Sélection internationale Session 2011
École Normale Supérieure Paris
Épreuve de culture scientifique - **Informatique** Durée : 4 heures

Pour les candidats ayant choisi **l'informatique** comme **spécialité principale**

Si vous ne parvenez pas à répondre à une question, vous pouvez cependant l'utiliser comme hypothèse pour les questions suivantes.

L'usage de la calculatrice n'est pas autorisé.

Exercice 1.

1. Nous considérons les formules logiques construites à partir de variables propositionnelles issues d'un ensemble infini dénombrable (ce qui signifie que cet ensemble de variables est en bijection avec $\mathbb{N}$). Ces formules utilisent les connecteurs logiques usuels $\vee$ (qui signifie *ou*, la disjonction), $\wedge$ (*et*, la conjonction) et $\neg$ (*non*, la négation), ainsi que leurs combinaisons ($\Rightarrow$, défini par $a \Rightarrow b$ si et seulement si $\neg a \vee b$, ...).

   Quels sont les valeurs Booléennes sur les variables p_1, ..., p_n qui rendent la formule suivante vraie : $F = ((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3) \wedge ... \wedge (p_{n-1} \Rightarrow p_n))$? Prouvez le formellement.

   Une formule pour laquelle il existe de telles valeurs la rendant vraie est dite satisfiable. Un ensemble de formules est dit satisfiable si il existe de telles valeurs qui satisfont toutes les formules dans l'ensemble (sans changer de valeurs, par exemple l'ensemble $a \wedge \neg b, b \vee c, a \wedge \neg c$ n'est pas satisfiable).

2. Nous considérons maintenant l'ensemble P de formules sur les variables $p_0, p_1, ...$ Prouvez que P est satisfiable si et seulement si tout sous-ensemble fini de P est satisfiable. Vous pourrez envisager de donner à la variable p_n la valeur v_n telle que :

   - $v_0 = faux$ si pour tout sous-ensemble fini Q de P, il existe au moins une distribution de valeurs satisfaisant Q et donnant la valeur *faux* à p_0.
   - $v_0 = vrai$ dans le cas contraire.
   - $v_{n+1} = faux$ si pour tout sous-ensemble fini Q de P, il existe au moins une distribution de valeurs satisfaisant Q et donnant la valeur v_0 à p_0, v_1 à p_1, ..., v_n à p_n, et *faux* à p_{n+1}.
   - $v_{n+1} = vrai$ dans le cas contraire.

3. Considérons un ensemble E d'éléments. Un graphe G sur E est une relation binaire entre les éléments de E qui est symétrique (si $(x, y) \in G$, alors $(y, x) \in G$) et antiréflexive (pour tout $x \in E$, $(x, x) \notin G$).

   Considérons k un entier positif. Un graphe G est dit k-colorable si il existe $c : E \mapsto \{1, 2, ..., k\}$ telle que :

   $$(x, y) \in G \Longrightarrow c(x) \neq c(y)$$

   Écrire une formule contenant des variables propositionnelles $X_{x,i}$ (utilisant les connecteurs $\vee$, $\wedge$ et $\neg$) avec $x \in E$ et $i \in \{1, 2, ..., k\}$ qui exprime le fait qu'un graphe G est k-colorable.

4. Montrez qu'un graphe est k-colorable si et seulement si tous ses sous-graphes finis sont k-colorable (un sous-graphe est un sous-ensemble d'un graphe qui est lui-même un graphe, il est fini s'il comporte un nombre fini de sommets).

Exercice 2.
Les *expressions régulières* sont des expressions de l'algèbre libre sur la signature $(\Sigma, +, \cdot, ^*, 0, 1)$ où Σ est un alphabet, $+$ et $\cdot$ sont des opérateurs binaires, l'opérateur * est unaire et 0 et 1 sont des constantes. Par exemple, $a \cdot ((b + c^*) \cdot c)$ est un expression régulière.

Une expression régulière E définie le langage $L(E)$ en interprétant les opérateurs $+$, $\cdot$ et * par l'union, la concaténation et l'itération (étoile de Kleene).

$$\begin{array}{lcl} L(0) & = & \emptyset \\ L(1) & = & \{\varepsilon\} \\ L(a) & = & \{a\} \text{ with } a \in \Sigma \\ L(E + F) & = & L(E) \cup L(F) \\ L(E \cdot F) & = & L(E) \cdot L(F) = \{ww' \mid w \in L(E) \wedge w' \in L(F)\} \\ L(E^*) & = & \bigcup_{n \in \mathbb{N}} L(E)^n \end{array}$$

où $L(E)^n$ est défini par :

$$L(E)^0 = \{\varepsilon\} \qquad L(E)^{i+1} = L(E) \cdot L(E)^i$$

L'expression de notre exemple $(a \cdot ((b + c^*) \cdot c))$ définit le langage qui contient tous les mots qui commencent par un a, puis qui sont suivis par un b ou la répétition arbitraire de c et se terminent par un c. Par exemple, $\{ac, abc, acc, accc\} \subset L(a \cdot ((b + c^*) \cdot c))$.

