

Devoir Surveillé 2 – Automates

Corrigé

Durée : 1h.

Ce devoir a été difficile à noter. Après examen des résultats des deux sujets, deux barèmes différents ont été utilisés ; après consultation de quelques enseignants, le dernier exercice a été converti en question bonus. Malgré plusieurs ré-évaluations, les notes sont très disparates, avec un fort écart-type et une moyenne de groupe à 12.1 (sans compter les absences). Quatre compétences étaient évaluées : savoir construire des automates, savoir interpréter des automates, savoir manipuler les automates, maîtriser les algorithmes ; une bonne connaissance des trois premières suffisait pour avoir une note correcte. Quelques erreurs ont cependant été fortement pénalisantes pour certaines copies. . . L'important est de voir si certaines notions sont mal maîtrisées, s'il vous faut plus de rigueur dans leur application, ou simplement aller un peu plus vite. Attention : la correction donnée ici est une des corrections possibles parmi d'autres, plusieurs automates / expressions pouvant reconnaître un même langage.

Exercice 1 Pour chacune des expressions rationnelles suivantes, construire un automate déterministe à états finis reconnaissant le même langage :

- 1 : a^*bab^*
- 2 : $(aab)^*b$
- 3 : $ba(aba)^*b$
- 4 : $a^*(aba)^*$

Corrigé 1 Cet exercice – facile dans tous les cas – était noté sur 5 points dans les deux sujets. La construction est directe pour les expressions rationnelles 1 et 2, un peu plus longue pour les expressions 2 et 4 ; il a généralement été bien réussi (marginale plus pour les sujets impairs). Des exemples d'automates reconnaissant les langages ainsi définis sont donnés en Fig. 1).

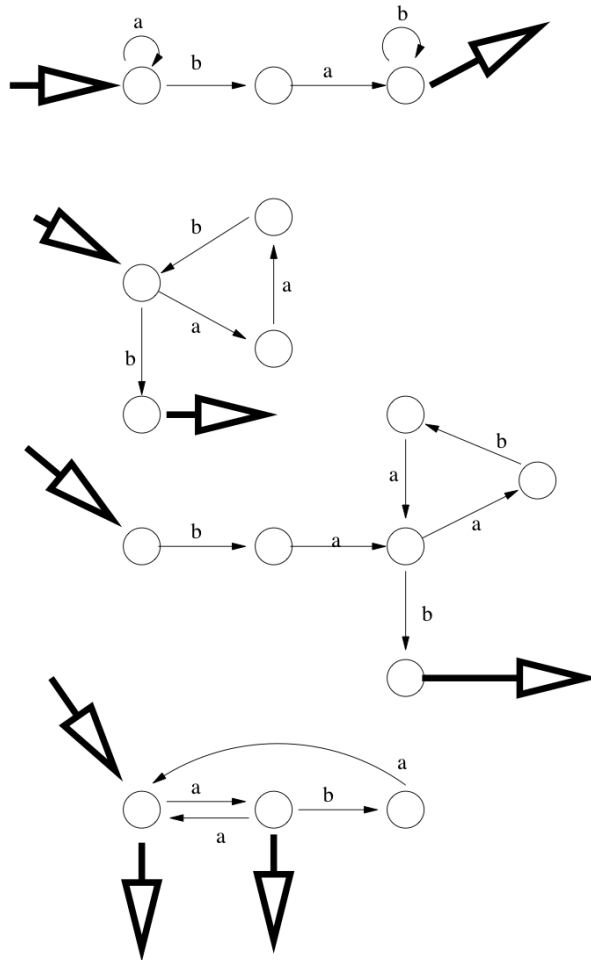


FIG. 1 – Corrigé Exercice 1

Note : suite à une inattention de ma part, la correction pour la question 4 était incorrecte ; certaines copies fausses ont été comptabilisées comme justes avec des automates ressemblant à celui en Fig. 1. Pour être plus correct, l'automate présenté en erratum, Fig. 2, correspond plus à la réalité.

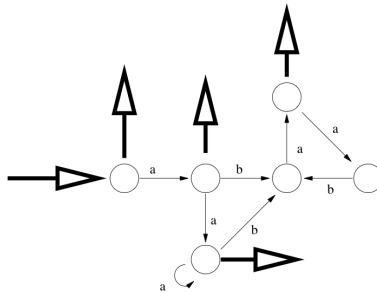


FIG. 2 – Corrigé Exercice 1 – Erratum

Exercice 2 Pour chacun des automates déterministes à états finis suivants, donner une expression rationnelle représentant le langage reconnu : (Fig. 3, 4, 5, 6).

