TD 4 – Machines de Turing, hiérachie en temps, temps polynomial

Exercice 1.

1. Décrire une machine de Turing reconnaissant le langage

$$\{m \in \{a,b\}^* \mid \ m \text{ contient autant de } a \text{ que de } b\}$$

2. Décrire une machine de Turing reconnaissant l'ensemble des palindromes sur $\{a,b\}$.

Solution de l'exercice 1. On peut se contenter d'une machine plaçant un symbole au départ sur son ruban de travail puis déplaçant la tête de celui-ci à droite quand elle rencontre un a sur le ruban de lecture et à gauche quand elle rencontre un b, et en fin de mot qui teste si elle lit le symbole écrit au départ.
On peut aussi facilement utiliser un compteur qu'on incrémente.

Exercice 2. Montrer qu'une machine de Turing à $k > 1$ rubans de travail fonctionnant en temps t peut être simulée par une machine de Turing à un seul ruban de travail fonctionnant en temps $O(t^2)$. D'où vient la perte de temps ?

Solution de l'exercice 2. On utilise la même technique que celle décrite dans le livre pour la machine universelle "simple", qui consiste à copier le contenu des k rubans de travail "colonne par colonne". La nouvelle machine doit alors parcourir tout son ruban de travail pour simuler une étape, d'où l'augmentation quadratique du temps.

Exercice 3. Expliquer comment il est possible de simuler efficacement le calcul d'une machine de Turing utilisant des rubans bi-infinis sur une machine de Turing utilisant des rubans semi-infinis.

Solution de l'exercice 3. Utiliser deux rubans semi-infinis pour simuler un ruban bi-infini. Chaque pas de calcul de la machine initiale est simulé par un nombre constant de pas de la nouvelle machine.
Prenons le cas d'une machine à un seul ruban. Une transition du type $\delta(q,a) = (q', a', \leftarrow)$ est remplacée par les transitions suivantes :

— $\delta(q, \#, a) = (q', \#, a', \downarrow, \rightarrow)$;
— $\delta(q, a, \#) = (\tilde{q}, a', \#, \leftarrow, \downarrow)$ où $\tilde{q}$ est un nouvel état ;
— $\delta(\tilde{q}, \alpha, \#) = (q', \alpha, \#, \downarrow, \downarrow)$ pour tout $\alpha \neq \#$;
— $\delta(\tilde{q}, \#, \#) = (q', \#, \#, \downarrow, \rightarrow)$.

Les transitions vers la droite sont simulées de manière symétrique.

Exercice 4.

1. Montrer que si, pour tout $n \in \mathbb{N}$, $f(n) \leqslant g(n)$, alors $\mathrm{DTIME}(f(n)) \subseteq \mathrm{DTIME}(g(n))$.
2. Montrer que s'il existe $N \in \mathbb{N}$ tel que, pour tout $n \geqslant N$, $f(n) \leqslant g(n)$ et si, pour tout $n \in \mathbb{N}$, $g(n) \neq 0$, alors $\mathrm{DTIME}(f(n)) \subseteq \mathrm{DTIME}(g(n))$.
3. Montrer que si, pour tout n, $t(n) \geqslant n$, alors $\mathrm{DTIME}(t(n))$ est clos par union finie, intersection finie et complémentaire.

Exercice 5. Montrer que les fonctions ci-dessous sont constructibles en temps :

1. $t(n) = c$,
2. $t(n) = n$,
3. $t(n) = 2^{n^c}$,
4. Montrer que si t_1 et t_2 sont constructibles en temps, alors il en est de même pour $t_1 + t_2$ et $t_1 t_2$.
5. Montrer qu'une fonction f constructible en temps et telle que $f(n) = o(n)$ est ultimement constante.

Exercice 6. Théorème de la lacune (Trakhtenbrot).
Le but de cet exercice est de montrer qu'il existe une fonction calculable $f : \mathbb{N} \to \mathbb{N}$ telle que $f(n) \geqslant n$ et $\mathrm{DTIME}(f(n)) = \mathrm{DTIME}(2^{f(n)})$.

1. Comparer avec le théorème de hiérarchie en temps vu en cours.

2. On considère $(M_i)_{i\in\mathbb{N}}$ une énumération des machines de Turing. Montrer qu'il suffit de construire une fonction $f : \mathbb{N} \to \mathbb{N}$ vérifiant la propriété suivante : pour tout i, aucune des machines $M_0, \dots, M_i$ ne s'arrête sur aucune entrée de taille i en un temps compris entre $f(i)$ et $i2^{f(i)}$.
3. Construire une fonction calculable f qui vérifie la propriété ci-dessus. Conclure.
4. Existe-t-il une fonction f calculable telle que $\mathrm{DTIME}(f(n)) = \mathrm{DTIME}\left(2^{2^{2^{f(n)}}}\right)$?

Solution de l'exercice 6.

