INF105
Contrôle de rattrapage — Corrigé

Théorie des langages

22 mars 2018

Consignes.

Les exercices sont totalement indépendants. Ils pourront être traités dans un ordre quelconque, mais on demande de faire apparaître de façon très visible dans les copies où commence chaque exercice.

L'usage de tous les documents (notes de cours manuscrites ou imprimées, feuilles d'exercices, livres) est autorisé.

L'usage des appareils électroniques est interdit.

Durée : 1h30

Barème *indicatif* : $7 + 8 + 5$ points.

(La longueur de l'exercice 3 ne doit pas effrayer : les réponses attendues sont plus courtes que les questions puisqu'il s'agit de lire et de commenter des raisonnements déjà écrits.)

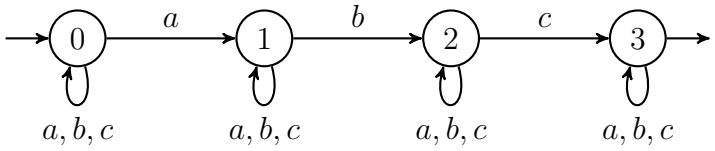
Ce corrigé comporte 8 pages (page de garde incluse)

Exercice 1.

Dans cet exercice, on pose $\Sigma = \{a, b, c\}$. On appelle L le langage des mots sur Σ qui ont abc comme sous-mot, c'est-à-dire, qui contiennent un a, un b et un c dans cet ordre mais non nécessairement consécutivement (à titre d'exemple, $abac \in L$).

(1) Proposer un automate non-déterministe (NFA), sans transition spontanée, $\mathscr{A}_1$ qui reconnaît le langage L. On justifiera rapidement pourquoi l'automate proposé convient.

Corrigé. On peut proposer l'automate $\mathscr{A}_1$ suivant :

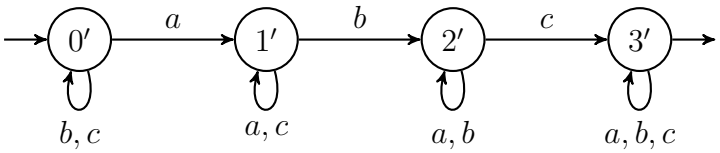


Pour se convaincre qu'il convient, on remarque qu'un chemin de l'unique état initial (0) à l'unique état final (3) dans cet automate est formé par trois transitions $0 \to 1$, $1 \to 2$ et $2 \to 3$, étiquetées par les lettres a, b et c respectivement, intercalées d'un nombre quelconque de transitions d'un état vers lui-même, étiquetées par une lettre quelconque : la lecture de ses étiquettes donne bien un mot ayant abc comme sous-mot ; et réciproquement, tout tel mot étiquette un chemin de 0 à 3 (une fois choisies les lettres a, b, c dans l'ordre qui correspondront aux transitions changeant d'état).

Remarque : il est correct de donner directement l'automate déterministe $\mathscr{A}_2$ représenté ci-dessous (puisque tout DFA est, en particulier, un NFA), à condition de justifier proprement qu'il convient. ✓

(2) Déterminiser (si nécessaire) l'automate $\mathscr{A}_1$ trouvé en (1) ; on appellera $\mathscr{A}_2$ l'automate résultant.

Corrigé. En notant $0' := \{0\}$, $1' := \{0, 1\}$, $2' := \{0, 1, 2\}$ et $3' := \{0, 1, 2, 3\}$ les états de l'automate déterministe $\mathscr{A}_2$ qui correspondent aux ensembles d'états de l'automate $\mathscr{A}_1$ qu'on vient d'indiquer, la déterminisation conduit à l'automate $\mathscr{A}_2$ suivant :



✓

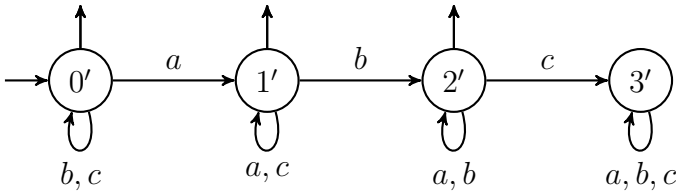
(3) Minimiser (si nécessaire) l'automate $\mathscr{A}_2$ obtenu en (2) ; on appellera $\mathscr{A}_3$ l'automate minimal (=canonique) résultant.

