Concours Communs Polytechniques 2004

Corrigé de l'épreuve d'informatique

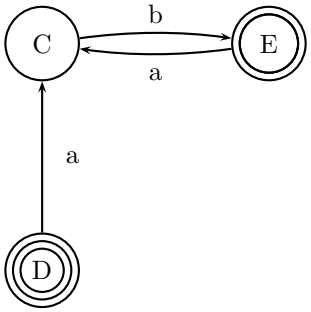
Partie I : Automates et langages

On remarquera que l'énoncé appelle "automate fini semi-indéterministe" ce que le programme appelle "automate fini", puis parle d'"expression régulière", traduction littérale de l'expression anglo-saxonne , au lieu d'"expression rationnelle". Enfin, il y a cinq erreurs dans l'énoncé, aux questions **I.5**, **II.19**, **II.25**, **II.32** et **II.34**.

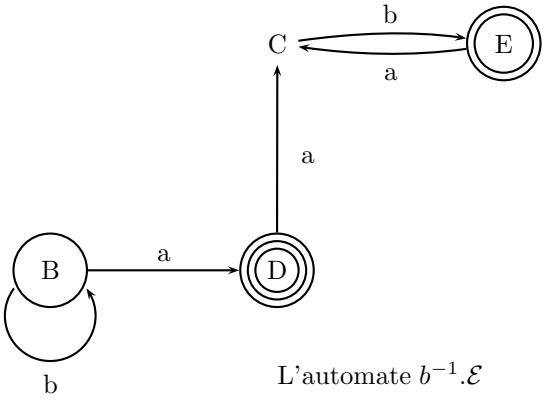
I.1. Le langage associé à $\mathcal{E}$ est, par exemple, représenté par l'expression rationnelle $(a+ba+b^*a^2)b(ab)^*+b+b^*a$.

I.2. En traduisant les définitions, nous obtenons directement :

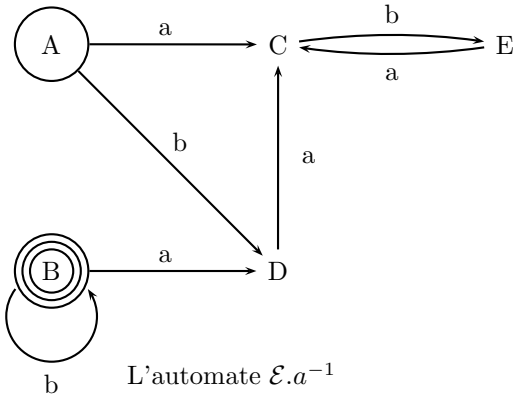
- Dans $a^{-1}\mathcal{E}$, les états initiaux sont C et D, et les états A et B ne sont pas accessibles :
- Dans $b^{-1}\mathcal{E}$, les états initiaux sont B et D, et l'état A n'est pas accessible :
- Dans $\mathcal{E}a^{-1}$, le seul état terminal est B et tous les états sont accessibles :
- Dans $\mathcal{E}b^{-1}$, les états terminaux sont A et C et tous les états sont accessibles :



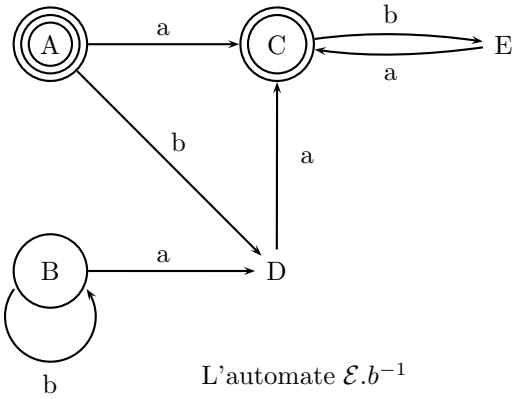
L'automate $a^{-1}.\mathcal{E}$



L'automate $b^{-1}.\mathcal{E}$



L'automate $\mathcal{E}.a^{-1}$



L'automate $\mathcal{E}.b^{-1}$

I.3. Expressions rationnelles associées à chacun de ces quatre automates :

- $a^{-1}\mathcal{E}$: $1 + (ab + b)(ba)^*$;
- $b^{-1}\mathcal{E}$: $(1 + ab^*)(ab)^*$;
- $\mathcal{E}a^{-1}$: b^* (l'état B est le seul état coaccessible) ;
- $\mathcal{E}b^{-1}$: $1 + (a + ba + b^*a^2)(ba)^*$.

I.4. Je ne comprends pas pourquoi la question est posée : $m^{-1}.\mathcal{A}$ et $\mathcal{A}.m^{-1}$ sont trivialement des automates fini semi-indéterministes puisque

- Q est un ensemble fini ;
- $I \subset Q$ et $\{q \in Q \mid \exists\, i \in I,\ (i, m, q) \in \gamma^*\} \subset Q$;
- $\{q \in Q \mid \exists\, t \in T,\ (q, m, t) \in \gamma^*\} \subset Q$ et $T \subset Q$;
- $\gamma \subset Q \times X \times Q$.

I.5. Cette question est fausse : le quantificateur $\forall\, q$ est évidemment aberrant. On demande en fait de montrer un résultat utilisé couramment dans le cours :

$$\forall\, m, n \in X^*,\ \forall\, o, d \in Q,\ \left(\exists\, q \in Q,\ (o, m, q) \in \gamma^* \wedge (q, n, d) \in \gamma^*\right) \iff (o, m.n, d) \in \gamma^*.$$

Soit donc $m, n \in X^*$ et $o, d \in Q$.