1. On a enlevé l'hypothèse de constructibilité en temps et le théorème de hiérarchie n'est plus valable.
2. Supposons que $L \in \mathrm{DTIME}(2^{f(n)})$. Ceci est attesté par une machine M_ℓ qui s'arrête sur tout entrée x en temps au plus $c2^{f(|x|)}$ pour une constante $c > 0$. Mais pour tout x tel que $|x| > \ell$, la machine M_ℓ s'arrête donc en temps au plus $f(|x|)$ par la propriété ci-dessus. On en déduit que $L \in \mathrm{DTIME}(f(n))$.
3. Pour un i fixé, on va définir $f(i)$ avec la bonne propriété. Soit k la suite définie par $k_0 = i$ et $k_{n+1} = k_n 2^{k_n}$ pour tout n. Pour chaque couple $(x, j) \in \{0,1\}^i \times \{0, \dots, i\}$, la machine M_j ne s'arrête pas ou s'arrête en un temps que l'on note $t(x, j)$. Soit $T = \{t(x, j) \mid (x, j) \in \{0,1\}^i \times \{0, \dots, i\}\}$. L'ensemble T est fini donc il existe $p \in \mathbb{N}$ tel que $]k_p, k_{p+1}] \cap T = \emptyset$. On vérifie facilement que le plus petit p vérifiant cette propriété est calculable. Poser $f(i) = k_p$ convient et f est calculable également.
4. Oui, même méthode.

Exercice 7. Montrer que les problèmes suivants sont dans P.

1. L'ensemble des graphes connexes.
2. L'ensemble des arbres.

Solution de l'exercice 7.

1. Supposons que le graphe est donné par sa matrice d'adjacence sous la forme

$$\#a_{1,1} \dots a_{1,n} \# a_{2,1} \dots a_{2,n} \# \dots \# a_{n_1} \dots a_{n,n} \#.$$

Nous appelerons les sommets $\{1, \dots, n\}$. Pour tester la connexité, on va effectuer un parcours en largeur pour calculer tous les sommets accessibles depuis le sommet 1, puis tester si cet ensemble est bien l'ensemble de tous les sommets.
Voyons comment implémenter ceci sur une machine de Turing. Nous allons utiliser quatre rubans de travail.
— Le premier ruban (R1) contient $\varepsilon_1 \dots \varepsilon_n$ où $\varepsilon_i = 1$ ssi le sommets i a été atteint par le sommet 1.
— Le deuxième ruban (R2) contient la file contenant les sommets à traiter de gauche à droite

$$\#v_1 \# v_2 \# \dots \# v_p \#.$$

— Les deux rubans suivants (R3 et R4) contiennent des compteurs permettant de mémoriser la ligne et la colonne courante lorsqu'on lit la matrice sur le ruban d'entrée.
Initialement, R1 contient 10^{n-1} et R2 contient #1#. Tant que la file n'est pas vide, on supprime le premier élément i de la file puis, par un balayage sur la matrice, on se rend à la i-ème ligne de la matrice (en utilisant le compteur sur R3). Tous les voisins qui n'avaient pas été visités sont marqués comme vus sur R1 et on les ajoute à la file sur R2. Les numéros de ces sommets sont gardés dans le compteur R4 au cours du balayage de la ligne i de la matrice. Une telle itération prend un temps linéaire en la taille de l'entrée. La file est vide après n itérations au plus. Vérifier que tous les sommets ont été vus se fait alors en temps linéaire (il faut vérifier que R2 contient 1^n). En tous l'algorithme prend en temps $O(n^3)$. L'algorithme est bien polynomial en la taille de l'entrée.
2. Un graphe à n sommets est un arbre ssi son nombre d'arêtes est égal à $n-1$ et s'il est connexe. On peut donc tester la connexité en temps polynomial avec l'algorithme de la première question puis compter le nombre d'arêtes en balayant la matrice (ceci se fait en temps linéaire en la taille de l'entrée).

Exercice 8. On considère le langage $L = \{1^n \mid n \text{ premier}\}$, i.e. l'ensemble des mots qui sont une suite de 1 dont la longueur est un nombre premier. Montrer que L appartient à P.

Solution de l'exercice 8. Un algorithme naïf pour tester si un nombre est premier consiste à tester la divisibilité par tous les nombres plus petits (ou jusqu'à la racine carrée). Si le nombre est codé par n bits il y a $O(2^n)$ d'entiers à essayer, mais si le nombre est codé en unaire, ce qui est le cas ici, le nombre d'entiers est borné par la longueur, ce qui est le nombre lui-même. On doit donc répéter un nombre linéaire de fois un test de divisibilité, celui-ci pouvant se faire en unaire en un nombre linéaire d'étapes.

Exercice 9. Montrer que les problèmes suivants sont dans P.

1. DNF-Sat, l'ensemble des (codes de) formules propositionnelles sous forme normale disjonctive qui sont satisfaisables. (Rappel : une formule propositionnelle est sous forme normale disjonctive si elle est de la forme $\bigvee_i \bigwedge_j \ell_{i,j}$, où les $\ell_{i,j}$ sont des variables ou des négations de variables).
2. CNF-Taut, l'ensemble des (codes de) formules propositionnelles sous forme normale conjonctive qui sont tout le temps vraies (quelle que soit l'affectation).

Solution de l'exercice 9.

