Concours Commun INP (CCINP), France

Session 2017, filière MP

Informatique

Durée : 4 heures

Ce que traite le sujet

- Partie I : logique et calcul des propositions
- Partie II : algorithmique et programmation (Informatique pour tous)
- Partie III : automates et langages

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/mp/2017/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUNS POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE MP

INFORMATIQUE

Jeudi 4 mai : 14 h - 18 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet est composé de trois parties, toutes indépendantes.

Ce sujet évalue les compétences acquises dans les enseignements d'*Informatique pour tous* et d'option *Informatique*. **Toutes les questions** doivent être traitées par les candidats.
Les questions demandant d'écrire une fonction en langage Python doivent exploiter le contenu des enseignements d'*Informatique pour tous*. Les questions demandant d'écrire une fonction en langage CaML doivent exploiter le contenu des enseignements d'option *Informatique*.
Les fonctions écrites en langage Python ne devront pas être récursives sauf dans la **question Q31** page 9.
Les fonctions écrites en langage CaML devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire **for**,**while**, ...), ni de références, ni d'exceptions.

Partie I - Logique et calcul des propositions

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau.

Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons X, Y et Z. Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner.

Le premier sujet abordé est la région dans laquelle se trouve le village :

X indique : « Le village se trouve dans la vallée » ;

Z réplique : « Non, il ne s'y trouve pas » ;

X reprend : « Ou alors dans les collines ».

Nous noterons V et C les variables propositionnelles associées à la région dans laquelle se trouve le village.

Nous noterons X_1 et Z_1 les formules propositionnelles correspondant aux affirmations de X et de Z sur le premier sujet.

Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

X dit : « Le chemin de gauche conduit au village » ;

Z répond : « Tu as raison » ;

X complète : « Le chemin de droite y conduit aussi » ;

Y affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;

Z indique : « Celui du milieu n'y conduit pas ».

Nous noterons G, M, D les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village.

Nous noterons X_2, Y_2 et Z_2 les formules propositionnelles correspondant aux affirmations de X, de Y et de Z sur le second sujet.

Q1. Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_1 et Z_1.

Q2. Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions X_1 et Z_1 dépendant des variables V et C.

Q3. En utilisant la résolution avec les propriétés des opérateurs booléens et les formules de De Morgan en calcul des propositions, déterminer dans quelle région vous devez vous rendre pour rejoindre le village.

Q4. Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_2, Y_2 et Z_2.

Q5. Représenter les informations données par les participants sous la forme de trois formules du calcul des propositions X_2, Y_2 et Z_2 dépendant des variables G, M et D.

Q6. En utilisant la résolution avec les tables de vérité en calcul des propositions, déterminer quel chemin vous devez suivre pour rejoindre le village.

Q7. En admettant que les trois participants aient menti, pouviez-vous prendre d'autres chemins ? Si oui, le ou lesquels ?

Partie II - Algorithmique et programmation (Informatique pour tous)

Cette partie étudie l'algorithme de recherche dichotomique de la position d'une valeur dans une séquence croissante d'entiers. Pour simplifier les notations et les preuves, les séquences manipulées ne contiennent qu'une seule fois chaque valeur. Les résultats obtenus se généralisent aux séquences croissantes quelconques.

Définition II.1 (séquence) : *soient* $d, f \in \mathbb{N}$ *avec* $d \leq f + 1$,

- *une séquence s de valeurs* v_i *avec* $i \in \mathbb{N}$ *et* $d \leq i \leq f$ *est notée* $s = \langle v_i \rangle_{i=d}^{f}$;
- *sa taille est notée* $| s | = f - d + 1$;
- *si* $i \in \mathbb{N}$ *et* $d \leq i \leq f$, *sa i-ème valeur est notée* $s_i = v_i$;
- *son domaine, c'est-à-dire l'intervalle des indices de ses valeurs, est noté* $dom(\langle v_i \rangle_{i=d}^{f}) = [d, f]$;
- *son co-domaine, c'est-à-dire l'ensemble de ses valeurs, est noté* $codom(\langle v_i \rangle_{i=d}^{f}) = \{v_i\}_{i=d}^{f}$;
- *la séquence vide de taille* 0 *est notée* $\langle\rangle$;
- *si* $d \leq d'$ *et* $d' \leq f' + 1$ *et* $f' \leq f$, *alors* $\langle v_i \rangle_{i=d'}^{f'}$ *de taille* $f' - d' + 1$ *désigne la sous-séquence de* $\langle v_i \rangle_{i=d}^{f}$ *contenant les valeurs de* $v_{d'}$ *à* $v_{f'}$.

