A 2014 INFO. MP

ECOLE DES PONTS PARISTECH,
SUPAERO (ISAE), ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES DE SAINT-ETIENNE, MINES DE NANCY,
TELECOM BRETAGNE, ENSAE PARISTECH (FILIERE MP)
ECOLE POLYTECHNIQUE (FILIERE TSI)

CONCOURS 2014

EPREUVE d'INFORMATIQUE

Filière : MP

Durée de l'épreuve : 3 heures.
L'utilisation d'une calculatrice est autorisée.

Sujet mis à la disposition des concours :
CYCLE INTERNATIONAL, ECOLES DES MINES, TELECOM SUDPARIS, TPE-EIVP.

L'énoncé de cette épreuve comporte 8 pages.

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :
INFORMATIQUE - MP

Recommandations aux candidats

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**

Composition de l'épreuve

L'épreuve est constituée :

- d'un exercice sur les automates et les langages : pages 2 et 3 ;
- d'un problème d'algorithmique et programmation : pages 3 à 8.

Première partie : automates et langages

Un *alphabet* Σ est un ensemble fini d'éléments appelés *lettres*. Un *mot sur* Σ est une suite finie, éventuellement vide, de lettres de Σ ; la *longueur* d'un mot u est le nombre de lettres composant u et est notée $|u|$; le mot de longueur nulle est noté ε. On note par Σ^* l'ensemble de tous les mots sur Σ. Un *langage sur* $\boldsymbol{\Sigma}$ est une partie de Σ^*.
Soit u un mot sur un alphabet Σ ; pour toute lettre x de Σ, on note $|u|_x$ le nombre d'occurrences de la lettre x dans le mot u.
On note $\mathbf{N}$ l'ensemble des entiers naturels.

On considère dans cet exercice l'alphabet $\Sigma = \{a, b\}$.
Soit f une application quelconque définie sur $\mathbf{N}$ et à valeurs dans $\mathbf{N}$. On note $L(f)$ l'ensemble des mots u appartenant à Σ^* vérifiant l'égalité $|u|_a = f(|u|_b)$.

❒ 1 – On considère la fonction f_1 définie par : $\forall\, n \in \mathbf{N}$, $f_1(n) = 2$. Dessiner un automate reconnaissant le langage $L(f_1)$.

On considère la fonction f_2 définie par : $\forall\, n \in \mathbf{N}$, $f_2(n) = 1$ si n est pair et $f_2(n) = 0$ sinon.

❒ 2 – Décrire $L(f_2)$ par une expression rationnelle de la forme $\alpha(bab + a + b)\beta$, où α et β sont des expressions rationnelles à déterminer. Justifier la réponse.
Remarque : on note aussi | l'opérateur noté + dans l'expression rationnelle ci-dessus.

❒ 3 – Dessiner un automate non nécessairement déterministe reconnaissant le langage décrit par l'expression rationnelle $bab + a + b$. Cet automate devra nécessairement posséder un seul état initial et un seul état final.

❒ 4 – En s'appuyant sur l'expression rationnelle obtenue à la question ❒ 2, compléter l'automate obtenu à la question précédente pour obtenir un automate non déterministe reconnaissant le langage $L(f_2)$. Cet automate devra nécessairement posséder un seul état initial et un seul état final.

❒ 5 – Déterminiser l'automate obtenu à la question précédente. On utilisera un algorithme vu en cours et on ne fera apparaître que les états accessibles depuis l'état initial.

❒ 6 – Montrer que si f n'est pas majorée par une constante, alors $L(f)$ n'est pas rationnel.

❒ 7 – On considère le langage $L_=$ sur Σ défini par $L_= = \{u \in \Sigma^*$ vérifiant $|u|_a = |u|_b\}$. Le langage $L_=$ est-il rationnel ?

❒ 8 – On considère le langage $L_\leq$ sur Σ défini par : $L_\leq = \{u \in \Sigma^*$ avec $|u|_a \leq |u|_b\}$. Le langage $L_\leq$ est-il rationnel ? On utilisera le résultat de la question précédente.

❒ 9 – On considère le langage $L_>$ sur Σ défini par : $L_> = \{u \in \Sigma^* \text{ avec } |u|_a > |u|_b\}$. Le langage $L_>$ est-il rationnel ? On utilisera le résultat de la question précédente.

❒ 10 – Montrer que la réciproque de la proposition énoncée dans la question ❒ 6 est fausse.
Indication : on pourra admettre que le langage P des mots de la forme b^n où b est une lettre et n est un entier premier n'est pas rationnel.

