

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

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

Remarque 1. On peut définir aussi des machines *non déterministes* à oracle.

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

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 ϕ . Sinon, on renvoie 1^n .

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

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

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

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

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 1 (Baker, Gill, Solovay). Il existe des oracles A et B tels que $P^A = NP^A$ et $P^B \neq NP^B$.

On prouvera ce théorème dans la suite de cette section.

Définition 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 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 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 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 2. Il existe un oracle A tel que $P^A \neq NP^A$.

Preuve. Pour tout A , on note $L_A := \{1^n \mid A^{=n} \neq \emptyset\}$. 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 .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 $w \in \Sigma^*$, 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 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

Figure 1 | Machine avec fonction de conseil²

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

2 Fonctions de conseil.

Définition 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. 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 :

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

Exemple 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 P/poly , NP/poly , P/log , NP/log , etc.

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.

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

Théorème 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

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

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 VALCIRC est dans P . \square