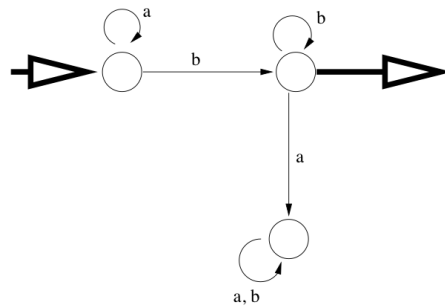


FIG. 3 – Automate 1, Exercice 2

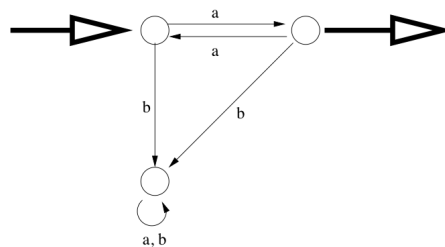


FIG. 4 – Automate 2, Exercice 2

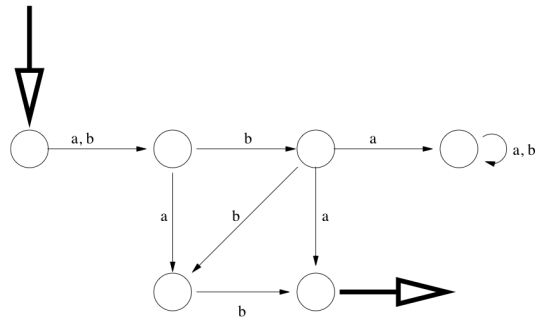


FIG. 5 – Automate 3, Exercice 2

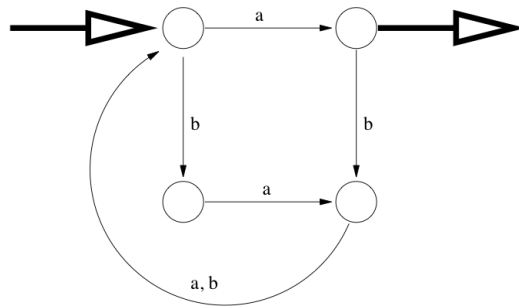


FIG. 6 – Automate 4, Exercice 2

Corrigé 2 *Cet exercice, également facile et bien réussi (beaucoup plus pour les sujets impairs), était noté sur 4 points dans les deux cas. Ici aussi, les premiers automates sont immédiats, les seconds (à l'erreur de sujet dans le numéro 3 près), un peu moins.*

1. a^*bb^*
2. $(aa)^*a$
3. $(a+b)(ab+ba+bbb)$
4. $((ab+ba)(a+b))^*a$

Exercice 3 Appliquer l'algorithme de détermination à l'automate à états finis suivant : (Fig. 7, 8).

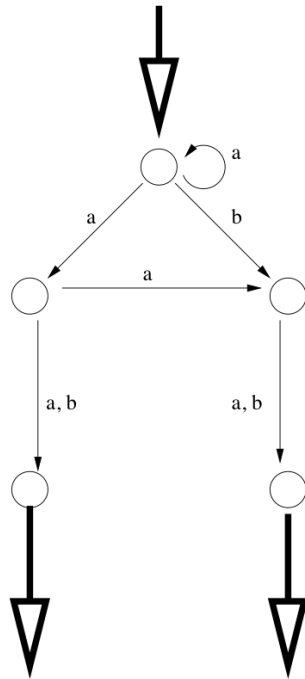


FIG. 7 – Automate 1, Exercice 3

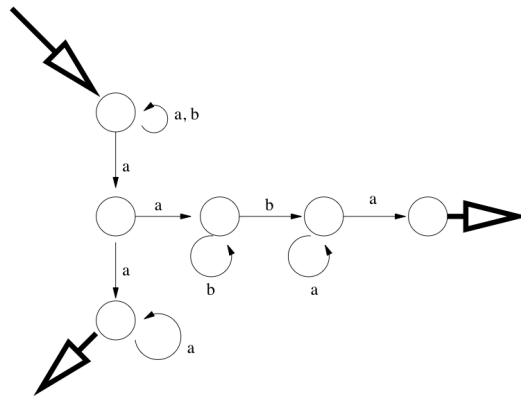


FIG. 8 – Automate 2, Exercice 3

Corrigé 3 Cet exercice, qui demandait une réelle réflexion, a été moins réussi dans le sujet impair que dans le sujet pair. Afin de maintenir une certaine équité, il a été noté sur 5 points dans les sujets pairs, sur 4 dans les sujets impairs.

Trois types d'erreurs ici : manque total de maîtrise de l'algorithme (peu fréquent), une ou plusieurs erreurs "mécaniques" dans son application (qui en entraînent généralement d'autres), erreurs dans la reconstruction de l'automate à l'arrivée (notamment oubli d'états terminaux).

	a	b
0	01	2
01	0123	23
2	4	4
Automate 1 : 0123	01234	234
23	4	4
4	-	-
01234	01234	234
234	4	4

- Fig. 9.