- Supposons qu'il existe $q \in Q$ tel que $(o, m, q) \in \gamma^*$ et $(q, n, d) \in \gamma^*$. Notons $n = e_1 e_2 \dots e_k$ où $e_1, \dots, e_k \in X$. En appliquant la définition de γ^*, on peut alors construire une suite $q_0, q_1, \dots, q_k$ d'états tels que $q_0 = q$, $q_k = d$ et $(q_i, e_i, q_{i+1}) \in \gamma$ pour tout i. En réappliquant la définition de γ^*, on montre (par récurrence) que $(o, m, q_0), (o, m.e_1, q_1), \dots, (o, m.e_1 e_2 \dots e_k, q_k)$ sont éléments de γ^*, et en particulier $(o, m.n, d) \in \gamma^*$.
- Réciproquement, supposons que $(o, m.n, d) \in \gamma^*$. En notant $m = e_1 e_2 \dots e_j$ et $n = e_{j+1} e_{j+2} \dots e_{j+k}$ où $e_1, \dots, e_{j+k} \in X$, il existe une suite $q_0, q_1, \dots, q_{j+k}$ d'états tels que $(q_i, e_i, q_{i+1}) \in \gamma$ pour tout i, avec $q_0 = o$ et $q_{j+k} = d$. Il suffit alors de poser $q = q_j$ pour avoir $(o, m, q) \in \gamma^*$ et $(q, n, d) \in \gamma^*$.

I.6. Soit $m, n \in X^*$. Comme les automates considérés ont la même relation de transition, on peut écrire :

$$\begin{aligned}
n \in L(m^{-1}.\mathcal{A}) &\iff \exists q \in Q,\ \exists\, i \in I,\ \exists\ t \in T,\ (i, m, q) \in \gamma^* \wedge (q, n, t) \in \gamma^* \\
&\iff \exists\, i \in I,\ \exists\ t \in T,\ \left(\exists q \in Q,\ (i, m, q) \in \gamma^* \wedge (q, n, t) \in \gamma^*\right) \\
&\iff \exists\, i \in I,\ \exists\ t \in T,\ (i, m.n, t) \in \gamma^* \\
&\iff m.n \in L(\mathcal{A})
\end{aligned}$$

$$\begin{aligned}
n \in L(\mathcal{A}.m^{-1}) &\iff \exists i \in I,\ \exists\, q \in Q,\ \exists\ i \in T,\ (q, m, t) \in \gamma^* \wedge (i, n, q) \in \gamma^* \\
&\iff \exists\, i \in I,\ \exists\ t \in T,\ \left(\exists q \in Q,\ (i, n, q) \in \gamma^* \wedge (q, m, t) \in \gamma^*\right) \\
&\iff \exists\, i \in I,\ \exists\ t \in T,\ (i, n.m, t) \in \gamma^* \\
&\iff n.m \in L(\mathcal{A})
\end{aligned}$$

Partie II : Algorithmique et programmation en CaML

II.1. On a, pour une valuation σ quelconque :

$$\hat{\sigma}(a \Rightarrow b) = 1 - \hat{\sigma}(a)(1 - \hat{\sigma}(b)) = 1 - (1 - \hat{\sigma}(\neg a))(1 - \hat{\sigma}(b)) = \hat{\sigma}(\neg a \vee b)$$

donc $a \Rightarrow b \equiv \neg a \vee b$.

II.2. On utilise ici les propriétés "évidentes" :

- pour $P_1, P_2, P_3 \in \mathcal{P}(\mathcal{F})$ avec $P_1 \equiv P_2$ et pour $op \in \{\wedge, \vee, \Rightarrow\}$, $P_1 \ op \ P_3 \equiv P_2 \ op \ P_3$;
- pour toute proposition P, $P \wedge \top \equiv P$ et $P \vee \bot \equiv P$.

On utilisera également l'écriture habituelle consistant à regrouper plusieurs $\wedge$ (resp. $\vee$) par associativité. Il faut remarquer que l'énoncé ne s'embarrasse pas des parenthèses, qu'elles soient inutiles ou non, puisque les propositions sont définies sans parenthésage !

Nous pouvons alors écrire :

$$\begin{aligned}
& (a \wedge c \Rightarrow \bot) \wedge (a \Rightarrow b) \wedge (\top \Rightarrow b \vee d) \\
\equiv\ & (\neg(a \wedge c) \vee \bot) \wedge (\neg a \vee b) \wedge (\neg\top \vee b \vee d) \\
\equiv\ & (\neg a \vee \neg c) \wedge (\neg a \vee b) \wedge (b \vee d) = C \\
\equiv\ & (\neg a \wedge \neg a \wedge b) \vee (\neg a \wedge \neg a \wedge d) \vee (\neg a \wedge b \wedge b) \vee (\neg a \wedge b \wedge d) \vee (\neg c \wedge b \wedge b) \vee (\neg c \wedge b \wedge d) \\
\equiv\ & (\neg a \wedge b) \vee (\neg a \wedge d) \vee (\neg a \wedge b) \vee (\neg a \wedge b \wedge d) \vee (\neg c \wedge b) \vee (\neg c \wedge b \wedge d) = D
\end{aligned}$$

et C et D sont respectivement des formes conjonctive et disjonctive.

II.3. La méthode est élémentaire : si `E` est vide, on renvoie le booléen `false`. Sinon, `E=t::queue` et on compare `f` à `t` : en cas d'égalité, on renvoie le booléen `true` ; sinon, on applique récursivement la fonction `appartenance` à `f` et à la queue de la liste.

```
let rec appartenance f = function
     [] -> false
     | t::_ when t=f -> true
     | _::queue -> appartenance f queue ;;
```

II.4. À chaque appel récursif, la taille de la liste diminue d'une unité : dans le pire des cas, le nombre d'appel récursif sera égal à la taille de la liste `E`. Ce cas se produit chaque fois que le fait `f` n'est pas élément de `E`.