Corrigé. On part bien d'un automate $\mathscr{A}_2$ déterministe complet sans état inaccessible. La première étape de l'algorithme de minimisation sépare deux classes d'états : $0', 1', 2'$ (non finaux) d'une part et $3'$ (final) de l'autre. Ensuite on sépare $0', 1'$ d'une part et $2'$ de l'autre (car les premiers ne conduisent qu'à des états non finaux, tandis que $2'$ conduit à un état final par la transition étiquetée c). L'étape suivante sépare $0'$ et $1'$ (car le premier ne conduit qu'à un état de la classe $\{0', 1'\}$ de l'étape précédente, tandis que $1'$ conduit à un état de la classe $\{2'\}$ par la transition étiquetée b). On a donc séparé tous les états, ce qui montre que $\mathscr{A}_3 = \mathscr{A}_2$ est minimal.

Remarque : on pouvait aussi invoquer l'argument suivant : puisque le mot abc doit être accepté et qu'aucun de ses sous-mots stricts ne l'est (ce qui exclut qu'il y ait une boucle dans le chemin d'acceptation), il faut au moins $|abc| + 1 = 4$ états dans n'importe quel automate reconnaissant L. Comme $\mathscr{A}_2$ a effectivement 4 états, il est forcément minimal. ✓

(4) Construire un automate $\mathscr{A}_4$ reconnaissant le langage $M := \Sigma^* \setminus L$ complémentaire de L.

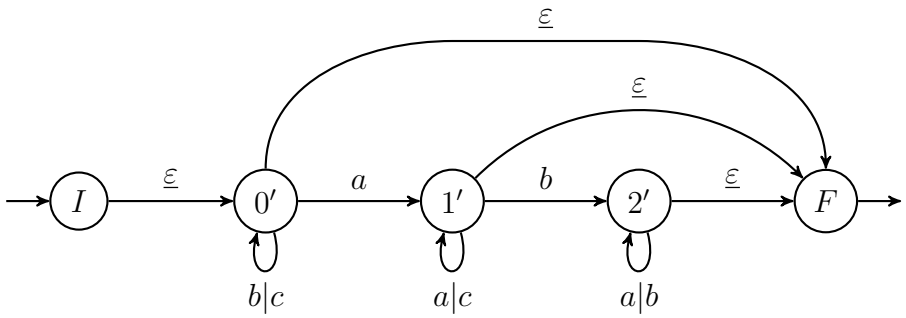
Corrigé. On obtient $\mathscr{A}_4$ en échangeant états finaux et non finaux dans $\mathscr{A}_3 = \mathscr{A}_2$, c'est-à-dire l'automate $\mathscr{A}_4$ suivant :



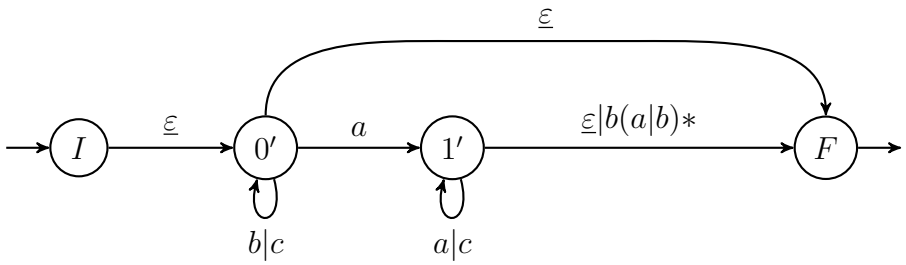
✓

(5) En appliquant à $\mathscr{A}_4$ la méthode d'élimination des états, déterminer une expression rationnelle dénotant le langage M. (Si on souhaite obtenir une expression plus compacte, il est recommandé de commencer l'élimination des états par ceux qui sont le plus loin de l'état initial.)

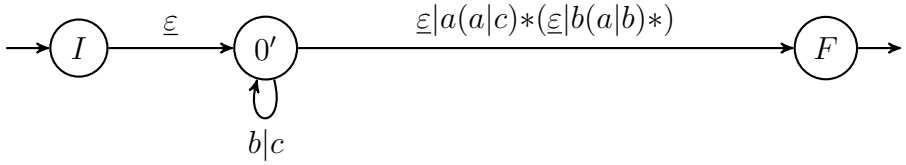
Corrigé. On va manipuler des automates finis à transitions étiquetées par des expressions rationnelles (ou « RNFA »). Dans un premier temps, on donne à l'automate un unique état final F en faisant pointer vers lui une transition spontanée (i.e., étiquetée par ε) depuis tout état qui était final, et un unique état initial I depuis lequel on fait partir une transition spontanée vers $0'$. L'état $3'$ peut être éliminé d'emblée puisqu'il est inutile (il n'est pas co-accessible : c'est un puits !). Par ailleurs, deux transitions entre les mêmes états sont remplacées par une seule étiquetée par la disjonction des étiquettes (par exemple, les deux transitions $0' \to 0'$ étiquetées b et c sont remplacées par une seule étiquetée $b|c$). On a donc affaire à l'automate suivant :



