Université Bordeaux – Master Sciences et Technologies, Informatique. Session automne 2015

# Examen Automates et Complexité, 16 décembre 2015, 14h – 17h

Documents autorisés : notes de cours et de TD.
**La notation attachera une grande importance à la clarté et à la concision des justifications.**
Le barème est indicatif. Sauf mention contraire, les questions sont indépendantes.

**Exercice 1 — Applications directes du cours (5 points).** Répondez aux questions suivantes en **justifiant brièvement** vos réponses (toute réponse non justifiée vaut 0 points) :

1) Existe-t-il une réduction du problème SAT au problème de correspondance de Post ?

2) On considère le langage $L$ des codes des machines de Turing qui acceptent si et seulement si leur entrée représente un nombre premier. Le langage $L$ est-il dans la classe **NP** ?

3) Si $L$ est un langage décidable de mots, est-il vrai que tout langage $K$ tel que $K \subseteq L$ est aussi décidable ?

4) Soit $f : \Sigma^* \to \Sigma^*$ une fonction calculable. Est-il vrai que pour tout langage semi-décidable, $L \subseteq \Sigma^*$, le langage $f(L)$ est également semi-décidable ?

5) Soit $f : \Sigma^* \to \Sigma^*$ une fonction calculable. Est-il vrai que pour tout langage décidable, $L \subseteq \Sigma^*$, le langage $f(L)$ est également décidable ? ∎

**Solution.**

1) **Oui**. La question est similaire à la question 1.*d*) du DS 1 et l'argument est exactement le même : soit $p = \{(a, a)\}$ et $n = \{(a, b)\}$ : $p$ est une instance positive du problème de correspondance de Post PCP et $n$ en est une instance négative. Soit $f$ la fonction qui a une formule $\varphi$ associe $p$ si $\varphi$ est satisfiable et $n$ sinon. Comme SAT est décidable, cette fonction est calculable. Par définition, $f(\varphi)$ est une instance positive de PCP si et seulement si $\varphi$ est une instance positive de SAT, donc $f$ est une réduction.
2) **Non.** La question est similaire à la question 1.*b*) du DS 1 et l'argument est exactement le même. La propriété « *être un nombre premier* » n'est pas triviale, donc $L$ est indécidable d'après le théorème de Rice, et donc n'est pas dans **NP**. À nouveau, la question n'était pas de savoir si le problème PRIME est ou non dans **NP**.
3) **Non.** Une réponse 'Oui' impliquerait que tout langage serait décidable. En effet, le langage de tous les mots est décidable (par une machine qui répond toujours oui), et ce langage contient tout autre langage.
4) **Oui.** Une machine $M$ qui accepte si et seulement si son entrée $x$ est dans $f(L)$ est la suivante : à l'étape $k$, $M$ génère tous les mots de longueur au plus $k$ et pour chacun d'eux, lance pendant $k$ pas de calcul la machine $M_L$ qui accepte uniquement si son entrée est dans $L$. Puis, $M$ calcule l'image par $f$ de chacun de ces mots sur lesquels $M_L$ a accepté (donc détectés dans $L$), et si l'une de ces images est l'entrée $x$, alors $M$ accepte. Par définition, si $M$ accepte $x$, c'est qu'il existe un mot de $L$ dont l'image par $f$ est $x$. Inversement, si $x$ est dans $f(L)$, il existe un mot $y$ de $L$ tel que $f(y) = x$, et ce mot sera accepté par $M$ à l'étape $\max(|y|, k)$ où $k$ est le nombre de pas nécessaires à $M_L$ pour accepter $y$.
5) **Non.** Il a été vu en TD (feuille 3, exercice 5) qu'on peut ajouter à tout $X \subseteq \mathbb{N}$ semi-décidable une composante pour obtenir un langage $Z \subseteq \mathbb{N} \times \mathbb{N}$ décidable. Pour un tel $X$ semi-décidable mais indécidable, la fonction $f : \mathbb{N}^2 \to \mathbb{N}^2$ qui à $(x, y)$ associe $(x, 0)$ est donc telle que $f(Z) = X \times \{0\}$ est indécidable. Il reste à représenter cette fonction sur $\Sigma^*$ plutôt que sur $\mathbb{N}^2$, ce qui se fait par composition avec une fonction caculable.

