ENSI 2003 — Informatique

Partie I : Logique et calcul des propositions

Question I.1 La règle « une seule des affirmations est vraie »se traduit par la formule :

$$(\overline{A}_1 \wedge A_2 \wedge A_3) \vee (A_1 \wedge \overline{A}_2 \wedge A_3) \vee (A_1 \wedge A_2 \wedge \overline{A}_3)$$

Question I.2 A_1 se traduit par $B \Rightarrow N$ ou $\overline{B} \vee N$; A_2 par $B \wedge N$ et A_3 par N.

Question I.3 $\overline{A}_1 \wedge A_2 \wedge A_3$ s'écrit $(B \wedge \overline{N}) \wedge (B \wedge N) \wedge N$ ce qui est une contradiction car contenant $\overline{N} \wedge N$.

$A_1 \wedge A_2 \wedge \overline{A}_3$ s'écrit $(\overline{B} \vee N) \wedge (B \wedge N) \wedge \overline{N}$ ce qui est évidemment aussi une contradiction.

La règle est donc équivalente à $(A_1 \wedge \overline{A}_2 \wedge A_3$ donc à $(\overline{B} \vee N) \wedge (\overline{B} \vee \overline{N}) \wedge N$; par application de la distributivité de $\vee$ par rapport à $\wedge$ elle est donc équivalente à $(\overline{B} \vee (N \wedge \overline{N})) \wedge N$ et finalement à $\overline{B} \wedge N$. Il faut donc poser les deux pieds sur la dalle noire.

Question I.4 A_1 s'écrit $\neg(P \Rightarrow G)$ soit $P \wedge \overline{G}$; A_2 s'écrit $\overline{M} \wedge G$ et A_3 s'écrit $\overline{G} \wedge \overline{P}$ ou $\overline{G} \wedge \overline{M}$ ou $\overline{G}$ suivant ce qui est illisible sur le texte.

On fait une table de vérité :

P	M	G	$P \wedge \overline{G}$	$\overline{M} \wedge G$	$\overline{G} \wedge \overline{P}$	$\overline{G} \wedge \overline{M}$	$\overline{G}$
0	0	0	0	0	1	1	1
0	0	1	0	1	0	0	0
0	1	0	0	0	1	0	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	1	1
1	0	1	0	1	0	0	0
1	1	0	1	0	0	0	1
1	1	1	0	0	0	0	0

L'examen de cette table montre que la règle (deux des affirmations sont vraies) n'est vérifiée que sur les cinquième et septième lignes ; On voit également que le symbole caché ne peut être que M ou G, et que si on choisit la cinquième ligne la porte s'ouvrira dans les deux cas : il faut donc appuyer sur le petit pavé.

Remarque : on peut aussi interpréter l'énoncé en disant qu'il ne doit y avoir qu'une seule solution donc cela exclut le cas $\overline{G}$ et il reste la même solution.

Partie II : Algorithmique et programmation en CaML

Question II.1 Une expression rationnelle du langage reconnu par l'automate donné est $a + a^*b$.

Question II.2 On peut définir la fonction `cardinal` par

`let cardinal = string_length` ou par :

```
let rec cardinal = function
  | [] -> 0
  | t::r -> 1 + cardinal r;;
```

Question II.3 Il est clair que le nombre d'appels récursifs dans l'appel `cardinal e` est égal au cardinal de `e`.

Question II.4 On peut définir la fonction `appartient` par

`let appartient = mem` ou par :

```
let rec appartient v = function
  | [] -> false
  | t::r -> v=t || appartient v r;;
```

Question II.5 Il est là aussi clair que le nombre maximal d'appels récursifs est égal à `1+|e|` et que ce nombre est atteint quand `v` n'est pas dans `e`.

Question II.6 On définit `ajout` par :

```
let ajout v e =
  if appartient v e then e else v::e ;;
```

Cette fonction ajoute bien `v` à `e` s'il ne s'y trouve pas déjà.

Question II.7 Définition de `egalite` :

```
let egalite e1 e2 =
    let rec aux = function
      | [] -> true
      | t::r -> appartient t e2 && aux r
    in cardinal e1 = cardinal e2 && aux e1;;
```

Question II.8 La fonction récursive `aux` teste si un ensemble est inclus dans `e2`, ce qui se vérifie immédiatement par récurrence : `t::r` est inclus dans `e2` si et seulement si `t` appartient à `e2` et `r` est inclus dans `e2`.