On élimine maintenant l'état $2'$:



(on a fait usage du fait évident que l'expression rationnelle $r\underline{\varepsilon}$ est équivalente à r). On élimine ensuite l'état $1'$:



Enfin, l'élimination de l'état $0'$ conduit au RNFA ayant seulement deux états I et F reliés par une transition étiquetée par

$$(b|c)*\Big(\underline{\varepsilon}|a(a|c)*\big(\underline{\varepsilon}|b(a|b)*\big)\Big)$$

qui est l'expression rationnelle recherchée (dénotant M).

L'élimination des états dans l'ordre $0', 1', 2'$ conduirait à l'expression rationnelle moins compacte

$$\underline{\varepsilon} \mid (b|c)* \mid (b|c)*a(a|c)* \mid (b|c)*a(a|c)*b(a|b)*$$

qui est cependant sans doute plus lisible.

Remarque : le langage M est aussi dénoté par l'expression rationnelle plus simple $(b|c)*\,(a|c)*\,(a|b)*$, mais celle-ci ne répond pas à la question car elle ne peut pas s'obtenir par élimination des états à partir de $\mathscr{A}_4$. ✓

Exercice 2.

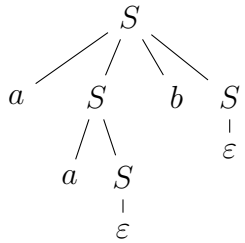
Dans cet exercice, on pose $\Sigma = \{a, b\}$. On appelle $L = L(G)$ le langage défini par la grammaire hors-contexte G d'axiome S et de nonterminaux $N = \{S\}$ sur l'alphabet Σ donnée par

$$S \to aSbS \mid aS \mid \varepsilon$$

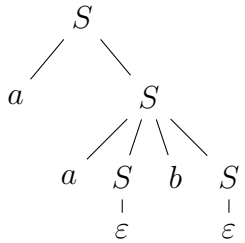
Les questions de cet exercice sont essentiellement indépendantes, si ce n'est que la question (6) fait appel à la conclusion (énoncée) des questions (2) à (5).

(1) Donner deux arbres d'analyse (pour G) différents du mot aab. Que peut-on dire de la grammaire G ?

Corrigé. On peut proposer les arbres d'analyse suivants :



d'une part, et



d'autre part

(en fait, on peut se convaincre que ce sont les seuls possibles). La grammaire G est donc ambiguë. ✓

L'objet des questions suivantes est de montrer que L n'est pas rationnel.

Soit M le langage $\{a^i b^j : i, j \in \mathbb{N}\}$ constitué des mots de la forme $a^i b^j$ avec i et j deux entiers naturels quelconques.

(2) Donner une expression rationnelle qui dénote M.

Corrigé. Le langage M est dénoté par l'expression rationnelle $a*b*$. ✓

Soit P le langage $\{a^i b^j : i \geq j\}$ constitué des mots de la forme $a^i b^j$ avec i et j deux entiers naturels vérifiant $i \geq j$ (autrement dit, les mots de M qui ont au moins autant de a que de b).

(3) Montrer que $P \subseteq L$.

Corrigé. Soient $i \geq j$ deux entiers naturels : on va expliquer comment produire le mot $a^i b^j$ selon les règles de G. À partir de l'axiome S, on commence par appliquer $i - j$ fois la règle $a \to aS$, ce qui donne $a \Rightarrow aS \Rightarrow aaS \Rightarrow \cdots \Rightarrow a^{i-j}S$. Appliquons maintenant j fois la règle $S \to aSbS$, suivie à chaque fois de $S \to \varepsilon$ sur le S de droite : on obtient ainsi $a^{i-j}S \Rightarrow a^{i-j+1}SbS \Rightarrow a^{i-j+1}Sb \Rightarrow a^{i-j+2}SbSb \Rightarrow a^{i-j+2}Sbb \Rightarrow \cdots \Rightarrow a^i Sb^j$. Il suffit de terminer par une application de $S \to \varepsilon$: on obtient ainsi une dérivation $S \Rightarrow^* a^i b^j$ qui prouve que $a^i b^j \in L$.

(On pouvait aussi, plus informellement, esquisser l'allure d'un arbre de dérivation de $a^i b^j$ un peu à la manière de l'un ou l'autre de ceux de la question (1), mais en appliquant $i - j$ fois la règle $a \to aS$ et j fois la règle $S \to aSbS$.) ✓

(4) Montrer que $L \cap M \subseteq P$.