**Exercice 2 — NP-Complétude (6 points).** Dans cet exercice, on suppose que $\mathbf{P} \neq \mathbf{NP}$. Pour chacun des problèmes suivants, dites si il est **NP**-complet ou dans **P**. Attention, les réponses doivent être **prouvées**. Si vous répondez qu'un problème est **NP**-complet vous devez prouver qu'il est dans **NP** *et* qu'il est **NP**-difficile. Si vous répondez qu'un problème est dans **P** vous devez donner un algorithme polynomial qui le résout.

*Remarque : toutes les réductions sont faciles, la principale difficulté est de trouver le bon problème à réduire.*

1) CHEMIN QUASI-HAMILTONIEN
   **ENTRÉE** : Un graphe orienté $G$.
   **QUESTION** : Le graphe $G$ possède-t-il un chemin qui visite chaque sommet au moins une fois et au plus deux fois ?

On rappelle que dans une formule propositionnelle, un *littéral* est une variable ou une négation de variable. Un littéral est *positif* si c'est une variable, *négatif* si c'est une négation de variable.

2) Sat Modifié

Une *formule 4-CNF* est une conjonction de clauses, chacune du type $(\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$, où chaque $\ell_i$ est un littéral. Par exemple, $(x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_5 \vee \neg x_7)$ est une formule 4-CNF.

**ENTRÉE** : Une formule 4-CNF.
**QUESTION** : Y a-t-il une affectation des variables pour laquelle chaque clause a un littéral vrai et un littéral faux ?

3) Horn-Sat

Une *clause de Horn* est une disjonction de littéraux qui contient au plus un littéral positif et un nombre arbitraire de littéraux négatifs. Par exemple, "$x$", "$\neg y \vee \neg x$" et "$\neg x \vee \neg y \vee z$" sont des clauses de Horn. Inversement, "$x \vee y \vee \neg z$" n'est **pas** une clause de Horn car elle contient deux littéraux positifs, $x$ et $y$.

**ENTRÉE** : Une formule $\varphi$ qui est une conjonction de clauses de Horn.
**QUESTION** : La formule $\varphi$ est-elle satisfiable ?

4) Couverture par ensemble

**ENTRÉE** : Une liste d'ensembles finis d'entiers $E_1, \ldots, E_k$ et un nombre entier $n$.
**QUESTION** : Existe-t-il un ensemble d'entiers $E$ à $n$ éléments tels que pour tout $i = 1, \ldots, k$, on a $E \cap E_i \neq \emptyset$ ? ∎

**Solution.**

1) Le problème Chemin Quasi-Hamiltonien est dans **NP** : on devine une suite de $2n$ sommets au plus (où $n$ est le nombre de sommets du graphe d'entrée) et on vérifie en temps polynomial qu'il est quasi-Hamiltonien.

Pour montrer que ce problème est **NP**-complet, il reste à réduire un problème déjà connu **NP**-complet à Chemin Quasi-Hamiltonien. On choisit Chemin Hamiltonien. On construit en temps polynomial une entrée $G'$ de Chemin Quasi-Hamiltonien à partir d'une entrée $G$ de Chemin Quasi-Hamiltonien : soient $u_1, \ldots, u_n$ les sommets de $G$. Pour construire $G'$, on ajoute à $G$ $n$ sommets $u'_1, \ldots, u'_n$, où la seule arête incidente à $u'_i$ le relie à à $u_i$. Le graphe $G'$ se construit bien à partir de $G$ en temps polynomial. De plus, $G$ a un chemin Hamiltonien si et seulement si $G'$ a un chemin quasi-Hamiltonien : si $G$ a un chemin Hamiltonien, par exemple $u_1 - u_2 - \cdots - u_n$, alors $G'$ a le chemin quasi-Hamiltonien $u_1 - u'_1 - u_1 - u_2 - u'_2 - u_2 \cdots - u_n - u'_n - u_n$. Inversement, si $G'$ a un chemin quasi-Hamiltonien, chaque sommet $u'_i$ ne peut être atteint qu'en passant par $u_i$. On construit un chemin Hamiltonien de $G$ en supprimant les sommets $u'_i$ du chemin de $G'$ (et en ne considérant qu'un passage par chaque sommet original de $G$).

Remarque : cette preuve suppose les graphes non orientés mais s'adapte facilement pour les graphes orientés : au lieu de relier $u_i$ à $u'_i$ par une arête, on relie ces sommets par 2 arcs $u_i \to u'_i \to u_i$.