Les valeurs contenues dans les séquences s manipulées par la suite sont toutes distinctes, c'est-à-dire que le cardinal du co-domaine de s est égal à la taille de s ($card(codom(s)) = | s |$).

Une séquence sera représentée en Python par une liste. Dans la suite de cet exercice, nous considérons la fonction `position` du programme suivant en langage Python.

```
def position(v,p):
    d = 0
    f = len(p)-1
    while (d < f):
        m = d + (f - d) // 2
        print (v, d, m, f, p[d], p[m], p[f])
        if (v < p[m]):
            f = m - 1
        elif (p[m] < v):
            d = m + 1
        else:
            f = d = m
    return d

exemple = [ 1, 4, 7, 9, 12, 15]
resultat = position(9,exemple)
```

Listing 1 – Recherche dichotomique en Python

Q8. Quelles sont les informations affichées lors de l'exécution du programme complet du **Listing 1** ? Que contient la variable `resultat` après cette exécution ? Que contient la variable `exemple` après cette exécution ?

Q9. Soient la séquence p et les entiers r et v tels que r = position(v,p), avec :

(i) $\mathtt{p} = \langle \mathtt{p}_0, \ldots, \mathtt{p}_n \rangle$ avec $n \in \mathbb{N}$;

(ii) $\forall i \in [0, n[, \mathtt{p}_i < \mathtt{p}_{i+1}$;

(iii) $\exists i \in [0, n], \mathtt{p}_i = \mathtt{v}$.

Montrer que $\mathtt{v} = \mathtt{p}_\mathtt{r}$.
Vous utiliserez pour cela la propriété suivante dont vous montrerez qu'il s'agit d'un invariant pour la boucle :

$$0 \leq \mathrm{d} \leq \mathrm{f} \leq n \wedge \exists i \in [\mathrm{d}, \mathrm{f}], \mathtt{p}_i = \mathtt{v}.$$

Q10. Montrer que le calcul de la fonction position se termine quelles que soient les valeurs de ses paramètres v et p.

Q11. Donner des exemples de valeurs des paramètres v et p de la fonction position qui correspondent au pire cas en temps d'exécution.
Montrer que la complexité en temps d'exécution dans le pire cas de la fonction position, en fonction de la taille n des séquences transmises comme paramètre, est de $O(\log(n))$.

Partie III - Automates et langages

Cette partie étudie l'opérateur de produit synchronisé $\|_S$ sur l'alphabet S de deux automates sur un même alphabet X tel que $S \subseteq X$. Cet opérateur est utilisé pour modéliser des activités concurrentes.

1 Mots et langages

Définition III.1 (alphabet, mot, langage) : *un alphabet X est un ensemble de symboles. $\epsilon \notin X$ est le symbole représentant le mot vide. $X^\star$ est l'ensemble contenant ϵ et les mots composés de séquences de symboles de X. Un langage L sur X est un sous-ensemble de $X^\star$.*

Nous étudierons deux implantations des mots et langages : d'une part en algorithmique récursive programmée en langage CaML ; d'autre part en algorithmique itérative programmée en langage Python.

1.1 Algorithmique récursive (option Informatique)

Nous utiliserons par la suite en CaML le type char pour représenter les symboles de l'alphabet et les listes de symboles pour représenter les mots.

Q12. Écrire en CaML une définition pour les types alphabet, mot et langage qui représentent les alphabets, mots et langages sur les symboles de cet alphabet.

Q13. Écrire en CaML une fonction prefixer, de type mot -> langage -> langage, telle que l'appel (prefixer p l) sur un mot p et un langage l, renvoie un langage contenant les mêmes mots que l préfixés par le mot p. L'algorithme utilisé ne devra parcourir qu'une seule fois le langage l. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Q14. Calculer une estimation de la complexité de la fonction prefixer en fonction du nombre de mots du langage l. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

1.2 Algorithmique itérative (Informatique pour tous)

Nous utiliserons par la suite en Python des chaînes de caractère de taille 1 (type `str`) pour représenter les symboles de l'alphabet.

Q15. Proposer une représentation en Python des mots et langages sur cet alphabet.

Q16. Écrire en Python une fonction `prefixer`, telle que l'appel `prefixer(p,l)` sur un mot `p` et un langage `l`, renvoie un langage contenant les mêmes mots que `l` préfixés par le mot `p`. L'algorithme utilisé ne devra parcourir qu'une seule fois le langage `l`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.