Corrigé. Considérons la propriété « avoir (au total) au moins autant de a que de b », ou si on veut $|\xi|_a \geq |\xi|_b$, sur un pseudo-mot ξ (un « pseudo-mot » signifiant, par définition, un mot sur $\Sigma \cup N = \{a, b, S\}$). Cette propriété est vérifiée par le pseudo-mot S. Elle est préservée si on remplace S par $aSbS$ ou aS ou ε puisque dans chacun de ces cas le nombre de a augmente d'au moins autant que le nombre de b. Elle est donc possédée par tout pseudo-mot, et en particulier tout mot, qui dérive de S selon G. On vient donc de prouver que tout mot de L a au moins autant de a que de b. Notamment, tout mot de $L \cap M$ est un mot de la forme $a^i b^j$ (par définition de M), et qui vérifie $i \geq j$ (c'est ce qu'on vient de prouver). Il est donc dans P. ✓

(5) Montrer que P n'est pas rationnel.

Corrigé. Supposons par l'absurde que $P = \{a^i b^j : i \geq j\}$ soit rationnel. Appliquons le lemme de pompage pour les langages rationnels au langage P : appelons k l'entier dont le lemme de pompage garantit l'existence. Considérons le mot $t := a^k b^k$ (qui appartient bien à P et est bien de longueur $\geq k$) : il doit alors exister une factorisation $t = uvw$ (avec $u, v, w \in \Sigma^*$, non nécessairement dans P) pour laquelle on a (i) $|v| \geq 1$, (ii) $|uv| \leq k$ et (iii) $uv^i w \in P$ pour tout $i \geq 0$. La propriété (ii) assure que uv est formé d'un certain nombre de répétitions de la lettre a (car tout préfixe de longueur $\leq k$ de $a^k b^k$ est de cette forme) ; disons $u = a^\ell$ et $v = a^m$, si bien que $w = a^{k-\ell-m} b^k$. La propriété (i) donne $m \geq 1$. Enfin, la propriété (iii) affirme que le mot $uv^i w = a^{k+(i-1)m} b^k$ appartient à P ; or pour $i = 0$, ceci est faux puisque $a^{k-m} b^k$ vérifie $k - m < k$. On a donc abouti à une contradiction, et c'est que P n'est pas rationnel. ✓

(6) Déduire des questions (2) à (5) que L n'est pas rationnel (on explicitera très soigneusement le raisonnement).

Corrigé. On a vu à la question (3) que $P \subseteq L$; il est par ailleurs trivial que $P \subseteq M$, et on a donc $P \subseteq L \cap M$. Mais on a vu à la question (4) que $L \cap M \subseteq P$: on a donc $L \cap M = P$. Le langage M est rationnel d'après la question (2). Si le langage L était lui aussi rationnel, le langage $L \cap M$ (c'est-à-dire P) le serait car l'intersection de deux langages rationnels est rationnelle. Or on a vu à la question (5) que P n'est pas rationnel. C'est donc que L ne l'est pas. ✓

Exercice 3.

Dans cet exercice, on s'intéresse au langage L formé des programmes e (codés, dans une formalisation quelconque de la calculabilité [1], comme des entiers naturels ou comme des mots sur un alphabet fixé sans importance) qui, quel que soit le paramètre n qu'on leur fournit en entrée, terminent en temps fini et retournent la valeur 0 : soit $L = \{e : (\forall n)\varphi_e(n) \downarrow = 0\}$. Si l'on préfère, L

1. Par exemple, un langage de programmation (Turing-complet) quelconque.

est l'ensemble de toutes les façons de coder la fonction constante égale à 0. On appellera aussi M le complémentaire de L. On se demande si L ou M sont semi-décidables.

(1) Thésée et Hippolyte se disputent pour savoir si L est semi-décidable. Thésée pense qu'il l'est, et il tient le raisonnement suivant pour l'expliquer :

> Pour savoir si un programme e est dans l'ensemble L, il suffit de l'examiner pour vérifier que toutes les instructions qui mettent fin au programme renvoient la valeur 0 : si c'est le cas, on termine en répondait « oui » (c'est-à-dire $e \in L$) ; sinon, on rentre dans une boucle infinie. Ceci fournit un algorithme qui semi-décide L.