2) Sat Modifié est **NP**-complet. D'une part, il est dans **NP** : on devine une affectation des variables et on teste en temps polynomial qu'elle rend $\varphi$ vraie. Pour montrer qu'il est **NP**-complet, on réduit 3-Sat à Sat Modifié. Soit $\varphi$ une instance de 3-Sat. On construit une instance $\varphi'$ de Sat Modifié : on ajoute une nouvelle variable $t$, les clauses de $\varphi'$ sont alors celles du type $(\ell_1 \vee \ell_2 \vee \ell_3 \vee t)$, où $(\ell_1 \vee \ell_2 \vee \ell_3)$ est une clause de $\varphi$. On doit montrer que $\varphi$ est satisfiable si et seulement si on peut trouver une affectation des variables de $\varphi'$ pour laquelle chaque clause a un littéral vrai et un littéral faux. Si $\varphi$ est satisfiable, on étend une affectation rendant $\varphi$ vraie en choisissant $t$ faux : l'affectation obtenue convient. Inversement, supposons qu'il existe une affectation des variables de $\varphi'$ pour laquelle chaque clause a un littéral vrai et un littéral faux. Si cette affectation évalue $t$ à faux, c'est que dans chaque clause de $\varphi'$, l'un des littéraux s'évalue à vrai. La restriction de l'affectation aux variables de $\varphi$ rend donc $\varphi$ vraie, $\varphi$ est donc satisfiable. Si cette affectation évalue $t$ à vrai, c'est que dans chaque clause de $\varphi'$, l'un des littéraux s'évalue à faux. La restriction de la **négation** de l'affectation aux variables de $\varphi$ rend donc $\varphi$ vraie, à nouveau $\varphi$ est donc satisfiable.

3) Le problème est dans **P** (cf. question 2 du sujet de programmation). Si chaque clause contient au moins un littéral négatif, il suffit d'affecter chaque variable à faux. Sinon, construit le plus petit ensemble de variables auxquelles on **doit** affecter vrai pour rendre la formule vraie. On initialise cette liste en y mettant les variables qui apparaissent seules et positivement dans une clause. On répète ensuite la boucle suivante : pour chaque clause $x \vee \neg x_1 \vee \ldots \vee \neg x_p$, si $x_1, \ldots, x_p$ sont déjà affectées à vrai, on ajoute $x$ à la liste de ces variables. On s'arrête lorsque la liste des variables devant être affectées à vrai n'évolue plus. Il reste à vérifier qu'aucune négation des variables de cette liste ne se trouve seule dans une clause.

À chaque étape de construction de la liste sauf la dernière, la liste est augmentée d'au moins une variable. Le nombre d'étapes est donc $O(n)$, où $n$ est le nombre de variables. On en déduit que l'algorithme est bien polynomial.

4) Le problème Couverture par ensemble est **NP**-complet. Pour montrer qu'il est dans **NP**, il faut adapter l'idée naïve de deviner $n$ entiers : elle ne fonctionne pas directement car la valeur de $n$ est exponentielle par rapport à la taille de sa représentation. L'argument est le même que pour l'exercice 2.a du DS 2 : on compare en temps linéaire $n$ à $k$. Si $n \geqslant k$, la donnée est une instance positive et on répond donc oui. Sinon, $n < k$ et on peut utiliser l'idée naïve, puisque la taille de la donnée est au moins $k$. On devine donc $n$ entiers dont la somme des tailles est inférieure à la taille de l'entrée. On vérifie ensuite que $E \cap E_i \neq \emptyset$ pour chaque $i$, en parcourant $E$ et $E_i$, ce qui se fait en temps $O(n|E_i|) = O(k|E_i|)$. L'algorithme est donc bien dans **NP**.
Le fait qu'il est **NP**-complet s'obtient très facilement par réduction à partir du problème Couverture de sommets : à un graphe $G$ à $k$ arêtes $e_1, \dots, e_k$ et un entier $n$, on associe l'instance $e_1, \dots, e_k, n$ du problème Couverture par ensemble. Autrement dit, les ensembles que l'on considère sont les arêtes (qui sont chacune des ensembles de sommets, à 2 éléments). Par définition, $G$ a une couverture de $n$ sommets si et seulement si $e_1, \dots, e_k$ a une couverture $E$ de taille $n$.

