Chapitre 6

Les classes PSPACE et IP

Complexité de temps

Nous allons considérer le temps comme mesure de complexité.

Nous avons déjà vu les classes de complexité P et EXPTIME.

Rappel :

$$\mathsf{EXPTIME} = \bigcup_{k \geq 0} \mathsf{TIME}(2^{n^k}).$$

Définition 6.1. Une fonction

$$t : \mathbb{N} \to \mathbb{N}$$

est totalement-temps-constructible s'il existe une MT M telle que

$$L(M) \in \mathsf{TIME}(t(n))$$

et si $\forall n \in \mathbb{N}$ et $\forall w \in \Sigma^*$ tels que $|w| = n$, on a que M s'arrête après $t(n)$ étapes sur entrée w. ▲

Remarques 6.2.

- Les fonctions $\lfloor \log(n+1) \rfloor$, n, n^k, 2^n et $n!$ sont totalement-temps-constructibles.

- Si t_1 et t_2 sont totalement-temps-constructibles, alors $t_1(n)t_2(n)$, $2^{t_1(n)}$ et $t_1(n)^{t_2(n)}$ sont totalement-temps-constructibles.

▲

Théorème 6.3. Si t_2 est totalement-temps-constructible et si

$$\liminf_{n\to\infty} \frac{t_1(n)\log(t_1(n))}{t_2(n)} = 0$$

alors $\mathsf{TIME}(t_1(n)) \subset \mathsf{TIME}(t_2(n))$. ■

Cela signifie, par exemple, que plus de langages peuvent être reconnus en temps $O(n^3)$ qu'en temps $O(n^2)$.

Théorème 6.4.

$$\mathsf{P} \subset \mathsf{EXPTIME}$$

■

Complexité d'espace

Considérons maintenant l'espace comme mesure de complexité :

- l'espace logarithmique ;
- l'espace polynômial ;
- l'espace exponentiel.

Définition 6.5. Soit

$$s : \mathbb{N} \to \mathbb{N}.$$

Une MT utilise un espace $s(n)$ si, pour tout mot de longueur n, la tête de lecture-écriture ne se déplace jamais vers la droite de plus de $s(n)$ positions.

Pour définir un espace de calcul inférieur à la taille de l'input, nous considérons que l'input réside sur un ruban en lecture seule, et qu'un autre ruban est utilisé pour faire le calcul. ▲

Définition 6.6.

$$\mathsf{SPACE}(s(n)) = \{L \mid L \text{ est décidé par une MT en } \overset{\text{espace}}{\cancel{\text{temps}}}\ O(s(n))\}.$$

▲

Définition 6.7.

$$\mathsf{L} = \mathsf{SPACE}(\log n)$$

▲

Définition 6.8.

$$\mathsf{PSPACE} = \bigcup_{k \geq 0} \mathsf{SPACE}(n^k)$$

▲

Définition 6.9.

$$\mathsf{EXPSPACE} = \bigcup_{k \geq 0} \mathsf{SPACE}(2^{n^k})$$

▲

Théorème 6.10.

$$\mathsf{TIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$$

■

et f(n) est totalement-temps-constructible

Théorème 6.11. Si $L \in \mathsf{SPACE}(f(n))$ et $f(n) \geq \log n$, alors il existe une constante c qui dépend de L telle que $L \in \mathsf{TIME}(c^{f(n)})$. ■

Définition 6.12. Une fonction

$$s : \mathbb{N} \to \mathbb{N}$$

est totalement-espace-constructible s'il existe une MT M telle que

$$L(M) \in \mathsf{SPACE}(s(n))$$

et si $\forall n \in \mathbb{N}$ et $\forall w \in \Sigma^*$ tels que $|w| = n$, on a que M utilise exactement $s(n)$ positions du ruban sur entrée w. ▲

Remarques 6.13.

- Les fonctions $\lfloor \log(n+1) \rfloor$, n, n^k, 2^n et $n!$ sont totalement-espace-constructibles.

- Si s_1 et s_2 sont totalement-espace-constructibles, alors $s_1(n)s_2(n)$, $2^{s_1(n)}$ et $s_1(n)^{s_2(n)}$ sont totalement-espace-constructibles.

▲

Théorème 6.14. Si s_2 est totalement-espace-constructible et si

$$\liminf_{n\to\infty} \frac{s_1(n)}{s_2(n)} = 0$$

alors $\mathsf{SPACE}(s_1(n)) \subset \mathsf{SPACE}(s_2(n))$. ■

Cela signifie, par exemple, que plus de langages peuvent être reconnus en espace $O(n^3)$ qu'en espace $O(n^2)$.