Ensuite `e1` est égal à `e2` si et seulement si `e1` est inclus dans `e2` et si `e1` et `e2` ont même nombre d'éléments.

Question II.9 L'appel `appartient t e2` nécessite au plus `|e2|` appels récursifs ; dans le pire cas (quand `e1=e2`) cet appel est effectué pour tous les éléments de `e1` soit une complexité totale majorée par : `|e1|.|e2|`, le test du nombre d'éléments étant lui de complexité `|e1|+|e2|`.

Question II.10 Il y a une erreur dans l'énoncé sur le type de `union` qui doit renvoyer un *etats* et non un *boolean*.

La fonction `union` est définie par :

```
let rec union e1 = function
  | [] -> e1
  | t::r -> if appartient t e1 then union e1 r
           else t :: union e1 r;;
```

Elle parcourt récursivement `e2` et si ses éléments ne sont pas déjà dans `e1` elle les ajoute dans l'union.

Question II.11 La fonction `intersection` est définie par :

```
let rec intersection e1 = function
  | [] -> []
  | t::r -> if appartient t e1 then t :: intersection e1 r
            else intersection e1 r;;
```

Cette fonction parcourt récursivement `e2` et n'en garde que les éléments qui sont déjà dans `e1` ce qui produit bien l'intersection des deux ensembles.

Question II.12 La fonction suivants est définie par :

```
let rec suivants O gamma =
  let rec aux t = function
    | [] -> []
    | (o,_,d)::s -> if o = t then d :: aux t s else aux t s
  in match O with
    | [] -> []
    | t::r -> union (aux t gamma) (suivants r gamma);;
```

Question II.13 La fonction `aux` est définie ainsi : l'appel `aux t g` calcule les suivants de `t` dans la relation `g`. Pour cela elle parcourt récursivement la relation `g` en sélectionnant les transitions d'origine `t` et en ajoutant leur destination dans le résultat. Ensuite `suivants` est définie récursivement en remarquant que `suivants (t::r)` est égal à l'union des suivants de `t` (que l'on calcule par `aux t gamma`) et de ceux de `r` (que l'on calcule par `suivants r gamma`).

La terminaison des fonctions est assurée car les appels récursifs se font sur des ensembles dont le cardinal diminue de 1 à chaque fois.

Question II.14 1. D'après la définition de A_i, on a bien $A_i \subset A_{i+1}$, ce qui montre que la suite A_i est croissante.

2. Les A_i étant inclus dans l'ensemble fini Q, la suite A_i ne peut être strictement croissante ; il existe donc un $k \in \mathbb{N}$ tel que $A_k = A_{k+1}$.

3. Pour alléger les notations, γ étant fixée, on notera $\mathcal{A}(O)$ au lieu de $\mathcal{A}(O, \gamma)$ et $\mathcal{S}(O)$ au lieu de $\mathcal{S}(O, \gamma)$.

Soit $d \in \mathcal{A}(\mathcal{S}(A_i))$; il existe donc (par définition de $\mathcal{A}$) $o' \in \mathcal{S}(A_i)$ et $m \in X^*$ tels que $(o', m, d) \in \gamma^*$; il existe ensuite (par définition de $\mathcal{S}$) $o \in A_i$ et $e \in X$ tels que $(o, e, o') \in \gamma$; on en déduit que $(o, em, d) \in \gamma^*$, et donc que $d \in \mathcal{A}(A_i)$, ce qui prouve que $\mathcal{A}(\mathcal{S}(A_i)) \subset \mathcal{A}(A_i)$.

D'autre part, il est clair par définition de $\mathcal{A}$ que, pour deux sous-ensembles O_1 et O_2 de Q, on a $\mathcal{A}(O_1 \cup O_2) = \mathcal{A}(O_1) \cup \mathcal{A}(O_2)$; on en déduit donc : $\mathcal{A}(A_{i+1}) = \mathcal{A}(A_i) \cup \mathcal{A}(\mathcal{S}(A_i))$ et d'après l'inclusion prouvée précédemment : $\mathcal{A}(A_{i+1}) = \mathcal{A}(A_i)$.