**Exercice 3 — Machines à Compteurs (6 points).** Dans cet exercice, on présente un nouveau modèle de calcul : les *machines à compteurs*. Soit $k \geqslant 1$ un entier, une machine à $k$ compteurs est un tuple $M = (Q, q_0, q_a, q_r, \delta)$ où :

- $Q$ est un ensemble fini d'états.
- $q_0 \in Q$ est l'état initial.
- $q_a \in Q$ est l'état final acceptant.
- $q_r \in Q$ est l'état final rejetant.
- $\delta : Q \times \{Z, \neg Z\}^k \to Q \times \{-1, 0, 1\}^k$ est une fonction de transition, où $Z$ est un symbole (qui va permettre de tester si un compteur vaut 0).

Une transition est donc de la forme

$$\delta(q, (t_1, \dots, t_k)) = (q', (n_1, \dots, n_k))$$

où $q, q' \in Q$, chaque $t_i$ vaut soit $Z$ soit $\neg Z$, et où chaque $n_i$ vaut $-1$, 0 ou 1. On impose de plus la condition suivante pour toute telle transition :

$$\text{pour chaque } i \in \{1, \dots, k\}, \text{ si } t_i = Z, \text{ alors } n_i \neq -1. \qquad (\mathcal{C})$$

On va définir l'exécution d'une machine à compteurs $M$ quelconque sur une valeur d'entrée $m \in \mathbb{N}$ (remarquez que l'entrée d'une machine à compteurs est donc un entier). La machine manipule $k$ variables $c_1, \dots, c_k$, les « compteurs » de $M$, contenant des entiers positifs ou nuls. Une transition peut tester si chaque compteur vaut 0 ou non, et agir sur chaque compteur (en le décrémentant, ou en le laissant inchangé, ou en l'incrémentant).

On appelle *configuration* de $M$ un tuple $(q, c_1, \dots, c_k)$ où $q$ est un état dans $Q$ et les $c_i$ sont des entiers **positifs ou nuls**. Si $C = (q, c_1, \dots, c_k)$ et $D = (q', d_1, \dots, d_k)$ sont deux configurations de $M$, on dit que $M$ passe de $C$ à $D$ par $\delta$, noté $C \xrightarrow{\delta} D$, si $\delta(q, (t_1, \dots, t_k)) = (q', (n_1, \dots, n_k))$ avec

- pour chaque $i$, on a $t_i = Z$ si $c_i = 0$ et $t_i = \neg Z$ si $c_i \neq 0$.
- pour chaque $i$, on a $d_i = c_i + n_i$.

Par exemple, si $C = (q, 3, 0, 4, 0)$ et $\delta(q, (\neg Z, Z, \neg Z, Z)) = (q', (-1, 1, 1, 0))$, alors $C \xrightarrow{\delta} D$ avec $D = (q', 2, 1, 5, 0)$. Autrement dit, dans l'état $q$, la transition $\delta(q, (\neg Z, Z, \neg Z, Z)) = (q', (-1, 1, 1, 0))$ teste que $c_1$ et $c_3$ sont non nuls, que $c_2$ et $c_4$ sont nuls, puis enlève 1 à $c_1$, ajoute 1 à $c_2$ et à $c_3$, laisse $c_4$ inchangé, et change l'état à $q'$. La condition $(\mathcal{C})$ interdit de décrémenter un compteur nul, ce qui garantit que chaque compteur reste positif ou nul. L'exécution de $M$ sur l'entrée $m \in \mathbb{N}$ est maintenant définie de la façon suivante : c'est une suite de configurations $C_0, C_1, C_2, \dots$ (possiblement finie ou infinie, la machine peut ne pas s'arrêter) telles que :

- on part de la configuration $C_0 = (q_0, m, \underbrace{0, \dots, 0}_{k-1})$. C'est-à-dire qu'on commence dans l'état initial $q_0$ et avec les compteurs nuls, sauf le premier qui contient l'entrée $m$.
- pour tout $i$, $C_i \xrightarrow{\delta} C_{i+1}$.

L'exécution termine dans une configuration $C_n$ si celle-ci utilise un des deux états finaux ($q_a$ ou $q_r$). Dans ce cas l'exécution est acceptante si elle termine dans l'état $q_a$ et rejetante si elle termine dans l'état $q_r$. Le langage d'une machine à compteur $M$ est l'ensemble $L(M)$ des entiers $m \in \mathbb{N}$ tels que l'exécution de $M$ sur $m$ est acceptante.