Les expressions régulières peuvent être compiler en automates qui testent si in mot appartient à un langage défini par une expression. Ainsi, les expressions régulières sont utilisées dans de nombreuses outils telles que des commandes de recherche ou des générateurs d'analyseurs lexicaux.

Dans cet exercice, vous allez définir des fonctions et prouver des propriétés utilisées lors de la compilation d'expressions régulières en automates.

Basic functions
Soit λ une fonction qui associe à chaque expression E l'expression 1 si $L(E)$ contient ε et 0 sinon. Cette fonction peut être définie récursivement sur la structure des expressions régulières :

$$\begin{array}{l} \lambda(0) = 0 \\ \lambda(1) = 1 \\ \lambda(a) = 0 \quad \text{for all } a \in \Sigma \end{array}$$

1. Compléter la définition de la fonction λ (cas $\lambda(E + F)$, $\lambda(E \cdot F)$, and $\lambda(E^*)$).

Soit $First(E)$ l'ensemble des premiers symboles des mots de $L(E)$:

$$First(E) = \{a \mid \exists v, av \in L(E)\}$$

2. Définir la fonction $first$ qui calcule l'ensemble $First(E)$.

les dérivées
Soit $\mathcal{L}$ un langage et w un mot, alors la dérivée de $\mathcal{L}$ par rapport à w, notée $w^{-1}\mathcal{L}$, est l'ensemble des mots w' tels que ww' est dans $\mathcal{L}$:

$$w^{-1}\mathcal{L} = \{w' \mid ww' \in \mathcal{L}\}$$

Par exemple, si $\mathcal{L} = \{a, aab, baa\}$, alors $a^{-1}\mathcal{L} = \{\varepsilon, ba\}$.

La notion de dérivation s'étend aux expressions régulières :

$$L(w^{-1}E) = w^{-1}L(E)$$

Si a est un symbole (un mot de longueur 1), la dérivée d'une expression régulière par rapport à a peut être définie par :

$$\begin{array}{lcll} a^{-1}0 & = & 0 & \\ a^{-1}1 & = & 0 & \\ a^{-1}a & = & 1 & \\ a^{-1}b & = & 0 & \text{with } a \neq b \end{array}$$

3. Montrez que $a^{-1}(E \cdot F) = (a^{-1}E) \cdot F + \lambda(E) \cdot (a^{-1}F)$.

4. Complétez les règles de calcul par les cas $a^{-1}(E + F)$ et $a^{-1}E^*$ et prouvez que cet algorithme est correcte.

5. Utilisez les règles précédentes pour trouver les dérivées de l'expression régulière $(a+b)^*abb$ par rapport à a et b.

6. Caractérisez les langages $\mathcal{L}$ pour lesquels $a^{-1}\mathcal{L} = \emptyset$.

7. Caractérisez les langages $\mathcal{L}$ pour lesquels $a^{-1}\mathcal{L} = \mathcal{L}$.

8. Définissez une fonction qui calcule la dérivée d'une expression régulière E par rapport à un mot w.

9. Prouvez que pour toute expression régulière E, l'ensemble des dérivées $w^{-1}E$ pour tout mot w est fini(modulo associativité, commutativité et idempotence de l'opérateur $+$).

Une expressions régulière E peut être décomposée en une somme d'expressions régulières en utilisant les dérivés :

$$E = \lambda(E) + \Sigma_{a\in\Sigma} a(a^{-1}E)$$

Puisque l'ensemble des dérivées est fini, on en déduit que cette décomposition s'applique aussi à chacune des dérivées de E. Ainsi, on peut construire un automate fini $(Q, \Sigma, \delta, i, F)$ reconnaissant le langage $L(E)$ qui est défini par :

- $Q = \{w^{-1}E \mid w \in \Sigma^*\}$
- $i = E$
- $F = \{q \mid \lambda(q) = 1\}$
- $\delta(q, a) = a^{-1}q, \ \forall q \in Q$ and $\forall a \in \Sigma$

On associe à chaque dérivée un état de l'automate, et les transitions de l'automate sont issues des décompositions.

Implantation
En utilisant le langage de programmation de votre choix, vous allez devoir programmer quelques fonctions sur les expressions régulières.

10. Justifiez en quelques lignes votre choix de langage de programmation.

11. Définissez la structure de donnée qui représente les expressions régulières.

12. Programmez la fonction `null` qui prend en argument une expression régulière E et retourne `true` si $L(E)$ contient ε et `false` sinon.

    Quelle est la complexité en temps de votre fonction ?

13. Programmez une fonction `first` qui prend en argument une expression régulière E et retourne l'ensemble `First(E)`.

    Si on suppose que l'ajout d'un élément dans un ensemble est fait en temps constant, quelle est la complexité en temps de votre fonction ? Est-il possible de fournir une implantation de la fonction `first` qui s'exécute en temps linéaire ?