Seconde partie : logique et programmation

Préliminaire concernant la programmation

Il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début d'épreuve le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que cela est nécessaire. Lorsque le candidat écrira une fonction ou une procédure, il pourra faire appel à une autre fonction ou procédure définie dans les questions précédentes ; il pourra aussi définir une procédure ou une fonction auxiliaire. Enfin, si les paramètres d'une fonction ou d'une procédure à écrire sont supposés vérifier certaines hypothèses, il ne sera pas utile, dans l'écriture de cette fonction ou de cette procédure, de tester si les hypothèses sont bien vérifiées.
Dans les énoncés du problème, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité, mais du point de vue mathématique pour la police en italique (par exemple *n*) et du point de vue informatique pour celle en romain (par exemple `n`).
On ne se préoccupera pas d'un éventuel dépassement du plus grand entier codable dans le langage de programmation choisi.

Indications pour la programmation
Caml
Si `n` est un entier, l'instruction
```
let A = make_matrix n n false;;
```
permet de construire une matrice carrée booléenne `A` à `n` lignes et `n` colonnes, et dont les cases sont initialisées à `false`.
Fin des indications pour Caml

Pascal
On utilisera dans tout le sujet les déclarations ci-dessous.
```
CONST MAX = 100;

type Tableau_Boolean = array [0 .. MAX - 1] of Boolean;

type Matrice_Boolean = array [0 .. MAX - 1] of Tableau_Boolean;
```

On supposera que la constante `MAX` est suffisamment grande pour que les tableaux et matrices de types `Tableau_Boolean` et `Matrice_Boolean` puissent coder les tableaux et matrices considérés dans ce problème.

Fin des indications pour Pascal

On considère un ensemble fini Π de n propositions logiques distinctes : $\Pi = \{P_0, P_1, \ldots, P_{n-1}\}$. On suppose qu'un ensemble d'implications entre ces propositions, appelé *ensemble des implications initiales* et noté I, a déjà été établi. On peut en général déduire d'autres implications à partir de l'ensemble des implications initiales en utilisant la transitivité des implications : pour deux propositions P et Q appartenant à Π, si Q se déduit de P à l'aide d'une suite d'implications appartenant à I, on dit que « P implique Q », ce que l'on note $P \Rightarrow Q$. Dans toute la suite, on s'intéresse aux implications que l'on peut déduire de I.

Pour tout P dans Π, on suppose que I contient l'implication $P \Rightarrow P$; une telle implication, nommée *boucle*, est notée $P \Rightarrow_0 P$.

Si P et Q sont dans Π, la notation $P \Rightarrow_1 Q$ signifie que l'implication $P \Rightarrow Q$ appartient à I (pour tout P dans Π, on a donc aussi $P \Rightarrow_1 P$).

Exemple 1 : $n = 4$, $I = \{P_0 \Rightarrow_0 P_0,\ P_1 \Rightarrow_0 P_1,\ P_2 \Rightarrow_0 P_2,\ P_3 \Rightarrow_0 P_3, P_0 \Rightarrow_1 P_2, P_2 \Rightarrow_1 P_3, P_3 \Rightarrow_1 P_0, P_3 \Rightarrow_1 P_1\}$.

Exemple 2 : $n = 4$, $I = \{P_0 \Rightarrow_0 P_0,\ P_1 \Rightarrow_0 P_1, P_2 \Rightarrow_0 P_2,\ P_3 \Rightarrow_0 P_3, P_0 \Rightarrow_1 P_1, P_1 \Rightarrow_1 P_2, P_2 \Rightarrow_1 P_1,\ P_3 \Rightarrow_1 P_2\}$.

Pour P et Q dans Π, P implique Q (autrement dit, $P \Rightarrow Q$) s'il existe un entier $k \geq 0$ et $k + 1$ propositions $P_{i_0}, P_{i_1}, \ldots, P_{i_j}, P_{i_{(j+1)}}, \ldots, P_{i_k}$ appartenant à Π tels que l'on ait :

- $P_{i_0} = P$,
- $P_{i_k} = Q$,
- pour j vérifiant $0 \leq j \leq k - 1$, l'implication $P_{i_j} \Rightarrow P_{i_{(j+1)}}$ appartient à I.