**I : Exemples de Machines à Compteurs.**

Pour les questions 1 et 2, on demande à la fois une explication intuitive de la machine à compteur demandée et sa description précise (c'est-à-dire sa liste de transitions). Pour les questions suivantes, on demande seulement des explications.

1) Donner une machine à 1 compteur $M$ dont le langage $L(M)$ est l'ensemble des entiers naturels pairs.

On dit que la machine à $k$ compteurs $M$ calcule une fonction totale $f : \mathbb{N} \to \mathbb{N}$ lorsqu'elle s'arrête dans $q_a$ sur toute entrée $m \in \mathbb{N}$, avec $c_1 = f(m)$ lorsque la machine s'arrête. Autrement dit, quand la machine s'arrête, elle est dans l'état $q_a$ et la valeur du premier compteur est $f(m)$.

2) Donner une machine à deux compteurs qui calcule le quotient entier de son entrée par 2 : si $m = 2x$ ou $m = 2x + 1$ avec $x \in \mathbb{N}$, la machine calcule $x$.

3) Décrire une machine à deux compteurs qui calcule la fonction $f(m) = 2m$.

**II : Machines à Compteurs et Machines de Turing.**

4) Soit $k \geqslant 1$ quelconque. Montrer que toute machine à $k$ compteurs peut être simulée par une machine de Turing (à plusieurs bandes). En d'autres termes, on demande d'expliquer comment, à partir du code source d'une machine à $k$ compteurs, on peut construire une machine de Turing qui accepte le même langage.

5) Montrer que toute machine de Turing à 1 bande peut être simulée par une machine à 4 compteurs. En d'autres termes, on demande d'expliquer comment, à partir du code source d'une machine de Turing, on peut construire une machine à 4 compteurs qui accepte le même langage.

*Indication : on pourra s'inspirer du codage utilisé dans le DS2 pour coder une machine de Turing par une machine à piles.*

6) Montrer que le problème suivant est indécidable :

**ENTRÉE** : Une machine $M$ à $k \geqslant 4$ compteurs et $m \in \mathbb{N}$ une entrée pour $M$.
**QUESTION** : Est-ce que $m \in L(M)$ ?

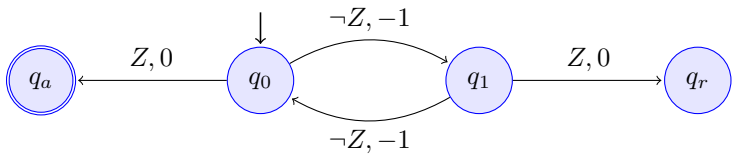
■

**Solution.**

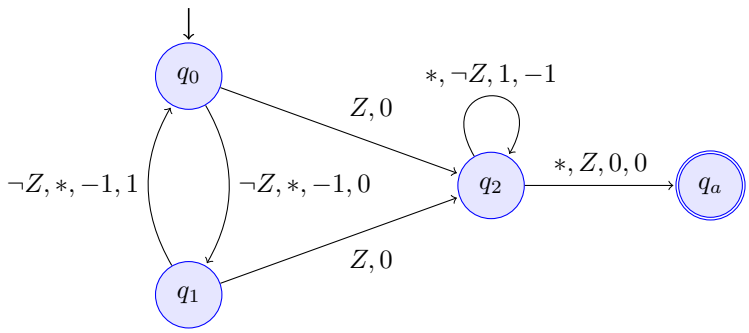
1) On implémente une boucle qui soustrait 2 à chaque passage. Cela conduit à la machine suivante :

$$\delta(q_0, \neg Z) = (q_1, -1), \quad \delta(q_1, \neg Z) = (q_0, -1), \quad \delta(q_0, Z) = (q_a, 0), \quad \text{et enfin } \delta(q_1, Z) = (q_r, 0).$$

On peut aussi représenter ces transitions de façon graphique, en mettant sur chaque transition à la fois la série de tests suivie de la série de mises à jour des compteurs (pour cet exemple : un test et une modification par transition, puisqu'on a un unique compteur). L'état initial a été représenté par une flèche entrante et l'état d'acceptation est doublement entouré :




2) C'est le même principe, à ceci près qu'on compte les tours de boucle dans le second compteur (transition de $q_1$ à $q_0$ dans la figure ci-dessous). On recopie le second compteur dans le premier une fois que celui-ci est nul. Cela est fait dans l'état $q_2$ : chaque transition enlève 1 au second compteur et ajoute 1 au premier compteur. Pour rendre le dessin plus lisible, on utilise $*$ pour exprimer qu'on ne teste pas le compteur correspondant (cela revient à mettre 2 transitions, l'une avec le test $Z$ et l'autre avec le test $\neg Z$).