II.5. On commence par tester l'appartenance de `f` à `E` : en cas de réponse négative, on renvoie la liste `f::E`, et en cas de réponse positive, on renvoie `E`.

```
let ajout f E =
     if appartenance f E then
        E
     else
        f::E ;;
```

Il est également possible d'écrire une procédure n'utilisant pas la fonction `appartenance` :

```
let rec ajout f E = match E with
      [] -> [f]
      | t::queue -> if f=t then
                      E
                    else
                      t::ajout f queue ;;
```

Cette seconde procédure ajoute `f` en fin de liste, alors que la première l'ajoute en début de liste.

II.6. On travaille ici récursivement sur l'ensemble `E1` : si `E1` est vide, on renvoie le booléen `true`. Sinon, avec `E1=t::queue`, on teste l'appartenance de `t` à `E2` : en cas de réponse négative, on renvoie le booléen `false` ; en cas de réponse positive, on applique récursivement la procédure à `queue` et à `E2`.

```
let rec inclusion E1 E2 = match E1 with
      [] -> true
      | t::queue -> if appartenance t E2 then
                      inclusion queue E2
                    else
                      false ;;
```

II.7. Est-il demandé ici de faire une analyse fine du pire des cas ? Doit-on compter uniquement les appels récursifs à la fonction `inclusion` ou bien ceux aux deux fonctions `inclusion` et `appartenance` ? Je penche pour la deuxième solution, afin que le nombre calculé soit en rapport avec le temps de calcul de la procédure ! Pour être un peu précis, il faut différencier deux cas :

- si le nombre n d'éléments de `E1` est inférieur ou égal à $k+1$ où k est le nombre d'éléments de `E2`, il semble à première vue que le pire des cas soit obtenu quand tous les éléments de `E1` sauf le dernier sont dans `E2`, les éléments communs étant de plus "à la fin" de la liste `E2`. Cela donne donc quelque chose comme :
$$\texttt{E1} = [x_1; x_2; \ldots; x_{n-1}; x_n],\ \texttt{E2} = [y_1; y_2; \ldots; y_k]$$
où $1 \leq n-1 \leq k$, $x_n \notin$ `E2` et $\{x_1, \ldots, x_{n-1}\} = \{y_{k-n+2}, \ldots, y_k\}$.

  Il y aura alors $(k) + (k-1) + \cdots + (k-n+2) + (k+1)$ appels à la fonction `appartenance` : le dernier $k+1$ correspond à la recherche malheureuse de x_n. Enfin, il y a n appels à la fonction `inclusion`. Cela donne un nombre d'appels récursifs égal à $\dfrac{n(2k-n+5)}{2}$.
- si $n-1 > k$, le pire des cas est obtenu quand les k premiers éléments de `E1` sont les éléments de `E2` : le nombre d'appels à la fonction `appartenance` est alors : $(k)+(k-1)+\ldots+(1)+(k+1)$, le dernier terme correspondant à la recherche du $(k+1)$-ème élément de `E1`, qui n'est pas dans `E2`. Comme on fait $k+1$ appels à la fonction `inclusion`, cela donne un nombre total d'appels récursifs égal à $\dfrac{(k+4)(k+1)}{2}$ (on peut remarquer que ce nombre coïncide avec celui calculé précédemment quand $k = n-1$).

Le temps de calcul dans le pire des cas est donc :

$$T(n,k) = \begin{cases} \dfrac{n(2k-n+5)}{2} & \text{si } n \leq k+1 \\ \dfrac{(k+4)(k+1)}{2} & \text{sinon} \end{cases}.$$

Je ne pense pas que l'énoncé, déjà peu clair sur la définition du nombre d'appels récursifs, demande de démontrer proprement que ces valeurs correspondent au pire des cas !

II.8. Il suffit d'utiliser la fonction `inclusion`, ce qui donne :

```
let egalite E1 E2 =
      if inclusion E1 E2 then
         inclusion E2 E1
      else
         false ;;
```

II.9. On travaille une nouvelle fois récursivement sur l'ensemble `E1` : si `E1` est vide, on renvoie la liste vide. Sinon, avec `E1=t::queue`, on teste l'appartenance de `t` à `E2` : en cas de réponse positive, on renvoie `soustraction queue E2` et en cas de réponse négative, on renvoie `t::soustraction queue E2`.

```
let rec soustraction E1 E2 = match E1 with
      [] -> []
      | t::queue -> if appartenance t E2 then
                       soustraction queue E2
                    else
                       t::soustraction queue E2 ;;
```

II.10. On a directement (lois de Morgan et application de la question **1.**) :

$$\begin{aligned}\mathcal{I}(\gamma) &\equiv \neg\left(\top \wedge \bigwedge_{i\in I} h_i\right) \vee \left(\bot \vee \bigvee_{j\in J} c_j\right)\\ &\equiv \bot \vee \left(\bigvee_{i\in I} \neg h_i\right) \vee \bot \vee \left(\bigvee_{j\in J} c_j\right)\\ &\equiv \left(\bigvee_{i\in I} \neg h_i\right) \vee \left(\bigvee_{j\in J} c_j\right)\end{aligned}$$

en convenant que la disjonction d'un nombre nul de litéraux est l'élément neutre pour la disjonction, i.e. $\bot$ (convention d'ailleurs nécessaire pour que $\bot$ soit une clause).

II.11. Si I est non vide, une valuation σ vérifiant $\sigma(h_1) = 0$ donnera $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$. De même, si J est non vide, une valuation σ vérifiant $\sigma(c_1) = 1$ donnera $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$. Par contre, si I et J sont vides, $\mathcal{I}(\gamma) \equiv \bot$. La seule connaissance absurde est donc $\emptyset \vdash \emptyset$.