Q17. Calculer une estimation de la complexité de la fonction `prefixer` en fonction du nombre de mots du langage `l`. Cette estimation ne prendra en compte que le nombre d'itérations effectuées.

2 Automate fini

2.1 Définition d'un automate fini

Définition III.2 (automate fini) : *un automate fini sur un alphabet X est un quintuplet $\mathcal{A} = (Q, X, I, T, \gamma)$ composé :*

- *d'un ensemble fini d'états : Q ;*
- *d'un ensemble d'états initiaux : $I \subseteq Q$;*
- *d'un ensemble d'états terminaux : $T \subseteq Q$;*
- *d'une relation de transition : $\gamma \subseteq Q \times X \times Q$.*

Pour une transition $(o, e, d) \in \gamma$ donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.
Remarquons que γ est le graphe d'une application de transition $\delta : Q \times X \to \mathcal{P}(Q)$ dont les valeurs sont définies par :

$$\forall o \in Q,\ \forall e \in X,\ \delta(o, e) = \{d \in Q \mid (o, e, d) \in \gamma\}.$$

La notation γ est plus adaptée que δ à la formalisation et la construction des preuves dans le cadre des automates non déterministes.

2.2 Langage accepté par un automate fini

Définition III.3 (transition sur un mot) : *l'extension $\gamma^\star$ de γ à $Q \times X^\star \times Q$ est définie par :*

- *fermeture réflexive : pour tout état q de Q,*

$$(q, \epsilon, q) \in \gamma^\star$$

- *fermeture transitive : pour tout symbole e de X, pour tout mot m de $X^\star$, pour tous états o, d de Q,*

$$(o, e.m, d) \in \gamma^\star \Leftrightarrow \exists q \in Q,\ ((o, e, q) \in \gamma) \wedge ((q, m, d) \in \gamma^\star).$$

Définition III.4 (langage accepté par un automate) : *le langage sur X accepté par un automate fini $\mathcal{A}$ est :*

$$L(\mathcal{A}) = \{m \in X^\star \mid \exists i \in I,\ \exists d \in T,\ (i, m, d) \in \gamma^\star\}.$$

2.3 Représentation graphique d'un automate

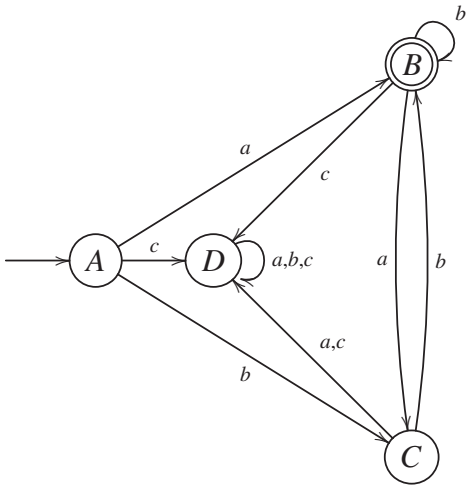
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la relation de transition γ sont représentées par un graphe orienté dont les nœuds encerclés sont les états et les arêtes sont les transitions ;
- tout état initial $i \in I$ est désigné par une flèche $\longrightarrow (i)$;
- tout état terminal t est entouré d'un double cercle $\textcircled{\textcircled{t}}$;
- une arête étiquetée par le symbole $e \in X$ va de l'état o à l'état d si et seulement si $(o, e, d) \in \gamma$.

Exemple III.1 : *l'automate* $\mathcal{E}_1 = (Q_1, X, I_1, T_1, \delta_1)$ *tel que*

$$Q_1 = \{A, B, C, D\} \quad X = \{a, b, c\} \quad I_1 = \{A\} \quad T_1 = \{B\}$$
$$\gamma_1 = \left\{ \begin{array}{llllll} (A,a,B), & (A,b,C), & (A,c,D), & (B,a,C), & (B,b,B), & (B,c,D), \\ (C,a,D), & (C,b,B), & (C,c,D), & (D,a,D), & (D,b,D), & (D,c,D) \end{array} \right\}$$