3) À nouveau une boucle : tant que le premier compteur n'est pas nul, on le décrémente et on incrémente deux fois le second compteur. Une fois le premier compteur nul, on transfère le second compteur dans le premier.

4) Il suffit de simuler chaque compteur par une bande, en remarquant qu'on peut simuler chaque opération : tester si une bande contient l'entier nul, incrémenter ou décrémenter l'entier représenté sur une bande (les machines de Turing pour ces opérations ont été vues en TD). L'état de la machine à compteurs correspond à l'état de la machine de Turing, à ceci près qu'on utilise des états intermédiaires car une instruction de la machine à compteurs est simulée par une suite d'instructions de la machine de Turing. Dans l'état intermédiaire, on retient l'état courant de la machine à compteurs simulée, ainsi que l'étape dans laquelle on se trouve vis-à-vis de l'opération que l'on est en train d'implémenter (test à 0, incrément ou décrément).

5) Comme indiqué dans l'énoncé, l'idée est la même que dans le DS 2 : on utilise un compteur mémorisant l'entier représenté par la partie à gauche de la tête de lecture, lue de gauche à droite, et un compteur mémorisant l'entier représenté par la partie à droite de la tête de lecture, lue de droite à gauche. Par exemple, si le mot écrit sur la bande est 11011101 et que la tête de lecture est sur le 3ème caractère '1', l'un des compteurs contiendra 6 (représenté par 110, la partie à gauche de la tête) et l'autre 23 (représenté par 10111, la partie à droite de la tête, lue de droite à gauche, position de la tête comprise).

   Un problème mineur est qu'avec l'alphabet de travail $\{0, 1\}$, les '0' de part et d'autre de la bande ne seront pas pris en compte (parce que les 0 de tête ne changent pas la valeur du nombre représenté). Un autre problème est que la machine peut écrire des blancs à l'intérieur de sa bande. On contourne ces problèmes en modifiant la machine pour qu'à l'intérieur de sa bande, elle utilise un symbole ■ au lieu du symbole blanc □, puis on interprète la valeur de la bande en base 4, en choisissant comme représentation des 4 symboles □, ■, 0, 1 les chiffres $0, 1, 2, 3$. De cette façon, aucun symbole à excepté les blancs entourant le calcul n'est interprété comme le chiffre 0, et la valeur des compteurs donne, de façon non ambiguë, les parties gauche et droite de la bande. Les opérations de la machine de Turing s'implémentent alors soit comme une division par 4 (pour la partie de la bande qui diminue), soit par une multiplication par 4 et des additions de 1, 2 ou 3 (pour la partie de la bande qui augmente), ce qu'on implémente à partir des machines des questions 2 et 3.

**Exercice 4 — Formules Booléennes Quantifiées (5 points).** On suppose fixé un ensemble $\mathcal{V}$ de variables. Une formule booléenne quantifiée est définie par induction à partir des éléments suivants :

- *Valeurs de vérité* : "Vrai" et "Faux" sont des formules.
- *Variables* : pour toute variable $x \in \mathcal{V}$, "$x$" est une formule.
- *Connecteurs logiques* : si $\varphi$ et $\psi$ sont des formules, "$\varphi \vee \psi$", "$\varphi \wedge \psi$" et "$\neg\varphi$" sont des formules.
- *Quantifications* : si $\varphi$ est une formule et $x \in \mathcal{V}$ est une variable, alors "$\exists x\ \varphi$" et "$\forall x\ \varphi$" sont des formules.

Par exemple "$\forall x \exists y \exists z\ x \vee (\neg x \wedge y \wedge \neg z)$" est une formule booléenne quantifiée. On va se restreindre aux formules qui vérifient les propriétés suivantes :

- Nos formules n'ont *pas de variables libres* sauf dans la question 4 : si la formule contient la variable $x$, alors $x$ se trouve sous une quantification "$\exists x$" ou "$\forall x$".
- Nos formules *quantifient chaque variable une seule fois* : pour toute variable $x \in \mathcal{V}$, la formule contient une seule quantification "$\exists x$" ou "$\forall x$".
- Nos formules ont tous leurs quantifications en tête, c'est-à-dire sont de la forme $Q_1 x_1 Q_2 x_2 \dots Q_k x_k\ \varphi$ où chaque $Q_i$ est soit $\forall$, soit $\exists$, et $\varphi$ n'a pas de quantification.