II.12. Il suffit d'utiliser la fonction `egalite` :

```
let comparaison (H1,C1) (H2,C2) =
      if egalite H1 H2 then
         egalite C1 C2
      else
         false ;;
```

II.13. On a vu à la question **10.** que chaque $\mathcal{I}(\gamma_k)$ était équivalente à une clause P_k. On a donc :

$$\mathcal{I}(\Gamma) \equiv \bigwedge_{k\in K} P_k = C$$

en convenant que la conjonction d'un nombre nul de formules est l'élément neutre pour la conjonction, i.e. $\top$ (convention d'ailleurs nécessaire car quand K est vide, $\mathcal{I}(\Gamma) = \top$ doit être une forme conjonctive).

II.14. • choisissons une valuation σ telle que $\sigma(a) = \sigma(b) = 1$ et $\sigma(c) = 0$ (une telle valuation existe, en supposant évidemment que a, b et c sont deux à deux distinctes). Alors $\hat{\sigma}(\mathcal{I}(\gamma_1)) = 0$ et γ_1 n'est pas une tautologie.

• $\mathcal{I}(\sigma_2) \equiv \neg a \vee a \equiv \top$ donc γ_2 est une tautologie.

• de même, $\mathcal{I}(\sigma_3) \equiv \neg b \vee b \vee c \equiv \top$ donc γ_3 est une tautologie.

• ici encore, $\mathcal{I}(\sigma_4) \equiv \neg a \vee \neg c \vee c \equiv \top$ donc γ_4 est une tautologie.

• choisissons une valuation σ telle que $\sigma(b) = 1$. Alors $\hat{\sigma}(\mathcal{I}(\gamma_5)) = 0$ et γ_5 n'est pas une tautologie.

• choisissons une valuation σ telle que $\sigma(c) = 0$. Alors $\hat{\sigma}(\mathcal{I}(\gamma_6)) = 0$ et γ_6 n'est pas une tautologie.

II.15. Une connaissance $((h_i)_{i\in I}, (c_j)_{j\in J})$ est une tautologie si et seulement si une des hypothèses est égale à l'une des conclusions : $\exists\ (i,j) \in I \times J,\ h_i = c_j$. En effet :

- s'il existe $i \in I$ et $j \in J$ tel que $h_i = c_j$, on a :

$$\mathcal{I}(\gamma) \equiv \left(\bigwedge_{\substack{m\in I \\ m\neq i}} \neg h_m \right) \vee \underbrace{\neg h_i \vee h_i}_{\equiv \top} \vee \left(\bigwedge_{\substack{n\in J \\ m\neq j}} c_n \right) \equiv \top$$

  donc $\mathcal{I}(\gamma)$ est une tautologie.

- supposons maintenant qu'aucune hypothèse ne soit une conclusion. Soit alors σ la valuation définie par

$$\forall\, f \in \mathcal{F},\ \sigma(f) = \begin{cases} 1 & \text{si } f \in \{h_i,\ i \in I\} \\ 0 & \text{sinon.} \end{cases}$$

  On a alors $\sigma(c_j) = 0$ pour tout $j \in I$ et $\sigma(h_i) = 1$ pour tout $i \in I$, donc $\hat{\sigma}(\mathcal{I}(\gamma)) = 0$ et $\mathcal{I}(\gamma)$ n'est pas une tautologie.

II.16. On a clairement $\mathcal{I}(\Gamma \cup \{\gamma\}) = \mathcal{I}(\Gamma) \wedge \mathcal{I}(\gamma) \equiv \mathcal{I}(\Gamma) \wedge \top \equiv \mathcal{I}(\Gamma)$ donc $\Gamma \cup \{\gamma\} \equiv \Gamma$.

II.17. Cette fonction ne présente toujours aucune difficulté et est construite comme la fonction `soustraction`:

```
let rec elimination = function
     [] -> []
     | gamma::queue -> if tautologie gamma then
                          elimination queue
                       else
                          gamma::elimination queue ;;
```

II.18. Les relations qui existent entre ces connaissances sont :

$$\begin{cases} \gamma_2 \prec \gamma_3,\ \gamma_1 \prec \gamma_3,\ \gamma_2 \prec \gamma_4 & \\ \gamma_i \prec \gamma_5 & \text{pour tout } i \in \{1,2,3,4\} \\ \gamma_i \prec \gamma_i & \text{pour tout } i \in \{1,2,3,4,5\} \end{cases}$$

II.19. Tout d'abord, supposons que $\gamma_2 \prec \gamma_1$. Comme :

$$\mathcal{I}(\gamma_1) = \left(\bigvee_{h\in H_1} \neg h \right) \vee \left(\bigvee_{c\in C_1} c \right) \text{ et } \mathcal{I}(\gamma_2) = \left(\bigvee_{h\in H_2} \neg h \right) \vee \left(\bigvee_{c\in C_2} c \right)$$

nous avons, pour σ valuation quelconque :

$$\begin{array}{rcll}\hat{\sigma}(\mathcal{I}(\gamma_1)) = 1 & \iff & \exists h \in H_1,\ \sigma(h) = 0 \vee \exists\, c \in C1,\ \sigma(c) = 1 & \\ & \implies & \exists h \in H_2,\ \sigma(h) = 0 \vee \exists\, c \in C2,\ \sigma(c) = 1 & \text{car } H_1 \subset H_2 \text{ et } C_1 \subset C_2 \\ & \implies & \hat{\sigma}(\mathcal{I}(\gamma_2)) = 1 & \end{array}$$

On en déduit que :

$$\begin{array}{rcl}\hat{\sigma}(\mathcal{I}(\{\gamma_1, \gamma_2\})) = 1 & \iff & \hat{\sigma}(\mathcal{I}(\gamma_1) \wedge \mathcal{I}(\gamma_2)) = 1 \\ & \iff & \hat{\sigma}(\mathcal{I}(\gamma_1)) = \hat{\sigma}(\mathcal{I}(\gamma_2)) = 1 \\ & \iff & \hat{\sigma}(\mathcal{I}(\gamma_1)) = 1 \end{array}$$

donc $\mathcal{I}(\gamma_1) \wedge \mathcal{I}(\gamma_2) \equiv \mathcal{I}(\gamma_1)$, soit $\{\gamma_1, \gamma_2\} \equiv \{\gamma_1\}$.