4. Soit $E \subset Q$ tel que $\mathcal{S}(E) \subset E$ et $d \in \mathcal{A}(E)$; il existe $o \in E$ et $m \in X^*$ tels que $(o, m, d) \in \gamma^*$. Montrons par récurrence sur la longueur de m que $d \in E$: si m est le mot vide, par définition de $\mathcal{S}$ on a $d \in \mathcal{S}(E)$ et donc $d \in E$; sinon soit $m = em'$ avec $e \in X$ et $m' \in X^*$, alors (par définition de γ^*) il existe $q \in Q$ tel que $(o, e, q) \in \gamma$ et $(q, m', d) \in \gamma^*$; d'où $q \in \mathcal{S}(E) \subset E$ et donc par récurrence $d \in E$. D'où $\mathcal{E} \subset E$.

5. D'après 4 et 2, comme $\mathcal{S}(A_k) \subset A_{k+1} = A_k$ on a $\mathcal{A}(A_k) \subset A_k$; comme par définition de $\mathcal{A}$, on a l'inclusion inverse : $\mathcal{A}(A_k) = A_k$. D'après 3, on a $\mathcal{A}(O) = \mathcal{A}(A_0) = \mathcal{A}(A_k)$, d'où $\mathcal{A}(O) = A_k$.

Question II.15 Définition de `accessibles` :

```
let accessibles O gamma =
  let rec aux e =
    let e1 = union e (suivants e gamma) in
      if egalite e e1 then e else aux e1
  in aux O;;
```

Question II.16 La fonction `accessibles` calcule la suite des A_i jusqu'à en trouver deux consécutifs égaux auquel cas elle s'arrête et le renvoie. La fonction auxilliaire `aux` implémente la définition de A_{i+1} puis teste si $A_i = A_{i+1}$. La terminaison de la fonction et sa correction sont assurées par la question 14.

Question II.17 Définition de `prefixe` :

```
let rec prefixe o e = function
  | [] -> []
  | t::r -> (o,e,t) :: prefixe o e r;;
```

On pourrait aussi utiliser `map` :

```
let rec prefixe o e ds =
  map (function t -> (o,e,t)) ds;;
```

Cette fonction parcourt simplement la liste `ds` en remplaçant chaque état `t` par la transition `(o,e,t)`.

Question II.18 Définition de `decompose` :

```
let rec decompose = function
  | [] -> [], []
  | ((_, e, _) as t)::r -> let r1, r2 = decompose r in
      if e = epsilon then t::r1, r2 else r1, t::r2;;
```

Cette fonction parcourt la liste et trie les transitions suivant qu'elles sont arbitraires ou non.

Question II.19 L'énoncé devrait préciser que $\vec{\gamma}_\epsilon$ est la plus petite relation vérifiant la relation donnée.

Il est immédiat de voir que $\vec{\gamma}_\epsilon$ contient en plus de γ les transitions (q, ϵ, q) pour $q \in \{A, B, C, D, E\}$ et la transition (A, ϵ, C).

Question II.20 L'énoncé veut sans doute dire un majorant et non une borne supérieure.

Si $(q, \epsilon, q') \in \vec{\gamma}_\epsilon$ (avec $q \neq q'$) il existe une suite finie q_i d'états tels que $q_0 = q$, $q_n = q'$ et pour tout $i = 1, \ldots, n$, $(q_{i-1}, \epsilon, q_i) \in \gamma_\epsilon$. On peut supposer (en enlevant d'éventuelles boucles) que les états q_i sont distincts. Une telle chaîne de n transitions va produire $\binom{n+1}{2}$ éléments dans $\vec{\gamma}_\epsilon$. On peut donc en déduire qu'un majorant de la taille de $\vec{\gamma}_\epsilon$ est $\binom{|\gamma_\epsilon|+1}{2}+|\gamma_\epsilon|+1$ (en comptant les transitions d'un état sur lui-même).

Question II.21 Si l'on identifie (comme le suggère l'énoncé) ϵ et Λ, $\vec{\gamma}_\epsilon$ est égale à $(\gamma_\epsilon)^*$ (en prenant $X = \emptyset$) ; d'après la définition de $\mathcal{A}$ on a $\mathcal{A}(\{o\}) = \{d \in Q \mid (o, \epsilon, d) \in \gamma_\epsilon^*\}$ (le seul mot possible étant le mot vide) ; ceci entraîne évidemment : $\vec{\gamma}_\epsilon = \{(o, \epsilon, d) \mid o \in Q, d \in \mathcal{A}(\{o\}, \gamma_\epsilon)\}$.