La valeur de vérité d'une formule booléenne quantifiée est définie avec l'interprétation habituelle des connecteurs et des quantifications. Par exemple, la formule $\forall x\, \exists y\ (x \wedge y) \vee (\neg x \wedge \neg y)$ est vraie (pour $x$ Vrai, on choisit $y$ Vrai, et pour $x$ Faux, on choisit $y$ Faux). Par contre, la formule $\forall x\, \forall y\ (x \vee y)$ est fausse.

Le problème Formule Booléenne Quantifiée est le suivant :

**ENTRÉE** : Une formule Booléenne quantifiée $\varphi$.
**QUESTION** : Est-ce que $\varphi$ est vraie ?

1) Montrer que tout problème **NP** se réduit polynomialement à Formule Booléenne Quantifiée.
2) Montrer que le problème Formule Booléenne Quantifiée est dans la classe **PSPACE**. Autrement dit, il faut expliquer comment calculer la valeur d'une formule booléenne quantifiée en utilisant un espace polynomial, de façon déterministe.

Dans les questions 3 et 4, on considère un problème $A \in$ **PSPACE** et une machine $M_A$ qui résout $A$ et travaille en espace polynomial.

3) Évaluer la taille du graphe des configurations de $M_A$ en fonction de la taille $n$ de l'entrée.
4) (**Difficile**) On dit que deux configurations $C, D$ de $M_A$ ont la propriété $\mathcal{P}_n$ s'il existe un chemin de taille au maximum $2^n$ dans le graphe des configurations de $M_A$ allant de $C$ à $D$.

En représentant les configurations avec des variables (comme dans la preuve du théorème de Cook), expliquer comment écrire pour chaque entier $n$, une formule booléenne quantifiée $\varphi_n$ de taille polynomiale, ayant des variables libres, et qui est satisfiable si et seulement s'il existe deux configurations ayant la propriété $\mathcal{P}_n$ (commencer par $\varphi_0$ et calculer $\varphi_{n+1}$ en fonction de $\varphi_n$).

5) Utiliser les questions précédentes pour montrer que Formule Booléenne Quantifiée est **PSPACE**-complet.

6) Quelle relation entre **PSPACE** et **NPSPACE** peut-on déduire de la question 4 ? Justifier la réponse. ∎

**Solution.** On appelle FBQ le problème Formule Booléenne Quantifiée.

1) À partir d'une instance $\varphi$ de Sat sur les variables $x_1, \ldots, x_n$, on construit l'instance $\varphi' = \exists x_1 \ldots \exists x_n \varphi$ de FBQ. Par définition de ces problèmes, $\varphi$ est satisfiable si et seulement si $\varphi'$ est vraie. Par ailleurs, $\varphi'$ se construit en temps linéaire à partir de $\varphi$. On a montré que Sat se réduit polynomialement à FBQ.

   On sait par ailleurs que tout problème de **NP** se réduit au problème **NP**-complet Sat. Par transitivité de la relation de réduction polynomiale, tout problème de la classe **NP** se réduit polynomialement à FBQ.

2) L'algorithme naturel d'évaluation récursif de $Q_1 x_1 \varphi$, où $Q_1 \in \{\exists, \forall\}$, est de calculer
   - $\varphi(0)$, c'est-à-dire $\varphi$ dans laquelle on remplace $x_1$ par 0, et
   - $\varphi(1)$, c'est-à-dire $\varphi$ dans laquelle on remplace $x_1$ par 1.

   Le résultat de l'évaluation est $\varphi(0) \vee \varphi(1)$ si $Q_1 = \exists$, et $\varphi(0) \wedge \varphi(1)$ si $Q_1 = \forall$.

   Cet algorithme ne requiert qu'un espace polynomial : la profondeur de la pile d'appels récursifs est le nombre de quantificateurs, soit $O(n)$ où $n$ est la taille de $\varphi$. La taille de la pile est également en $O(n)$ : à chaque appel, on conserve la valeur choisie de chaque variable, la valeur de $\varphi(0)$ et celle de $\varphi(1)$. Enfin, lorsque chaque variable est fixée, évaluer la partie sans quantificateurs se fait en espace logarithmique. Le problème FBQ requiert donc en fait un espace linéaire.