1. Une disjonction de termes est satisfaisable ssi l'un des termes est satisfaisable, ce qui est possible ssi il ne contient pas une variable et sa négation. Il suffit donc de tester si cela se produit ou non.
2. Une formule propositionnelle est une tautologie ssi sa négation est tout le temps fausse, ce qui veut dire que sa négation n'est pas satisfaisable. Comme la négation d'une formule sous forme normale conjonctive est facilement équivalente à une formule sous forme normale disjonctive, ceci donne une réduction de CNF-Taut à DNF-Sat.

Exercice 10. Clauses de Horn.

1. Une clause de Horn est une disjonction de littéraux contenant au plus un littéral positif (par exemple $\neg x \vee \neg y \vee z$, ou x, ou $\neg x \vee \neg y$, mais pas $x \vee y$). Le problème HornSat est de déterminer, sur la donnée d'un ensemble de clauses de Horn, si leur conjonction est satisfaisable. Montrer que HornSat est dans P. (On pourra considérer l'opération suivante : trouver une clause réduite à un seul littéral, celui-ci doit alors prendre la valeur vraie, et on peut donc modifier les autres clauses qui contiendraient ce littéral ou sa négation. Étudier ce qui arrive lorsqu'on itère cette opération.)
2. DualHornSat est le problème de déterminer la satisfaisabilité d'un ensemble de clauses où chaque clause contient au plus un littéral négatif. Montrer que DualHornSat est dans P.

Exercice 11. Soit Det le problème de déterminer, sur la donnée d'une matrice carré d'entiers A et d'un entier i, la valeur du i-ième bit de $\det(A)$.

1. Si on a une borne n sur les dimensions de A et m sur le nombre de bits de chaque coefficient entier de la matrice A, donner une borne sur le nombre de bits de $\det(A)$ qui soit polynomiale en n et m.
2. Expliquer pourquoi le problème Det appartient à P.

Un graphe biparti est un graphe $G = (V_1 \dot\cup V_2, E)$, où E est un sous-ensemble de $V_1 \times V_2$ appelé l'ensemble des arcs de G. Supposons que $V_1 = V_2 = \{1, \dots, n\}$. On associe à G une matrice A de dimention $n \times n$:

$$A_{i,j} = \begin{cases} x_{i,j} & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

où les $x_{i,j}$ sont des variables. Le déterminant de A est alors un polynôme en les variables $x_{i,j}$. Un couplage parfait de G est un sous-ensemble C de E tel que tout sommet de $V_1 \dot\cup V_2$ appartienne à exactement un seul arc de C.

3. Montrer que G a au moins un couplage parfait ssi $\det A$ n'est pas le polynôme identiquement nul.
4. Peut-on en déduire un algorithme fonctionnant en temps polynomial et déterminant si un graphe biparti a un couplage parfait ?

Solution de l'exercice 11.

1. En utilisant la formule pour le déterminant comme somme sur les permutations, on voit que la valeur absolue de celui-ci est majorée par $n!2^{mn}$, soit un nombre qui peut s'écrire avec $O(n^2 m)$.
2. On peut utiliser un pivot de Gauss pour calculer le déterminant, ou des méthodes sans division (Mahajan & Vinay). Dans le cas du pivot de Gauss, il faut se convaincre que l'on peut écrire les résultats intermédiaires, c'est-à-dire qu'en faisant à chaque étape les simplifications ceux-ci restent de longueur polynomiale en bits. Sans donner trop de détails, une façon de faire est d'observer que les coefficients apparaissant sur la diagonale lors du pivot de Gauss correspondent à des sous-déterminants de la matrice originale.
3. Les monômes du déterminant ainsi obtenus correspondent exactement aux couplages parfaits : le polynôme n'est pas le polynôme nul ssi il existe au moins un tel couplage.
4. Il nous reste donc à tester si le polynôme est le polynôme nul. Cela ne semble malheureusement pas facile, donc la solution souvent utilisée est de faire un test avec possibilité d'erreur, en utilisant un lemme qui nous garantit que si on teste notre polynômes en prenant au hasard les valeurs des variables dans

un ensemble assez grand on aura une chance importante de trouver une valeur non-nulle si le polynôme était non-nul.

Exercice 12.
Soit X un problème dans P et Y un problème non trivial (i.e. non vide et différent de Σ^* tout entier). Montrer que X se réduit à Y pour la réduction many-one en temps polynomial.

Solution de l'exercice 12. Soit $X \in$ P. Soit Y non trivial. $Y \neq \varnothing$ donc $\exists u \in \{0,1\}^*, u \in Y$ et $Y \neq \Sigma^*$ donc $\exists v \in \{0,1\}^*, v \notin Y$.
On cherche une fonction f calculable en temps polynomial telle que quel que soit x dans $\{0,1\}^*$, $x \in X \iff f(x) \in Y$. $X \in$ P donc X est décidable en temps polynomial. f peut donc calculer en temps polynomial si $x \in X$ et renvoyer u si $x \in X$, et v sinon. X se réduit donc polynomialement à Y. Donc si on définit la P-complétude par : « A est P-complet ssi $A \in$ P et $\forall B \in P, B \leqslant_p A$ », cela ne donne pas une notion intéressante.