Théorème 6.15.

$$\mathsf{L} \subset \mathsf{PSPACE} \subset \mathsf{EXPSPACE}$$

■

Théorème 6.16.

$$\mathsf{L} \subseteq \mathsf{P}$$

Preuve. Si $A \in \mathsf{L}$ alors il existe une MT qui décide A en espace borné par $a \log n$ et donc qui est bornée en temps par

$$\begin{aligned} c^{a \log n} &= 2^{b \log n} \\ &= (2^{\log n})^b \\ &= n^b. \end{aligned}$$

■

Théorème 6.17.

$$\mathsf{P} \subseteq \mathsf{PSPACE}.$$

■

Corollaire 6.18.

$$\mathsf{L} \subset \mathsf{P} \ \text{ ou } \ \mathsf{P} \subset \mathsf{PSPACE}.$$

■

Exemples 6.19.

- $\{w \mid |w|_{\mathtt{a}} = |w|_{\mathtt{b}}\} \in \mathsf{L}$
- $\{w \mid w = w^{\mathrm{R}}\} \in \mathsf{L}$
- $\{\langle x, y, z\rangle \mid x, y, z \in \mathbb{N} \text{ et } z = x + y\} \in \mathsf{L}$
- $\{\langle x, y, z\rangle \mid x, y, z \in \mathbb{N} \text{ et } z = xy\} \in \mathsf{L}$

▲

Définition 6.20. Une expression booléenne quantifiée (EBQ) est une expression de la forme $Q_1 x_1\, Q_2 x_2\, \ldots\, Q_n x_n\ :\ E$ où :

- les x_i sont des variables booléennes ;
- les Q_i sont des quantificateurs (*pour tout* $\forall$ ou *il existe* $\exists$) ;
- E est une expression booléenne ;
- toutes les variables apparaissant dans E doivent être quantifiées.

▲

Définition 6.21.

$$QBF = \{\langle E \rangle \mid E \text{ est une EBQ dont la valeur est } 1\}$$

▲

Exemple 6.22. L'expression $\exists x \, \forall y \, \exists z \ : \ (x \vee y) \wedge (y \vee z)$ est une EBQ qui appartient à QBF car :

- on choisit $x = \texttt{1}$;
- l'expression devient : $\forall y \, \exists z \ : \ y \vee z$;
- si $y = \texttt{0}$, on choisit $z = \texttt{1}$;
- si $y = \texttt{1}$, on choisit $z = \texttt{1}$;
- dans les deux cas l'expression donne 1.

▲

Exemple 6.23. L'expression $\forall x \, \forall y \, \exists z \ : \ (x \vee y) \wedge \neg z$ est une EBQ qui n'appartient pas à QBF car :

- si $x = \texttt{0}$ et $y = \texttt{0}$, l'expression devient $\exists z \ : \ \texttt{0}$;
- cette expression vaut $\texttt{0}$ quelle que soit la valeur choisie pour z.

▲

Théorème 6.24.

$$QBF \in \mathsf{PSPACE}$$

Aperçu de la preuve. On montre qu'on peut décider de l'appartenance d'une EBQ à QBF en espace polynômial à l'aide d'un algorithme récursif.

- Si l'EBQ ne possède qu'un seul quantificateur, alors l'expression a au plus une variable et on peut facilement calculer sa valeur en espace polynômial.

- Si l'EBQ contient deux quantificateurs ou plus, alors on considère deux cas :
  - Si l'EBQ commence par $\forall$, alors on donne successivement les valeurs `0` et `1` à la variable quantifiée par ce $\forall$, et on appelle récursivement la procédure de vérification deux fois. On retourne la conjonction des deux valeurs retournées par ces appels.

  - Si l'EBQ commence par $\exists$, alors on donne successivement les valeurs `0` et `1` à la variable quantifiée par ce $\exists$, et on appelle récursivement la procédure de vérification deux fois. On retourne la disjonction des deux valeurs retournées par ces appels.

La profondeur de la récursion est bornée par le nombre de variables, donc la taille de la pile et l'espace mémoire utilisé seront de tailles polynômiales. ■

Définition 6.25. Le langage L est PSPACE-complet si

- $L \in \mathsf{PSPACE}$;
- pour tout $L' \in \mathsf{PSPACE}$ on a $L' \leq_{\mathrm{P}} L$.

▲

Théorème 6.26.

QBF est PSPACE-complet.