Question II.22 Définition de la fonction `fermeture` :

```
let rec fermeture gamma = match gamma with
  | [] -> []
  | (o,_,_)::r -> union (prefixe o epsilon (accessibles [o] gamma))
                        (fermeture r);;
```

Question II.23 L'énoncé n'est pas très clair ; la fonction demandée n'est pas sensée calculer $\vec{\gamma}$ (ce que d'ailleurs elle ne peut pas faire puisqu'il faudrait connaître Q pour les (q, ϵ, q)) mais seulement « les transitions engendrées par la fermeture transitive de γ », ce qui n'est pas très clair ; faut-il y mettre seulement les nouvelles transitions ?

Dans le doute j'ai appliqué la formule de la question précédente, en se limitant aux états apparaissant à l'origine d'une transition de γ.

Question II.24 Chaque transition de $\gamma_X \triangleright \gamma_\epsilon$ provenant de la composée d'une transition de γ_X et d'une de γ_ϵ, la taille de $\gamma_X \triangleright \gamma_\epsilon$ est majorée par le produit des tailles de γ_X et de γ_ϵ.

Question II.25 On prolonge chaque transition de γ_X, quand c'est possible par une transition de $\vec{\gamma}_\epsilon$, et on obtient :

$\gamma_X \triangleright \gamma_\epsilon = \{(C,a,B),(C,a,C),(C,b,E),(A,a,D),(A,a,E)\}$.

Question II.26 Définition de la fonction `compose` :

```
let rec compose gamma1 gamma2 =
  let rec aux (o,e,q) = function
    | [] -> []
    | (q',_,d)::r -> if q'=q then (o,e,d)::(aux (o,e,q) r)
                              else aux (o,e,q) r
  in match gamma1 with
    | [] -> []
    | (o,e,q)::r -> union (aux (o,e,q) gamma2) (compose r gamma2);;
```

L'appel `aux (o,e,d) g` calcule les transitions obtenues en composant (o, e, d) avec une des transitions de la relation arbitraire g, ce qui se vérifie facilement par récurrence. Puis compose applique aux successivement à tous les éléments de γ_1 et prend l'union du tout.

Question II.27 Soit un mot u reconnu par l'automate A. Si u est le mot vide c'est qu'il existe un chemin dans A étiqueté uniquement par ϵ qui va d'un état initial i à un état terminal t, ce qui veut dire que $t \in \mathcal{S}(I, \vec{\gamma}_\epsilon)$ donc que t est aussi état initial de $\mathcal{E}(A)$ d'où $\mathcal{E}(A)$ reconnaît aussi le mot vide.

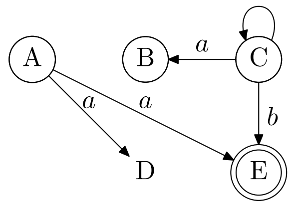
Si le mot n'est pas vide : $u = u_1 \dots u_n$, il existe une suite d'états q_i tels que : $q_0 \in I$, $q_n \in T$ et, pour tout $i = 1, \dots, n$, il existe q'_i tel que $(q_{i-1}, u_i, q'_i \in \gamma_X$ et $(q'_i, \epsilon, q_i) \in \vec{\gamma}_\epsilon)$, ce qui entraine que $(q_{i-1}, u_i, q_i) \in \gamma_X \triangleright \vec{\gamma}_\epsilon$, donc que le mot u est reconnu par $\mathcal{E}(A)$. La réciproque se fait de la même façon.

Les deux automates A et $\mathcal{E}(A)$ reconnaisssent donc le même langage.

Question II.28 En utilisant les questions 20 et 24 et en remarquant que $\gamma_X \subset \gamma_X \triangleright \vec{\gamma}_\epsilon$, on a comme majorant de la taille de $\gamma_X \triangleright \vec{\gamma}_\epsilon$: $|\gamma_X|.\Big(\binom{|\gamma_\epsilon|+1}{2} + |\gamma_\epsilon| + 1\Big)$.

La déterminisation d'un automate de taille n pouvant donner un automate de taille 2^n, la taille obtenue ici est meilleure.

Question II.29 L'automate semi-indéterministe associé à celui de la question II-1 est :



Question II.30 La fonction `semi` traduit seulement la définition de $\mathcal{E}(A)$ en utilisant les fonctions des questions précédentes :

```
let semi (Q, I, T, gamma) =
  let gammaX, gammaE = decompose gamma in
  let gammaF = fermeture gammaE in
  let gamma' = compose gammaX  gammaF in
  let I' = union I (suivants I gammaF)
  in (Q, I', T, gamma');;
```