Avec les notations ci-dessus, on dit alors qu'il existe une *preuve de longueur k de l'implication* $P \Rightarrow Q$, ce que l'on note $P \Rightarrow_k Q$. Les implications de I sont donc les preuves de longueur 0 (si $P = Q$) ou 1.

Dans l'exemple 1, on a $P_0 \Rightarrow_2 P_3$ car on a $P_0 \Rightarrow_1 P_2$ et $P_2 \Rightarrow_1 P_3$; on a aussi $P_0 \Rightarrow_3 P_3$ car on peut ajouter une boucle en considérant les trois implications $P_0 \Rightarrow_0 P_0$, $P_0 \Rightarrow_1 P_2$ et $P_2 \Rightarrow_1 P_3$. En revanche, on n'a pas : $P_0 \Rightarrow_2 P_1$.

Dans l'exemple 1, les implications qui peuvent être prouvées mais qui n'appartiennent pas à I sont : $P_0 \Rightarrow P_1$, $P_0 \Rightarrow P_3$, $P_2 \Rightarrow P_0$, $P_2 \Rightarrow P_1$, $P_3 \Rightarrow P_2$.

❒ 11 – Pour l'exemple 2, donner la liste des implications qui peuvent être prouvées mais qui n'appartiennent pas à I.

❒ 12 – Soient P et Q dans Π ; soient h et k deux entiers positifs ou nuls vérifiant : $h \leq k$. Montrer que si on a $P \Rightarrow_h Q$, alors on a aussi $P \Rightarrow_k Q$.

❒ 13 – Soient P et Q dans Π. Montrer qu'on a l'implication $P \Rightarrow Q$ si et seulement si on a $P \Rightarrow_{n-1} Q$.

Une matrice booléenne est une matrice dont les coefficients prennent uniquement les valeurs faux et vrai (`false` et `true` en langage de programmation). Le produit de matrices booléennes s'obtient selon la formule habituelle en prenant comme somme de deux valeurs booléennes le « ou logique » (disjonction, notée $\vee$) et comme produit de deux valeurs booléennes le « et logique » (la conjonction, notée $\wedge$) ; le produit de deux matrices A et B est noté $A \times B$.

Par exemple, si on considère les deux matrices : $A_0 = \begin{pmatrix} \text{vrai} & \text{vrai} \\ \text{faux} & \text{vrai} \end{pmatrix}$ et $B_0 = \begin{pmatrix} \text{faux} & \text{vrai} \\ \text{vrai} & \text{faux} \end{pmatrix}$, le produit $A_0 \times B_0$ vaut $\begin{pmatrix} \text{vrai} & \text{vrai} \\ \text{vrai} & \text{faux} \end{pmatrix}$.

On ne s'intéressera dans ce problème qu'à des matrices carrées ; la dimension d'une matrice carrée est son nombre de lignes (et donc de colonnes). Si k est un entier strictement positif, on obtient la matrice A^k en multipliant $k - 1$ fois la matrice A par elle-même.

❒ 14 – On considère deux matrices carrées booléennes A et B de même dimension d. Il s'agit d'écrire en langage de programmation le calcul du produit $A \times B$.

Caml : Écrire en Caml une fonction nommée `mult` telle que, si `A` et `B` codent A et B, alors `mult A B` renvoie une matrice codant le produit $A \times B$.

Pascal : Écrire en Pascal une fonction nommée `mult` telle que si :

- `A` et `B`, de type `Matrice_Boolean`, codent A et B,
- `d`, de type `Integer`, contient la valeur de d,

alors `mult(A, B, d)` renvoie une matrice, de type `Matrice_Boolean`, codant le produit $A \times B$.

On considère encore l'ensemble Π des n propositions logiques $P_0, P_1, \ldots, P_{n-1}$ et l'ensemble I d'implications initiales entre ces propositions. On associe à Π et I une matrice A carrée booléenne de dimension n définie de la façon suivante :

- les lignes et les colonnes de A sont indicées de 0 à $n - 1$;
- soient i et j deux entiers vérifiant $0 \leq i \leq n - 1$ et $0 \leq j \leq n - 1$; en notant $A[i, j]$ le coefficient de A situé sur la ligne d'indice i et la colonne d'indice j, $A[i, j]$ vaut vrai si et seulement si l'implication $P_i \Rightarrow P_j$ appartient à I.