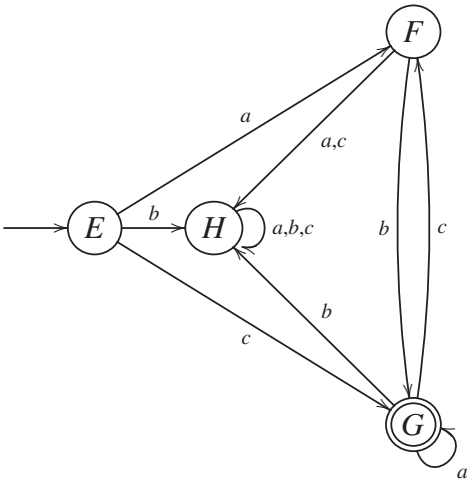
est représenté par le graphe suivant :



Exemple III.2 : *l'automate* $\mathcal{E}_2 = (Q_2, X, I_2, T_2, \delta_2)$ *tel que*

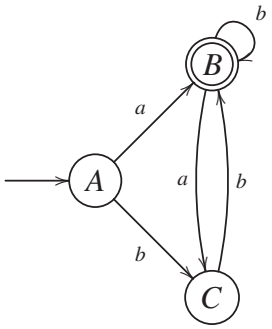
$$Q_2 = \{E, F, G, H\} \quad X = \{a, b, c\} \quad I_2 = \{E\} \quad T_2 = \{G\}$$
$$\gamma_2 = \left\{ \begin{array}{llllll} (E,a,F), & (E,b,H), & (E,c,G), & (F,a,H), & (F,b,G), & (F,c,H), \\ (G,a,G), & (G,b,H), & (G,c,F), & (H,a,H), & (H,b,H), & (H,c,H) \end{array} \right\}$$

est représenté par le graphe suivant :

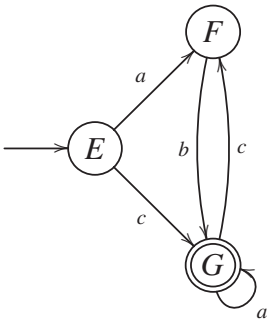


Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne participent pas à la construction de mots du langage. Il s'agit, d'une part, des états qui ne figurent dans aucune séquence de transitions allant de l'état initial à un état terminal et, d'autre part, des transitions dont les états inutiles sont l'origine ou la destination. Un automate dans lequel ces états et transitions inutiles ont été éliminés est appelé automate émondé.

Exemple III.3 : *le sous-automate de $\mathcal{E}_1$ (**Exemple III.1** de la page 6) composé des états et transitions utiles est représenté par le graphe suivant :*



Exemple III.4 : *le sous-automate de $\mathcal{E}_2$ (**Exemple III.2** de la page 6) composé des états et transitions utiles est représenté par le graphe suivant :*



Q18. Donner, sans la justifier, une expression régulière ou ensembliste représentant les langages $L(\mathcal{E}_1)$ et $L(\mathcal{E}_2)$ sur $X = \{a, b, c\}$ accepté par l'automate $\mathcal{E}_1$ de **l'Exemple III.1** et $\mathcal{E}_2$ de **l'Exemple III.2**, situés sur la page 6.

Nous étudierons deux implémentations des automates : d'une part en algorithmique récursive programmée en langage CaML ; d'autre part en algorithmique itérative programmée en langage Python.

2.4 Algorithmique récursive (option Informatique)

Nous utiliserons par la suite en CaML le type `int` pour représenter les états d'un automate.

Q19. Écrire en CaML une définition pour les types `etat`, `transition` et `automate` qui représentent les états et les transitions ainsi que les automates sur l'alphabet des symboles. Définir en CaML la valeur `expl_III_1` de type `automate` correspondant à l'automate de l'**Exemple III.1** de la page 6.

Q20. Écrire en CaML une fonction `valider`, de type `automate -> bool`, telle que l'appel `(valider a)` sur l'automate `a` renvoie la valeur `true` si l'automate `a` est valide, c'est-à-dire s'il respecte les contraintes de la **Définition III.2** de la page 5, et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois les transitions de l'automate `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Q21. Écrire en CaML une fonction `accepter`, de type `mot -> automate -> bool`, telle que l'appel `(accepter m a)` sur un mot `m` et un automate `a`, renvoie la valeur `true` si le mot `m` appartient au langage accepté par l'automate `a` et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le mot `m`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Q22. Calculer une estimation de la complexité de la fonction `accepter` en fonction du nombre de symboles du mot `m` et du nombre de transitions de l'automate `a`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

2.5 Algorithmique itérative (Informatique pour tous)

Nous utiliserons par la suite en Python le type `int` pour représenter les états d'un automate.

Q23. Proposer une représentation en Python des automates sur l'alphabet des symboles. Définir en Python la variable `expl_III_1` initialisée avec une valeur correspondant à l'automate de l'**Exemple III.1** de la page 6.

