ **Proposition 4.1.** Il existe un oracle  $A$  tel que  $P^A = NP^A$ .

☞ **Preuve.** On choisit  $A = QBF$ , et on a bien  $P^{QBF} = PSPACE$ .

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

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

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

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

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

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  $p_i$ .<sup>1</sup> 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  $w \in \Sigma^*$ , à l'étape  $i$ , il est soit dans  $A$ , soit dans  $\bar{A}$ , soit on ne sait pas encore. « À la fin » (étape  $\omega$ ), 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  $1^{n_i}$ . 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  $1^{n_i}$ . 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  $1^{n_i}$ ,

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

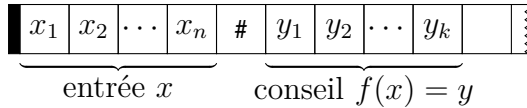
On en conclut le théorème de Baker-Gill-Solovay.

## 4.2 Fonctions de conseil.

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

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

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



**Figure 4.1** | Ruban d'entrée d'une machine avec fonction de conseil<sup>2</sup>

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

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

et

$$\text{log} := \left\{ f \mid \begin{array}{l} \text{il existe } c \text{ une constante telle} \\ \text{que } |f(n)| \leq c \cdot \log n \text{ pour tout } n \geq 1 \end{array} \right\}.$$

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

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

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

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

$$x \in A \quad \Longleftrightarrow \quad \langle x, f(n) \rangle \in B,$$

où  $B \in \text{P}$ . On a que  $B$  peut être résolu par une famille de circuits booléens  $(C_n)_{n \geq 1}$ . Supposons que  $\langle x, y \rangle = 1^{|x|}0xy$ . 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  $(C_n)_{n \geq 1}$  de taille polynomiale. On donne comme conseil  $f(n)$  une description du circuit  $C_n$ . On peut conclure que  $A \in \text{P/poly}$  puisque le problème d'évaluation  $\text{VALCIRC}$  est dans  $\text{P}$ .  $\square$

---

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