■

Définition 6.27. Une expression régulière avec exposants (ERE) sur un alphabet Σ est récursivement définie comme suit :

- $\emptyset$ est une ERE (l'ensemble vide) ;
- ε est une ERE (le mot vide) ;
- si $x \in \Sigma$, alors x est une ERE (un symbole) ;
- si E est une ERE, alors (E) est une ERE (parenthésage) ;
- si E est une ERE, alors E^* est une ERE (fermeture de Kleene) ;
- si E est une ERE et si $n \in \mathbb{N}$, alors E^n est une ERE (exponentiation) ;
- si E_1 et E_2 sont des ERE, alors $E_1 E_2$ est une ERE (concaténation) ;
- si E_1 et E_2 sont des ERE, alors $E_1|E_2$ est une ERE (union).

Définition 6.28. Soit E une ERE sur l'alphabet Σ. Le langage engendré par E, $L(E)$, est l'ensemble des mots de Σ^* qu'on peut former en appliquant les opérations de E sur les symboles de Σ. La priorité des opérations est définie par l'ordre dans lequel elles apparaissent à la définition 6.27. ▲

Exemples 6.29. Si $\Sigma = \{\mathtt{a}, \mathtt{b}\}$:

- $L(\mathtt{a}^*\mathtt{b}) = \{\mathtt{b}, \mathtt{ab}, \mathtt{aab}, \ldots\}$
- $L((\mathtt{aa})^3) = \{\mathtt{aaaaaa}\}$
- $L((\mathtt{aa}|\mathtt{bb})^3) =$
  $\{\mathtt{aaaaaa}, \mathtt{aaaabb}, \mathtt{aabbaa}, \mathtt{aabbbb}, \mathtt{bbaaaa}, \mathtt{bbaabb}, \mathtt{bbbbaa}, \mathtt{bbbbbb}\}$
- $L(\mathtt{a}(\mathtt{a}|\mathtt{bb})^2) = \{\mathtt{aaa}, \mathtt{aabb}, \mathtt{abba}, \mathtt{abbbb}\}$

▲

Définition 6.30.

$$TOUT_{\text{ERE}} = \{\langle E \rangle \mid E \text{ est une ERE et } L(E) = \Sigma^*\}$$

▲

Exemple 6.31.

$$\langle (\mathtt{a}|\mathtt{b}|\varepsilon)^*((\mathtt{aa})^3|\varepsilon|\mathtt{a}|\mathtt{b}|\mathtt{aa}|\mathtt{ab}|\mathtt{ba}|\mathtt{bb})\rangle \in TOUT_{\mathrm{ERE}}$$

▲

Théorème 6.32.

$$TOUT_{\mathrm{ERE}} \in \mathsf{EXPSPACE}.$$

■

Définition 6.33. Le langage L est EXPSPACE-complet si

- $L \in$ EXPSPACE ;
- pour tout $L' \in$ EXPSPACE on a $L' \leq_{\mathrm{P}} L$.

▲

Théorème 6.34.

$\overline{TOUT_{\mathrm{ERE}}}$ est EXPSPACE-complet.

■

Théorème 6.35.

$$\mathsf{PSPACE} \subset \mathsf{EXPSPACE}.$$

■

Corollaire 6.36.

$$TOUT_{\mathrm{ERE}} \notin \mathsf{P}.$$

■

Preuves interactives

Nous savons que la classe NP contient les langages dont les mots possèdent des preuves d'appartenance courtes.

~~La classe co-NP est la famille des langages dont aucun mot ne possède une preuve d'appartenance courte.~~

Définition 6.37.

$$\text{co-NP} = \{L \mid \overline{L} \in \text{NP}\}.$$

▲

Considérons qu'Arthur est une machine possédant une puissance de calcul raisonnable (temps polynômial) et que Merlin est un oracle tout-puissant. De plus, Arthur ne fait pas confiance à Merlin.

Pour $L \in \mathsf{NP}$, Merlin peut facilement convaincre Arthur que $w \in L$ si cela est vrai, et on voit facilement que Merlin ne peut pas tromper Arthur.

Est-il possible pour Merlin de convaincre Arthur que $w \in L$ pour $L \in \mathsf{co\text{-}NP}$?

Quelle est la classe des problèmes qui possèdent une *preuve interactive* ?

Définition 6.38. La classe de complexité IP est l'ensemble des langages L tels que $\exists V, P : \forall P', w$:

- V (le *vérificateur*) est une MT probabiliste de temps polynômial ;
- P (le *prouveur*) est une MT ;
- V et P interagissent en échangeant un nombre polynômiaux de messages ;
- si $w \in L$ alors $\Pr[\langle V, P\rangle \text{ accepte } w] \geq \frac{2}{3}$;
- si $w \notin L$ alors $\Pr[\langle V, P'\rangle \text{ accepte } w] \leq \frac{1}{3}$.