Par contre, la réciproque est fausse : $\gamma_1 = \{a\} \vdash \{a\}$ et $\gamma_2 = \{b\} \vdash \{b\}$ donnent $\mathcal{I}(\gamma_1) \wedge \mathcal{I}(\gamma_2) \equiv \mathcal{I}(\gamma_1) \equiv \top$ sans que l'on ait $\gamma_2 \prec \gamma_1$.

Je pense que l'énoncé a oublié de préciser que les connaissances considérées n'étaient pas des tautologies (ce qui est naturel, puisque la partie **5** a permis d'éliminer les tautologies). Supposons donc que γ_1 et γ_2 ne soient pas des tautologies et que l'on ait $\gamma_2 \not\prec \gamma_1$. Deux cas sont alors possibles :

- s'il existe $h_0 \in H_1 \setminus H_2$, considérons la valuation σ qui associe 1 aux éléments de H_2 et 0 à tous les autres faits. En particulier, $\sigma(h_0) = 0$ et $\sigma(c) = 0$ pour tout $c \in C_2$ (car γ_2 n'est pas une tautologie). Ainsi :

$$\begin{array}{rcl}\hat{\sigma}\left(\mathcal{I}(\gamma_1)\right) & = & \hat{\sigma}\left(\left(\bigvee_{h \in H_1} \neg h\right) \vee \left(\bigvee_{c \in C1} c\right)\right) = 1 \\ \hat{\sigma}\left(\mathcal{I}(\gamma_2)\right) & = & \hat{\sigma}\left(\left(\bigvee_{h \in H_2} \neg h\right) \vee \left(\bigvee_{c \in C2} c\right)\right) = 0 \end{array}$$

  et donc $\hat{\sigma}\left(\mathcal{I}(\{\gamma_1, \gamma_2\})\right) = 0 \neq 1 = \hat{\sigma}\left(\mathcal{I}(\{\gamma_1\})\right)$.

- s'il existe $c_0 \in C_1 \setminus C_2$, la conclusion est identique, en considérant la valuation σ qui associe 0 aux éléments de C_2 et 1 à tous les autres faits (on a cette fois $\sigma(c_0) = 1$ et $\sigma(h) = 1$ pour tout $h \in H_2$).

Les bases de connaissances $\{\gamma_1, \gamma_2\}$ et $\{\gamma_1\}$ ne sont donc pas équivalentes, ce qui achève la preuve :

Si γ_1 et γ_2 sont des connaissances et si γ_2 n'est pas une tautologie, les bases de connaissances $\{\gamma_1, \gamma_2\}$ et $\{\gamma_1\}$ sont équivalentes si et seulement si $\gamma_2 \prec \gamma_1$.

II.20. La procédure est une nouvelle fois calquée sur les précédentes.

```
let rec absorbable (H,C) = function
     [] -> false
     | (H1,C1)::queue -> if inclusion H1 H & inclusion C1 C then
                            true
                         else
                            absorbable (H,C) queue ;;
```

II.21. Les couples de connaissances auquel il est intéressant d'appliquer $\triangleright$ sont (γ_1, γ_4), (γ_2, γ_1), (γ_2, γ_4) et (γ_3, γ_2), qui donnent respectivement les bases :

$$\{\{a\} \vdash \{c, d\}\},\ \{\{a, b\} \vdash \{d, e\}\},\ \{\{b\} \vdash \{b, e\}, \{c\} \vdash \{c, e\}\},\ \{\{b, c\} \vdash \emptyset\}.$$

au premier rang, nous obtenons donc cinq nouvelles connaissances :

$$\{a\} \vdash \{c,d\},\ \{a,b\} \vdash \{d,e\},\ \{b\} \vdash \{b,e\},\ \{c\} \vdash \{c,e\}\} \text{ et } \{b,c\} \vdash \emptyset.$$

II.22. Pour $f \in H_1 \cap C_2$, notons γ_f la connaissance $(H_1 \setminus \{f\}) \cup H_2 \vdash C_1 \cup (C_2 \setminus \{f\})$. Comme $H_1 \cap C_2$ contient au moins deux éléments, il existe $a \in H_1 \cap C_2$ avec $a \neq f$: cet élément est donc commun à l'ensemble des hypothèses et à l'ensemble des conclusions de γ_f, ce qui prouve que γ_f est une tautologie.

Cela permet de ne considérer que les transformations $\gamma_1 \rhd \gamma_2$ pour lesquelles $H_1 \cap C_2$ est un singleton. Je ne vois pas pourquoi la question est posée à ce stade du problème, la complétion de Ω n'étant formalisée que dans les questions suivantes. L'idée est sans doute qu'en appliquant l'opérateur $\rhd$ sur Ω jusqu'à stabiliser la base, on construit la complétion Ω_1 : c'est la plus petite base contenant Ω et "stable par $\rhd$", dans le sens où $\gamma_1 \rhd \gamma_2 \subset \Omega_1$ pour $\gamma_1, \gamma_2 \in \Omega_1$. Si maintenant on stabilise Ω en n'appliquant $\rhd$ que sur les couples (γ_1, γ_2) pour lesquelles $H_1 \cap C_2$ est un singleton, on obtient une "plus petite" complétion Ω_2 : c'est la plus petite base contenant Ω et "(un peu moins) stable par $\rhd$", dans le sens où $\gamma_1 \rhd \gamma_2 \subset \Omega_1$ pour $\gamma_1, \gamma_2 \in \Omega_1$ tels que $H_1 \cap C_2$ soit un singleton. On a évidemment $\Omega_2 \subset \Omega_1$, mais on peut montrer que pour $\gamma \in \Omega_1$, ou bien γ est une tautologie, ou bien il existe une connaissance plus générale que γ dans Ω_2. Si on savait ce que l'on compte faire de ces complétions de bases, on serait alors convaincu que la base Ω_2 rend les mêmes services que Ω_1, le calcul de Ω_2 étant plus rapide (on évite de calculer un bon nombre de connaissances inutiles).