Q24. Écrire en Python une fonction `valider`, telle que l'appel `valider(a)` sur l'automate `a` renvoie la valeur `true` si l'automate `a` est valide, c'est-à-dire s'il respecte les contraintes de la **Définition III.2** de la page 5, et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois les transitions de l'automate `a`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.

Q25. Écrire en Python une fonction `accepter`, telle que l'appel `accepter(m,a)` sur un mot `m` et un automate `a`, renvoie la valeur `true` si l'automate `a` accepte le mot `m` et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois le mot `m`. Cette fonction devra être itérative ou faire appel à des fonctions auxiliaires itératives.

3 Synchronisation de mots

3.1 Définition et Propriétés

Nous définissons d'abord l'opération de synchronisation de mots qui produit un langage.

Définition III.5 (synchronisation de mots) : *soient m_1 et m_2 deux mots de $X^\star$ et $S \subseteq X$ un alphabet de synchronisation, l'opération de synchronisation de m_1 et m_2 sur S, notée $m_1 \|_S m_2$, produit un langage sur X tel que : pour tous mots m, m_1, m_2 de $X^\star$, pour tous symboles s, s_1, s_2 de S avec $s_1 \neq s_2$ et pour tous symboles x, x_1, x_2 de $X \setminus S$, nous avons :*

$$
\begin{aligned}
&m_1 \|_S m_2 = m_2 \|_S m_1 \\
&\epsilon \|_S \epsilon = \{\epsilon\} \\
&(s.m) \|_S \epsilon = \emptyset \\
&(x.m) \|_S \epsilon = x.(m \|_S \epsilon) \\
&(s.m_1) \|_S (s.m_2) = s.(m_1 \|_S m_2) \\
&(s_1.m_1) \|_S (s_2.m_2) = \emptyset \\
&(s.m_1) \|_S (x.m_2) = x.((s.m_1) \|_S m_2) \\
&(x_1.m_1) \|_S (x_2.m_2) = x_1.(m_1 \|_S (x_2.m_2)) \cup x_2.((x_1.m_1) \|_S m_2).
\end{aligned}
$$

Q26. Soient les alphabets $X = \{a, b, c, d\}$ et $S = \{b\}$, construire le langage $(a.b.c) \|_S (a.b.d)$ en détaillant chaque étape.

Q27. Montrer que pour tous mots m_1, m_2 de $X^\star$,

$$\epsilon \in m_1 \|_S m_2 \Leftrightarrow ((m_1 = \epsilon) \wedge (m_2 = \epsilon)).$$

Q28. Montrer que pour tout symbole s de S, pour tous mots m, m_1, m_2 de $X^\star$, il existe des mots m'_1, m'_2 de $X^\star$ tels que :

$$s.m \in m_1 \|_S m_2 \Leftrightarrow ((m_1 = s.m'_1) \wedge (m_2 = s.m'_2) \wedge (m \in m'_1 \|_S m'_2)).$$

Q29. Montrer que pour tout symbole x de $X \setminus S$, pour tous mots m, m_1, m_2 de $X^\star$, il existe des mots m'_1, m'_2 de $X^\star$ tels que :

$$x.m \in m_1 \|_S m_2 \Leftrightarrow ((m_1 = x.m'_1) \wedge (m \in m'_1 \|_S m_2)) \vee ((m_2 = x.m'_2) \wedge (m \in m_1 \|_S m'_2)).$$

Nous nous limiterons à l'étude d'une implantation en algorithmique récursive que nous programmerons à la fois en langage CaML et en langage Python.

3.2 Implantation récursive en langage CaML (option Informatique)

Q30. Écrire en CaML une fonction `synchro_mot`, de type `mot -> mot -> alphabet -> langage`, telle que l'appel `(synchro_mot m1 m2 s)` sur les mots `m1` et `m2` et sur l'alphabet de synchronisation `s`, renvoie le langage $\texttt{m1} \|_{\texttt{s}} \texttt{m2}$. L'algorithme utilisé ne devra parcourir qu'une seule fois les mots `m1` et `m2`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

3.3 Implantation récursive en langage Python (Informatique pour tous)

Q31. Écrire en Python une fonction `synchro_mot`, telle que l'appel `synchro_mot(m1,m2,s)` sur les mots `m1` et `m2` et sur l'alphabet de synchronisation `s`, renvoie le langage $\texttt{m1} \|_{\texttt{s}} \texttt{m2}$. L'algorithme utilisé ne devra parcourir qu'une seule fois les mots `m1` et `m2`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