Ainsi, les matrices A_1 et A_2 correspondant respectivement à l'exemple 1 et à l'exemple 2 sont :

$$\begin{pmatrix} \text{vrai} & \text{faux} & \text{vrai} & \text{faux} \\ \text{faux} & \text{vrai} & \text{faux} & \text{faux} \\ \text{faux} & \text{faux} & \text{vrai} & \text{vrai} \\ \text{vrai} & \text{vrai} & \text{faux} & \text{vrai} \end{pmatrix}$$

Matrice A_1

$$\begin{pmatrix} \text{vrai} & \text{vrai} & \text{faux} & \text{faux} \\ \text{faux} & \text{vrai} & \text{vrai} & \text{faux} \\ \text{faux} & \text{vrai} & \text{vrai} & \text{faux} \\ \text{faux} & \text{faux} & \text{vrai} & \text{vrai} \end{pmatrix}$$

Matrice A_2

❒ 15 – Montrer que, pour i et j vérifiant $0 \leq i \leq n - 1$, $0 \leq j \leq n - 1$ et pour tout k strictement positif, le coefficient $A^k[i, j]$ vaut vrai si et seulement si on a $P_i \Rightarrow_k P_j$.

❒ 16 – Montrer que, pour tout $k \geq n-1$, on a $A^k = A^{n-1}$.

On appelle *fermeture transitive de A* et on note *FT*(*A*) la matrice A^{n-1}.

❒ 17 – Il s'agit d'écrire en langage de programmation une fonction nommée *FT* qui calcule la fermeture transitive de *A* en utilisant des multiplications de matrice.
Caml : Écrire en Caml la fonction `FT` telle que, si `A` code la matrice *A*, alors `FT A` renvoie la matrice *FT*(*A*).
Pascal : Écrire en Pascal la fonction `FT` telle que si :

- `A`, de type `Matrice_Boolean`, code la matrice *A*,
- `n`, de type `Integer`, contient la dimension de *A*,

alors `FT(A, n)`, de type `Matrice_Boolean`, renvoie la matrice *FT*(*A*).

❒ 18 – Soit *P* une proposition appartenant à Π. Il s'agit d'écrire en langage de programmation une fonction nommée *deduction* qui détermine toutes les propositions *Q* appartenant à Π telles que l'on ait $P \Rightarrow Q$. On utilisera pour cela la récursivité.
ATTENTION : On exige que la **complexité** de cette fonction soit **de l'ordre de n^2**. On ne justifiera pas la complexité de la fonction qui sera écrite.
Caml : Écrire en Caml la fonction `deduction` telle que si :

- `A` code la matrice *A*,
- `i` est un entier compris entre 0 et $n-1$,

alors `deduction A i` renvoie un tableau de booléens de longueur *n* tel que, pour *j* compris entre 0 et $n-1$, la valeur d'indice `j` de ce tableau vaut `true` si et seulement si on a $P_i \Rightarrow P_j$.
Pascal : Écrire en Pascal la fonction `deduction` telle que si :

- `A`, de type `Matrice_Boolean`, code la matrice *A*,
- `n`, de type `Integer`, contient la valeur de *n*,
- `i`, de type `Integer`, contient un entier compris entre 0 et $n-1$,

alors `deduction(A, n, i)` renvoie un tableau de type `Tableau_Boolean` tel que, pour *j* compris entre 0 et $n-1$, la valeur d'indice `j` de ce tableau vaut `true` si et seulement si on a $P_i \Rightarrow P_j$.

❒ 19 – Il s'agit d'écrire en langage de programmation une fonction nommée *FT_bis* **de complexité n^3** calculant la matrice *FT*(*A*).
Caml : En utilisant la fonction `deduction`, écrire en Caml la fonction `FT_bis` telle que si `A` code la matrice *A*, alors `FT_bis A` renvoie la matrice *FT*(*A*).
Pascal : En utilisant la fonction `deduction`, écrire en Pascal la fonction `FT_bis` telle que si :

- `A`, de type `Matrice_Boolean`, code la matrice *A*,
- `n`, de type `Integer`, contient la valeur de *n*,

alors `FT_bis(A, n)` renvoie la matrice *FT*(*A*).

Soient *P* et *Q* deux propositions appartenant à Π. On dit que les propositions *P* et *Q* sont *équivalentes* si on a : $P \Rightarrow Q$ et $Q \Rightarrow P$.
Soit *P* appartenant à Π. On dit ici que *P* est un *axiome* si on a la propriété suivante : quelle que soit la proposition *Q* appartenant à Π, si on a $Q \Rightarrow P$, alors on a aussi $P \Rightarrow Q$ et donc *P* et *Q* sont équivalentes ; autrement dit, *P* est équivalente à toute proposition qui l'implique.