La preuve de la propriété avancée n'est pas évidente (mais je ne pense pas que le correcteur attende une preuve) : l'idée est qu'en restreignant $\rhd$, on perd à chaque pas des tautologies. Il faut donc voir ce que donne $\gamma_1 \rhd \gamma_2$ quand γ_1 ou γ_2 est une tautologie. Le lecteur courageux pourra démontrer :

Si $\gamma_1 = H_1 \vdash C_1$ est une tautologie et si $\gamma_2 = H_2 \vdash C_2$ vérifie $H_1 \cup C_2 \neq \emptyset$, alors tout élément de $\gamma_1 \rhd \gamma_2$ est ou bien une tautologie, ou bien moins général que γ_2 (selon que l'élément f choisi dans $H_1 \cup C_2$ est dans C_1 ou non).

Si $\gamma_2 = H_2 \vdash C_2$ est une tautologie et si $\gamma_1 = H_1 \vdash C_1$ vérifie $H_1 \cup C_2 \neq \emptyset$, alors tout élément de $\gamma_1 \rhd \gamma_2$ est ou bien une tautologie, ou bien moins général que γ_1 (selon que l'élément f choisi dans $H_1 \cup C_2$ est dans H_2 ou non).

Ainsi, en oubliant les tautologies, on oublie également des éléments moins généraux que certains déjà obtenus. Il faudrait maintenant voir ce que ces "nouveaux éléments oubliés" donnent par application de $\rhd$ ... le lecteur très courageux pourra terminer l'étude !

II.23. Il y a une nouvelle fois une petite ambiguïté dans la question : doit-on composer "à gauche" par γ_1, ou bien des deux côtés à la fois ? Après avoir fait la suite du problème, on comprend qu'il faut choisir la seconde possibilité. D'autre part, doit-on tenir compte de la conclusion de la question précédente ? Ce serait évidemment plus facile, mais dans le doute ... je donne deux procédures.

Description de l'algorithme "complet" :

> 1,2,3 et 4 : tout cela est dit dans l'énoncé : les deux paramètres sont une connaissance `gamma` et une base `Omega`, assujettis aux contraintes habituelles, et la fonction renvoie un objet de type `base`, obtenue en réunissant les bases `gamma` $\rhd \gamma_2$ pour γ_2 décrivant `Omega`.
>
> 5 et 6 : nous commençons par écrire deux fonctions auxiliaires :
>
> – la fonction `construction : connaissance -> connaissance -> faits -> base` renvoie, quand on l'applique à `(H1,C1) (H2,C2) E` avec `E` contenu dans `H1` $\cap$ `C2`, la base des connaissances de `(H1,C1)` $\rhd$ `(H2,C2)` associée aux différents éléments de `E`. Cette fonction est récursive : si `E` est vide, on renvoie la liste vide ; sinon, on applique la construction à la queue de la liste `E` et on ajoute la connaissance `(H,C)` déduite en utilisant la tête de `E`, par le biais de la fonction `ajoutB`.

– la fonction `triangle` qui, appliquée à deux connaissances `(H1,C1)` et `(H2,C2)` renvoie la base `(H1,C1)` $\triangleright$ `(H2,C2)`. On calcule l'intersection `E` de `H1` et `C2` et on applique la fonction construction à `(H1,C1) (H2,C2) E`.

L'analyse est ensuite relativement simple :

– si `Omega` est vide, on renvoie la liste vide ;

– si `Omega=gamma2::queue`, on applique la fonction `triangle` à `gamma` et `gamma2`, à `gamma2` et `gamma`, puis on réunit, par le biais de `unionB`, les deux bases ainsi obtenues et le résultat de l'application récursive de la procédure `composition` à `gamma` et `queue`.

7 : On montre par récurrence sur la longueur des listes que les procédures se terminent : par exemple, pour la fonction `construction`, il n'y a pas d'appel récursif quand `E` est vide, et quand `E` est non vide, l'appel récursif se fait sur la queue de la liste `E`, dont la longueur est égale à $|\,E\,|-1$.

Description de l'algorithme "simplifié" : on se contente d'appliquer la construction seulement dans le cas où `E` est un singleton. La fonction `construction` est alors inutile, son rôle étant simplement d'éviter d'utiliser une boucle dans la fonction `triangle` (je rappelle que l'énoncé interdit d'utiliser `while` et `for`).

II.24. L'analyse précédente donne :

```
let rec construction (H1,C1) (H2,C2) = function
     [] -> []
     | f::E -> let H = union (soustraction H1 [f]) H2 in      (* H et C sont simplement *)
               let C = union C1 (soustraction C2 [f]) in      (* utilisés pour faciliter la lecture *)
               ajoutB (H,C) (construction (H1,C1) (H2,C2) E);;


let triangle (H1,C1) (H2,C2) = let E=intersection H1 C2 in
     construction (H1,C1) (H2,C2) E;;

let rec composition (H1,C1) = function
     [] -> []
     | (H2,C2)::queue -> let omega1 = triangle (h1,c1) (h2,c2) in
                         let omega2 = triangle (h2,c2) (h1,c1) in
                         unionB (unionB omega1 omega2) (composition (h1,c1) queue);;
```

Pour la composition "simplifiée", il suffit de modifier la fonction `triangle` :

```
let triangle (H1,C1) (H2,C2) = match with intersection H1 C2
     f::[] -> [union (soustraction H1 [f]) H2, union C1 (soustraction C2 [f])]
     _ ->[];;
```