4 Synchronisation de langages

Soit l'opération interne de synchronisation sur S des langages définie par :

Définition III.6 (synchronisation de langages) : *soient L_1 et L_2 deux langages sur l'alphabet X et $S \subseteq X$ un alphabet de synchronisation, le langage $L_1 \|_S L_2$ sur X résultant de la synchronisation sur S des mots de L_1 et L_2 est défini par :*

$$L_1 \|_S L_2 = \bigcup_{m_1 \in L_1, m_2 \in L_2} m_1 \|_S m_2.$$

Q32. Écrire en CaML une fonction `synchro_lang`, de type `langage -> langage -> alphabet -> langage`, telle que l'appel `(synchro_lang l1 l2 s)` sur les langages `l1` et `l2` et sur l'alphabet de synchronisation `s`, renvoie le langage $\texttt{l1} \|_{\texttt{s}} \texttt{l2}$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

5 Synchronisation d'automates

5.1 Définition

Soit l'opération interne sur les automates finis déterministes définie par :

Définition III.7 (synchronisation d'automates) : *soient* $\mathcal{A}_1 = (Q_1, X, I_1, T_1, \gamma_1)$ *et* $\mathcal{A}_2 = (Q_2, X, I_2, T_2, \gamma_2)$ *deux automates finis déterministes sur l'alphabet* X *et* $S \subseteq X$ *un alphabet de synchronisation, l'automate qui résulte de la synchronisation sur* S *de* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *est* $\mathcal{A} = (Q_1 \times Q_2, X, I_1 \times I_2, T_1 \times T_2, \gamma_{\mathcal{A}})$ *où la relation de transition* $\gamma_{\mathcal{A}}$ *est définie par : pour tous états* o_1, d_1 *de* Q_1 *et* o_2, d_2 *de* Q_2*, pour tout symbole* s *de* S *et pour tout symbole* x *de* $X \setminus S$ *:*

$$((o_1, o_2), s, (d_1, d_2)) \in \gamma_{\mathcal{A}} \Leftrightarrow ((o_1, s, d_1) \in \gamma_1 \wedge (o_2, s, d_2) \in \gamma_2)$$

$$((o_1, o_2), x, (d_1, d_2)) \in \gamma_{\mathcal{A}} \Leftrightarrow ((o_1, x, d_1) \in \gamma_1 \wedge o_2 = d_2) \vee (o_1 = d_1 \wedge (o_2, x, d_2) \in \gamma_2).$$

Q33. En considérant l'**Exemple III.1** et l'**Exemple III.2** de la page 6, construire l'automate $\mathcal{E}_1 \|_S \mathcal{E}_2$ avec $S = \{b\}$. Le résultat devra être émondé (seuls les états et les transitions utiles devront être construits).

Q34. Écrire en CaML une fonction `synchro_auto`, de type `automate -> automate -> alphabet -> automate`, telle que l'appel `(synchro_auto a1 a2 s)` sur les automates `a1` et `a2` et sur l'alphabet de synchronisation `s`, renvoie l'automate $\texttt{a1} \|_{\texttt{s}} \texttt{a2}$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

5.2 Propriétés

Q35. Montrer que si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont des automates finis, alors $\mathcal{A}_1 \|_S \mathcal{A}_2$ est un automate fini.

Q36. Montrer que pour tout mot m de $X^\star$, il existe des mots m_1, m_2 de $X^\star$ tels que : pour tous états o_1, d_1 de Q_1 et o_2, d_2 de Q_2 :

$$((o_1, o_2), m, (d_1, d_2)) \in \gamma_{\mathcal{A}}^\star \Leftrightarrow (m \in m_1 \|_S m_2 \wedge (o_1, m_1, d_1) \in \gamma_1^\star \wedge (o_2, m_2, d_2) \in \gamma_2^\star).$$

Q37. Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates finis, montrer que :

$$L(\mathcal{A}_1 \|_S \mathcal{A}_2) = L(\mathcal{A}_1) \|_S L(\mathcal{A}_2).$$

FIN

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Logique, ensembles et applications
- Listes et algorithmes de recherche (séquentielle, dichotomique)
- Spécifier, tester, prouver : terminaison et correction
- Complexité des algorithmes itératifs et récursifs
- Langage OCaml : expressions, types, filtrage, récursivité
- Bases de Python : expressions, types, instructions, fonctions

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/mp