▲

Définition 6.39.

$$ISO = \{\langle G_1, G_2\rangle \mid G_1 \text{ et } G_2 \text{ sont des graphes isomorphes}\}$$

▲

Le langage ISO est clairement dans NP. On ne sait pas s'il est NP-complet. Son complément est par définition dans co-NP et on ne sait pas le mettre dans NP, c'est-à-dire qu'on ne connaît pas en général de preuve courte de l'énoncé "G_1 et G_2 ne sont pas isomorphes".

Théorème 6.40.

$$\overline{ISO} \in \mathsf{IP}$$

Aperçu de la preuve. Voici l'algorithme du vérificateur :

Faire deux fois :

- Choisir au hasard $G \in \{G_1, G_2\}$.
- Renommer de façon aléatoire les noeuds de G.
- Demander au prouveur de trouver à quel graphe (G_1 ou G_2) G est isomorphe. Si le prouveur donne la bonne réponse deux fois, alors on accepte sinon on rejette.

Si $\langle G_1, G_2 \rangle$ est dans $\overline{ISO}$ alors les deux graphes ne sont pas isomorphes. Le prouveur peut calculer à quel graphe le graphe G donné par le vérificateur est isomorphe et donner la bonne réponse. Il existe donc un prouveur qui fait accepter le vérificateur avec probabilité $1 \geq \frac{2}{3}$.

D'autre part, si $\langle G_1, G_2 \rangle$ n'est pas dans $\overline{ISO}$ alors les deux graphes sont isomorphes. Le graphe G produit par le prouveur est donc isomorphe à G_1 et à G_2. Comme il a été produit aléatoirement, il est impossible pour le prouveur de savoir de quel graphe il s'agit et donc au mieux il peut essayer de deviner. Il réussira les deux tests avec probabilité $\frac{1}{4} \leq \frac{1}{3}$. ■

Théorème 6.41.

$$\mathsf{IP} = \mathsf{PSPACE}$$

■

Complexité parallèle

Nous allons étudier les problèmes ayant une solution parallélisable.

Avec un nombre de processeurs qui croît exponentiellement avec la taille de l'input on peut résoudre tous les problèmes dans NP en temps polynômial. Nous n'allons donc considérer qu'un nombre polynômial de processeurs.

Le modèle de parallélisme le plus simple est celui des *familles uniformes de circuits*, mais les résultats que nous verrons s'appliquent aussi à des machines munies de multiprocesseurs programmées en langage évolué.

Plusieurs notions d'*uniformité* existent. Nous allons nous concentrer sur celle qui est la plus utilisée.

Définition 6.42. Une famille uniforme de circuits est un ensemble de circuits $\{C_n \ : \ n \geq 1\}$, où n donne le nombre de bits d'entrée, tel qu'il existe une fonction calculable passant de 1^n à $\langle C_n \rangle$ qui prend un espace de taille logarithmique en n. ▲

Définition 6.43. La taille d'un circuit est le nombre de ses portes. ▲

Définition 6.44. La profondeur d'un circuit est le nombre maximal de portes traversées en considérant tous les chemins partant d'un bit d'entrée, suivant les arêtes du circuit, et finissant à un bit de sortie. ▲

Définition 6.45. Un langage (une fonction) est dans $\mathsf{NC}(k)$ s'il est décidé (elle est calculée) par une famille uniforme de circuits $\{C_n\}$ de taille polynômiale en n et de profondeur dans $O(\log^k n)$. ▲

Définition 6.46.

$$\mathsf{NC} = \bigcup_{k \geq 1} \mathsf{NC}(k).$$

▲

La classe NC est donc la classe des langages que l'on peut décider en temps polylogarithmique si l'on dispose d'un nombre polynômial de processeurs.

Une famille de circuits de profondeur $O(\log n)$ décide un langage qui est nécessairement dans NC, mais ce n'est pas nécessairement le cas d'une famille de circuits de profondeur $O(\log^2 n)$.

La taille du circuit est dans

$$O(2^{\log^2 n}) = O(n^{\log n}) \not\subseteq O(p(n)).$$

Théorème: NC(1) est inclus dans L.

Si L est dans NC(1) alors il existe une machine de Turing dans L qui produit le circuit qui décide d'un langage en SPACE(log(n)). La circuit peut être évalué par une fouille en profondeur qui nécessite de ranger le chemin et les résultats partiels du chemin. Puisque le chemin a longueur log(n), SPACE(log(n)) est suffiusant.l

Théorème 6.47.

$$L \subseteq NC \subseteq P$$

■

Exemples 6.48. Langages ou fonctions qui sont dans NC :

- tout ce qui est dans L ;
- $\{\langle G, s, t\rangle \mid$ il existe un chemin entre s et t dans le graphe orienté $G\}$;
- l'addition, la multiplication et la division binaires ;
- trier une liste d'entiers.

▲