II.25. Il y a une faute de frappe dans l'énoncé : il faut évidemment lire $\gamma_1 = H_1 \vdash C_1$. Posons $H = (H_1 \setminus \{f\}) \cup H_2$, $C = C_1 \cup (C_2 \setminus \{f\})$ et $\gamma = H \vdash C$. Nous avons donc :

$$\gamma_1 \rhd \gamma_2 = \{\gamma\} \text{ et } \mathcal{I}(\{\gamma_1, \gamma_2\} \cup \gamma_1 \rhd \gamma_2) = \mathcal{I}(\gamma_1) \wedge \mathcal{I}(\gamma_2) \wedge \mathcal{I}(\gamma).$$

Il faut donc montrer que pour toute valuation σ, on a $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$ dès que $\hat{\sigma}(\mathcal{I}(\gamma_1)) = \hat{\sigma}(\mathcal{I}(\gamma_2)) = 1$. Supposons donc que $\hat{\sigma}(\mathcal{I}(\gamma_1)) = \hat{\sigma}(\mathcal{I}(\gamma_2)) = 1$. Nous avons donc :

$$\big[(\exists h \in H_1,\ \sigma(h) = 0) \text{ ou } (\exists c \in C_1,\ \sigma(c) = 1)\big] \text{ et } \big[(\exists h \in H_2,\ \sigma(h) = 0) \text{ ou } (\exists c \in C_2,\ \sigma(c) = 1)\big].$$

Nous pouvons distinguer trois cas :

- s'il existe $c \in C_1$ tel que $\sigma(c) = 1$, on a en particulier $c \in C$ et donc $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$;
- s'il existe $h \in H_2$ tel que $\sigma(h) = 0$, on a en particulier $h \in H$ et donc $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$;
- si les deux cas précédents ne se produisent pas, il existe $h \in H_1$ tel que $\sigma(h) = 0$ et il existe $c \in C_2$ tel que $\sigma(c) = 1$. Comme $\sigma(h) \neq \sigma(c)$, on a $c \neq h$: c et h ne peuvent donc pas être simultanément égaux à f : on en déduit que ou bien $c \neq f$ et $c \in C$, ou bien $h \neq f$ et $h \in H$. Dans les deux cas, nous avons $\hat{\sigma}(\mathcal{I}(\gamma)) = 1$.

Remarque : l'hypothèse $H_1 \cap C_2 = \{f\}$ ne sert à rien, puisque le résultat reste valable :

- quand $H_1 \cap C_2 = \emptyset$ ($\gamma_1 \rhd \gamma_2$ est vide) ;
- quand $|\, H_1 \cap C_2 \,| \geq 2$ ($\gamma_1 \rhd \gamma_2$ ne contient que des tautologies).

II.26. Le 1. est évident. Notons ensuite $\mathcal{F}_f$ l'ensemble des faits qui apparaissent effectivement dans les connaissances de Ω : comme Ω est finie, $\mathcal{F}_f$ est également fini. Par construction, les connaissances des Ω_k sont toutes dans l'ensemble fini :

$$\{H \rhd C \,|\, H \subset \mathcal{F}_f \text{ et } C \subset \mathcal{F}_f\}.$$

La suite $|\Omega_k|$ est donc croissante et majorée : cette suite d'entiers est donc stationnaire. On peut donc définir ℓ comme le plus petit entier k tel que $\Omega_k = \Omega_{k+1}$ (par inclusion, l'égalité des cardinaux est équivalente à l'égalité des parties). On a alors $\Omega_\ell = \Omega_{\ell+1}$, donc $\Omega_{\ell+1} = \Omega_{\ell+2}$ par construction, puis par récurrence $\Omega_k = \Omega_\ell$ pour tout $k \geq \ell$. On a enfin $\Omega = \Omega_0 \equiv \Omega_1 \equiv \ldots \equiv \Omega_\ell = \overline{\Omega}$, donc $\Omega \equiv \overline{\Omega}$.

II.27. Je ne sais pas si le concepteur du sujet veut que l'on se limite à l'application de la "règle" $\rhd$ aux connaissances telles que $H_1 \cap C_2$ est un singleton (voir fin de la question **22**). Heureusement, dans l'exemple donné,* on ne trouve jamais de cas où H_1 et C_2 ont deux faits en commun. Pour simplifier l'écriture, j'identifie un singleton $\{\gamma\}$ à son unique élément élément, ce qui permet d'écrire $\gamma_1 \rhd \gamma_2 = \gamma$ au lieu de $\gamma_1 \rhd \gamma_2 = \{\gamma\}$.

On ne peut composer au premier rang que les connaissances (γ_2, γ_1) et (γ_3, γ_2) :

$$\begin{cases} \gamma_2 \rhd \gamma_1 = \{a, b\} \vdash \{d, e\} = \gamma_4 \\ \gamma_3 \rhd \gamma_2 = \{b, c\} \vdash \emptyset = \gamma_5 \end{cases}$$

Au second rang, on a à nouveau deux possibilités :

$$\begin{cases} \gamma_3 \rhd \gamma_4 = \{a, b\} \vdash \{d\} = \gamma_6 \\ \gamma_5 \rhd \gamma_1 = \{a, b\} \vdash \{d\} = \gamma_6 \end{cases}$$

Aucune autre construction n'est maintenant possible : nous avons donc

$$\overline{\Omega} = \Big\{ \{a, b\} \vdash \{c, d\}\,,\ \{b, c\} \vdash \{e\}\,,\ \{e\} \vdash \emptyset\,,\ \{a, b\} \vdash \{d, e\}\,,\ \{b, c\} \vdash \emptyset\,,\ \{a, b\} \vdash \{d\} \Big\}$$