3) Puisque $M_A$ travaille en espace polynomial, le contenu de chaque bande est par définition de taille polynomiale. En ajoutant l'état et la position des têtes de lecture, on peut donc représenter une configuration par un mot de taille $p(n)$ où $p$ est un polynôme et $n$ est la taille de l'entrée. Soit $k$ la taille de l'alphabet nécessaire pour écrire une configuration, qui dépend de l'alphabet de travail $\Gamma$ de $M_A$, de ses nombres d'états et de bandes. Chaque configuration est donc un mot de longueur $p(n)$ sur un alphabet à $k$ lettres. Il y a $k^{p(n)}$ tels mots, la taille du graphe des configuration est donc $2^{\alpha n^\ell}$ pour des constantes $\alpha, \ell$ ne dépendant que de $M_A$.

4) Comme dans le théorème de Cook, on représente les configurations de $M_A$ par un vecteur de variables propositionnelles (représentant l'état de la machine, le contenu des bandes, et la position des têtes). Ces vecteurs sont contraints par des formules propositionnelles qui assurent l'encodage correct. Si $\vec{c}, \vec{d}$ sont des vecteurs de variables associés à deux configurations $C, D$, la formule $\varphi_n(\vec{c}, \vec{d})$ exprime qu'il y a un chemin de longueur au plus $2^n$ entre les configurations $C$ et $D$. Cette formule est définie inductivement :
   - $\varphi_0(\vec{c}, \vec{d})$ doit exprimer que $C, D$ sont reliées par un chemin de longueur 0 ou 1 : soit $C = D$, ce qui s'exprime facilement, soit $M_A$ peut passer de $C$ à $D$ en une transition. Cette formule a déjà été écrite dans la preuve du théorème de Cook et se calcule en temps polynomial par rapport à la taille de l'entrée.
   - Puis, $\varphi_{n+1}$ exprime qu'il existe un chemin de longueur au plus $2^{n+1}$ entre $C$ et $D$, c'est-à-dire qu'il existe une configuration $E$ et des chemins de longueur au maximum $2^n$ de $C$ à $E$ et de $E$ à $D$. On voudrait donc définir $\varphi_{n+1}(\vec{c}, \vec{d})$ comme $\exists \vec{e} \varphi_n(\vec{c}, \vec{e}) \wedge \varphi_n(\vec{e}, \vec{d})$. On ne peut pas le faire car la formule $\varphi_n$ serait de taille $2^{O(n)}$. Cette difficulté est évitée par une alternance de quantificateurs permettant de n'utiliser récursivement qu'une occurrence de $\varphi_n$, et donc, de calculer $\varphi_n$ en temps polynomial.

$$\varphi_{n+1}(\vec{c}, \vec{d}) \stackrel{\text{def}}{=} \exists \vec{e} \forall \vec{a} \, (\vec{a} = \vec{c}) \vee (\vec{a} = \vec{d}) \Rightarrow \varphi_n(\vec{a}, \vec{e})$$

5) La question 2 montre que le problème FBQ est dans **PSPACE**. La question 4 montre que tout problème de la classe **PSPACE** se réduit polynomialement à lui, donc qu'il est **PSPACE**-difficile. En effet, l'entrée de $M_A$ est acceptée si et seulement s'il existe un chemin de la configuration initiale $\vec{c}_0$ à la configuration acceptante $\vec{c}_a$ (qu'on peut supposer unique), donc un chemin simple, c'est-à-dire si $\varphi_{\alpha n^\ell}(\vec{c}_0, \vec{c}_a)$ est vraie où $\alpha$ et $\ell$ sont les constantes de la question 3. Comme on peut calculer cette formule en temps polynomial, on a bien montré que le problème $A$ se réduit polynomialement à FBQ. On en déduit que FBQ est **PSPACE**-complet.

6) L'évaluation de la taille du graphe des configurations est la même pour une machine non déterministe. On peut donc refaire la preuve de la question 3 dans le cas d'une machine non déterministe utilisant un espace polynomial. Cela montre que tout problème de la classe **NPSPACE** se réduit polynomialement à FBQ. Le problème FBQ est donc **PSPACE** et **NPSPACE**-difficile. On en déduit que **PSPACE** = **NPSPACE**.

Fin