Hippolyte, elle, pense que L n'est pas semi-décidable. Son argument est le suivant :

> Si L était semi-décidable, je pourrais m'en servir pour résoudre le problème de l'arrêt. En effet, donné un programme e' et une entrée m sur laquelle je cherche à tester l'arrêt de e', je peux fabriquer le programme e qui prend une entrée n, lance l'exécution de e' sur l'entrée m pendant au plus n étapes et à la fin renvoie 1 si ces n étapes ont suffi à terminer l'exécution de e', et 0 sinon. Dire que $e \in L$ signifie que e' ne termine jamais sur m, donc pouvoir semi-décider L permet de résoudre algorithmiquement le problème de l'arrêt, ce qui est impossible.

Qui a raison ? Expliquer précisément quelle est l'erreur (ou les erreurs) commise(s) par le raisonnement incorrect, et détailler les éventuels passages incomplets dans le raisonnement correct.

Corrigé. C'est Hippolyte qui a raison (L n'est pas semi-décidable).

L'argument de Thésée est stupide : une instruction mettant fin au programme avec une valeur autre que 0 pourrait ne jamais être atteinte, et réciproquement, même si toutes les instructions mettant fin au programme renvoyaient 0, il pourrait aussi ne jamais terminer. (Au passage, l'argument de Thésée semblerait même montrer que L est décidable, pas juste semi-décidable.)

L'argument d'Hippolyte, lui, est correct : il montre que si L était semi-décidable, le complémentaire du problème de l'arrêt (l'ensemble des e' qui ne terminent jamais) serait semi-décidable, ce qui n'est pas le cas. (Le complémentaire du problème de l'arrêt n'est pas semi-décidable, car s'il l'était, le problème de l'arrêt serait décidable, vu qu'il est déjà semi-décidable.) Parmi les points qui méritent éventuellement d'être précisés, on peut mentionner : le fait que e se déduise algorithmiquement de e' et m ; et le fait qu'il est algorithmiquement possible de lancer l'exécution d'un programme sur n étapes (peu importe la définition exacte de « étape » tant que chacune termine à coup sûr en temps fini) et tester si elle est bien finie au bout de ce temps. ✓

(2) Achille et Patrocle se disputent pour savoir si M (le complémentaire de L)

est semi-décidable. Achille pense qu'il l'est, et il tient le raisonnement suivant pour l'expliquer :

> Pour savoir si un programme e est dans l'ensemble M, il suffit de tester successivement les valeurs $\varphi_e(n)$ pour tous les n possibles : si l'on rencontre un n tel que $\varphi_e(n)$ n'est pas 0, alors on termine en répondant « oui » (c'est-à-dire $e \in M$) ; sinon, on ne va jamais terminer, et cela signifie que $e \notin M$. Ceci fournit un algorithme qui semi-décide M.

Patrocle, lui, pense que M n'est pas semi-décidable. Son argument est le suivant :

> Si M était semi-décidable, je pourrais m'en servir pour résoudre le problème de l'arrêt. En effet, donné un programme e' et une entrée m sur laquelle je cherche à tester l'arrêt de e', je peux fabriquer le programme e qui prend une entrée n, l'ignore purement et simplement, exécute e' sur l'entrée m et à la fin renvoie 0. Dire que $e \in M$ signifie que e' ne termine pas sur m, donc pouvoir semi-décider M permet de résoudre algorithmiquement le problème de l'arrêt, ce qui est impossible.

Qui a raison ? Expliquer précisément quelle est l'erreur (ou les erreurs) commise(s) par le raisonnement incorrect, et détailler les éventuels passages incomplets dans le raisonnement correct.

Corrigé. C'est Patrocle qui a raison (M n'est pas semi-décidable).

Le raisonnement d'Achille se fonde sur l'idée que le complémentaire de « l'ensemble des programmes qui renvoient toujours la valeur 0 » est « l'ensemble des programmes qui renvoient parfois une valeur autre que 0 », ce qui oublie la possibilité que le programme ne termine pas. C'est là la principale erreur (le reste est globalement correct).

L'argument de Patrocle, lui, est correct : il montre que si M était semi-décidable, le complémentaire du problème de l'arrêt (l'ensemble des e' qui ne terminent jamais) serait semi-décidable, ce qui n'est pas le cas. (Le complémentaire du problème de l'arrêt n'est pas semi-décidable, car s'il l'était, le problème de l'arrêt serait décidable, vu qu'il est déjà semi-décidable.) On fait appel à des fonctions constantes, et qui ne peuvent renvoyer que 0 ou ne pas terminer, mais ce n'est pas spécialement problématique. Parmi les points qui méritent éventuellement d'être précisés, on peut mentionner le fait que e se déduise algorithmiquement de e' et m. ✓