II.28. L'ensemble précédent ne contient aucune tautologie. Par contre, $\gamma_1 \prec \gamma_6$, $\gamma_2 \prec \gamma_5$ et $\gamma_4 \prec \gamma_6$ donc on peut réduire la base en :

$$\rrbracket\,]\, \overline{\Omega}\, [\, \llbracket = \Big\{ \{e\} \vdash \emptyset\,,\ \{b, c\} \vdash \emptyset\,,\ \{a, b\} \vdash \{d\} \Big\}$$

II.29. La fonction `deduction` permet de calculer Ω_{i+1} en fonction de Ω_i : on pourrait donc très facilement écrire une fonction `completion`, en appliquant `deduction` tant que l'ensemble augmente (cela nécessite simplement d'utiliser une fonction qui renvoie le nombre d'éléments d'une liste). Cependant, cela serait très maladroit, car la fonction `deduction` calcule l'ensemble des $\gamma_A \rhd \gamma_j$ pour $\gamma_i, \gamma_j \in \Omega$ avant de le "mélanger" à l'ensemble Ω. Nous allons donc travailler de la façon suivante :

- on commence par construire la fonction récursive `construire` de type `base -> base -> base`. En notant `Omega1` et `Omega2` les deux paramètres de cette fonction, l'analyse est la suivante :
  - si `Omega2` est vide, elle renvoie `Omega1` ;
  - si `Omega2=gamma::queue`, on note :

    `Nouveaux= soustractionB (composition gamma Omega1) (AjoutB gamma Omega1)`

    puis on applique récursivement la fonction `construire` à `Omega1`$\cup\{$`gamma`$\}$ et à `Queue`$\cup$`Nouveaux`.

  Si l'on travaillait itérativement, cela reviendrait à faire passer un à un les éléments de `Omega2` à `Omega1`, mais en ajoutant à chaque pas la liste des "nouveaux éléments" à `Omega2`. On remarquera que la construction "oublie" l'élément $\gamma \rhd \gamma$: si γ n'est pas une tautologie, $\gamma \rhd \gamma$ est vide ; sinon, $\gamma \rhd \gamma = \{\gamma\}$.
- il reste ensuite à appliquer la procédure `construire` aux bases `[]` et `Omega`.

II.30. Il suffit de traduire l'analyse précédente :

```
let completion Omega =
   let rec construire Omega1 = function
      []-> Omega1
      | gamma::queue -> let Omega3=ajoutB gamma Omega1 in
                        let Nouveaux = soustractionB (composition gamma Omega1) Omega3 in
                           construire Omega3 (unionB queue Nouveaux)
   in construire [] Omega;;
```

II.31. On a $]\!]\,\overline{\Omega}\,[\![\,[=]\!]\,\overline{\Omega}\,[\![=\left\{\{e\}\vdash\emptyset\,,\ \{b,c\}\vdash\emptyset\,,\ \{a,b\}\vdash\{d\}\right\}$, ce qui donne $\left\{\{e\}\vdash\emptyset\,,\ \{c\}\vdash\emptyset\,,\ \{a\}\vdash\emptyset\right\}$ en supprimant b de toutes les hypothèses et d de toutes les conclusions. Cette base est minimale, donc

$$\frac{\Omega}{V,F}=\left\{\{e\}\vdash\emptyset\,,\ \{c\}\vdash\emptyset\,,\ \{a\}\vdash\emptyset\right\}$$

II.32. Il y a encore une erreur dans l'énoncé : il faut sans doute lire $=$ à la place de $\equiv$. L'auteur souhaite donc vraisemblablement que l'on démontre que la réponse à une requête ne contient pas de tautologie. La preuve est simple : en notant $\Omega_1 =]\!]\,\overline{\Omega}\,[\![\,[$, nous pouvons déjà remarquer que Ω_1 ne contient pas de tautologie, puisque l'on vient de supprimer toutes les tautologies. L'intégration de V et F conserve cette propriété : si $H\vdash F\in\Omega_1$, $H\cap C$ est vide, et donc à plus forte raison $(H\setminus V)\cap(C\setminus F)$ l'est également. On en déduit donc que

$$\Omega_2=\{(H\setminus V)\vdash\cap(C\setminus F)\mid H\vdash C\in\Omega_1\}$$

ne contient aucune tautologie. Enfin, $\dfrac{\Omega}{V,F}\subseteq\Omega_2$, donc la réponse à la requête ne contient aucune tautologie.

II.33. On retrouve la même erreur, mais cette fois-ci, on ne peut pas remplacer $\equiv$ par $=$: en général, la réponse à une interrogation n'est pas stable par $\rhd$, comme le montre l'exemple suivant :

$$\begin{cases}\Omega=\{\{b,e\}\vdash\{e\},\{b,e\}\vdash\{a,d\},\{b,d\}\vdash\{a,c\},\{a,b\}\vdash\{c,d\}\}\\ V=\{a,c\}\\ F=\{b,e\}\end{cases}$$

donne $\dfrac{\Omega}{V,F}=\left\{\{b,e\}\vdash\{a,c\},\{b\}\vdash\{c,d\},\{b,d\}\vdash\{a,c\},\{b,e\}\vdash\{a,d\}\right\}$ et $\{b\}\vdash\{a,c\}\in\overline{\dfrac{\Omega}{V,F}}\setminus\dfrac{\Omega}{V,F}$.

Il faut vraisemblablement ajouter une hypothèse sur Ω, ce qui démontre une nouvelle fois l'ambiguïté d'une bonne partie de l'énoncé.

II.34. La seule construction non encore définie est l'intégration. Celle-ci se fait naturellement (et récursivement), comme fonction locale à `interrogation`:

```
let interrogation V F Omega =
      let rec integration = function
            [] -> []
            | (H,C)::queue -> (soustraction H V,soustraction C F)::integration queue in
      minimisation (integration (elimination (minimisation (completion Omega)))) ;;
```