❒ 20 – Donner tous les axiomes dans l'exemple 1.

❒ 21 – Donner tous les axiomes dans l'exemple 2.

On pose $B = FT(A)$. Des questions précédentes, on déduit que, si i et j sont deux entiers compris entre 0 et $n - 1$, on a $P_i \Rightarrow P_j$ si et seulement si $B[i, j]$ vaut vrai.

❒ 22 – Il s'agit, connaissant la matrice B, de programmer une fonction *est_axiome* qui indique si une proposition donnée est ou non un axiome.
Caml : Écrire en Caml la fonction `est_axiome` telle que si :
- `B` code la matrice B,
- `i` est un entier compris entre 0 et $n - 1$,

alors `est_axiome B i` renvoie la valeur `true` si P_i est un axiome et la valeur `false` sinon.
Pascal : Écrire en Pascal la fonction `est_axiome` telle que si :
- `B`, de type `Matrice_Boolean`, code la matrice B,
- `n`, de type `Integer`, contient la valeur de n,
- `i`, de type `Integer`, contient un entier compris entre 0 et $n - 1$,

alors `est_axiome(B, n, i)` renvoie la valeur `true` si P_i est un axiome et la valeur `false` sinon.

On appelle *suite unidirectionnelle de propositions* une suite $(Q_0, Q_1, \ldots, Q_h)$, où h est un entier positif ou nul, telle que :
1. pour i vérifiant $0 \leq i \leq h$, Q_i appartient à Π,
2. pour i vérifiant $0 \leq i \leq h - 1$, $Q_i \Rightarrow Q_{i+1}$,
3. pour i vérifiant $0 \leq i \leq h - 1$, Q_{i+1} n'implique pas Q_i.

❒ 23 – Montrer que les propositions d'une suite unidirectionnelle de propositions sont deux à deux distinctes.

❒ 24 – Soit Q une proposition appartenant à Π. Montrer qu'il existe un axiome P avec $P \Rightarrow Q$.

On admet le résultat suivant : on peut partitionner Π en sous-ensembles de sorte que deux propositions de Π soient équivalentes si et seulement si elles appartiennent au même sous-ensemble. On appelle classes d'équivalence ces sous-ensembles.
Par définition : les classes d'équivalence sont non vides, l'union des classes d'équivalence est égale à Π, l'intersection de deux classes d'équivalence est vide.

Dans l'exemple 1, il y a deux classes d'équivalence : les classes $\{P_0, P_2, P_3\}$ et $\{P_1\}$.

❒ 25 – Donner les classes d'équivalence dans l'exemple 2.

❒ 26 – On considère une classe d'équivalence C contenant un axiome. Montrer que toutes les propositions contenues dans C sont des axiomes.

On dit qu'une classe d'équivalence est une *classe source* si elle contient un axiome (auquel cas tous les éléments de la classe source sont des axiomes).

Soit X une partie de Π. On dit ici que X est une *axiomatique* si, quelle que soit la proposition Q appartenant à Π, il existe une proposition P appartenant à X vérifiant $P \Rightarrow Q$.

❒ 27 – Montrer qu'on obtient une axiomatique de cardinal minimum en choisissant une et une seule proposition dans chacune des classes sources.

❒ 28 – On pose encore $B = FT(A)$. Il s'agit d'écrire en langage de programmation une fonction nommée *axiomatique* qui détermine une axiomatique de cardinal minimum. On pourra utiliser un tableau pour éliminer, lorsqu'on a choisi un axiome, les axiomes qui lui sont équivalents.

Caml : Écrire en Caml la fonction `axiomatique` telle que, si `B` code la matrice B, alors `axiomatique B` renvoie une liste d'entiers contenant les indices de propositions de Π formant une axiomatique de cardinal minimum.

Pascal :

On définit le type suivant :

```
type Pile = record
     table : array [0 .. MAX - 1] of Integer;
     nb : Integer;
end;
```

Si `P` est de type `Pile`, `P` code une pile de `P.nb` entiers situés dans `P.table` entre les indices 0 et `P.nb` – 1.

Écrire en Pascal la fonction `axiomatique` telle que si :

- `B`, de type `Matrice_Boolean`, code la matrice B,
- `n`, de type `Integer`, contient la valeur de n,

alors `axiomatique(B, n)` renvoie un résultat de type `Pile` contenant les indices de propositions de Π formant une axiomatique de cardinal minimum.