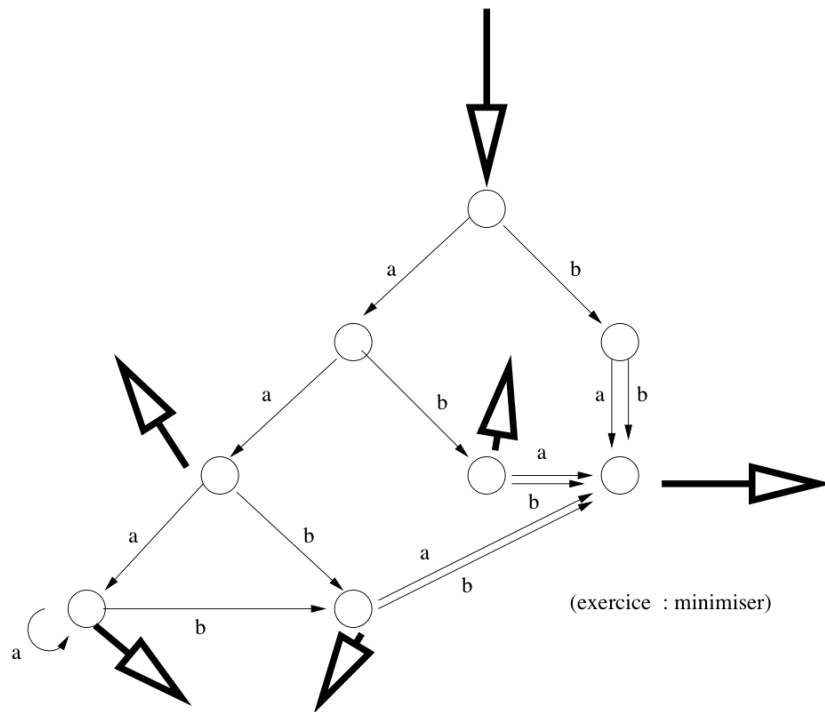


FIG. 9 – Corrigé Exercice 3, automate 1

	a	b
0	01	0
01	0123	0
Automate 2 :	0123	034
	0123	034
	0145	034
	0145	0
	012345	034
	012345	034

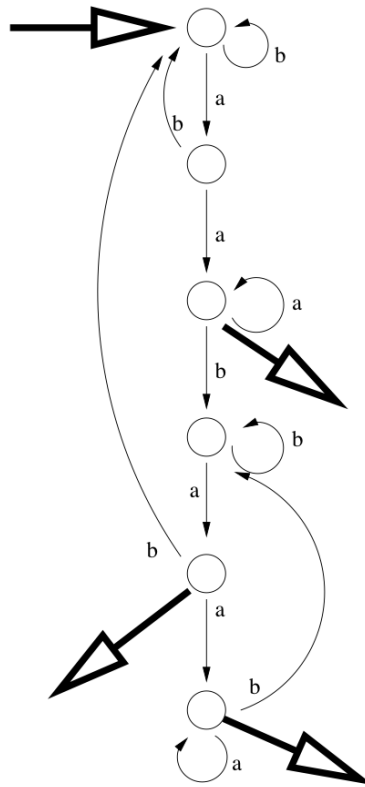


FIG. 10 – Corrigé Exercice 3, automate 2

Exercice 4 Par la méthode de votre choix, en détaillant les éventuelles étapes intermédiaires, donner l'automate à états finis, déterministe, complet et minimal reconnaissant le langage correspondant à chacune des expressions rationnelles suivantes :

- 1 : $(aa + (aba)^* + (b + a)^*)^*$
- 2 : $(ab + (baa)^* + baa + b)^*$
- 3 : $(((aba)^* + aa^*)^* + a^*b + a)^*$
- 4 : $((abb + ba)^* + (aaa + b)^* + aab)^*$

Corrigé 4 *L'exercice "difficile" du devoir. La réussite est assez faible, et un peu plus importante pour les sujets impairs que pour les sujets pairs ; pour rééquilibrer, il était noté sur 6 points pour les sujets pairs, 7 points pour les sujets impairs. Il y avait de multiples méthodes pour arriver au résultats, utilisés avec plus ou moins de bonheur : construction d'automates quelconques reconnaissant le langage, puis optimisation ; exploration systématique ; construction par résidus (longue, non vue et non utilisée) ; simplification des expressions rationnelles avant la construction (marche particulièrement bien pour le sujet impair, et à l'origine du seul sans-faute du groupe). Une certaine indulgence sur le résultat a été apportée (il y avait des points si l'automate reconnaissait une partie significative du langage, tout le langage mais plus encore, s'il était déterministe mais non minimal, ou si la construction était intéressante mais mal réalisée ; les automates ne reconnaissant pas le langage voulu, non déterministes, non complets et non justifiés ont reçu moins d'attention). Remarquer que $(e_1 + e_2^* + (e_3 + e_4)^*)^* = e_1^* + e_2^* + e_3^* + e_4^*$: l'étoile de Kleene sur l'ensemble de l'expression se distribue.*

Expression 1 : on peut remarquer que $(e + (a + b)^*)^* = (a + b)^*$, quelle que soit l'expression e . Cela permet de construire très rapidement l'automate cherché. Dans le cas contraire, construire un automate reconnaissant simultanément $(aa)^*$, $(aba)^*$, a^* et b^* (on peut en construire un déterministe sans trop de difficultés), compléter, minimiser. Illustration Fig. 11.

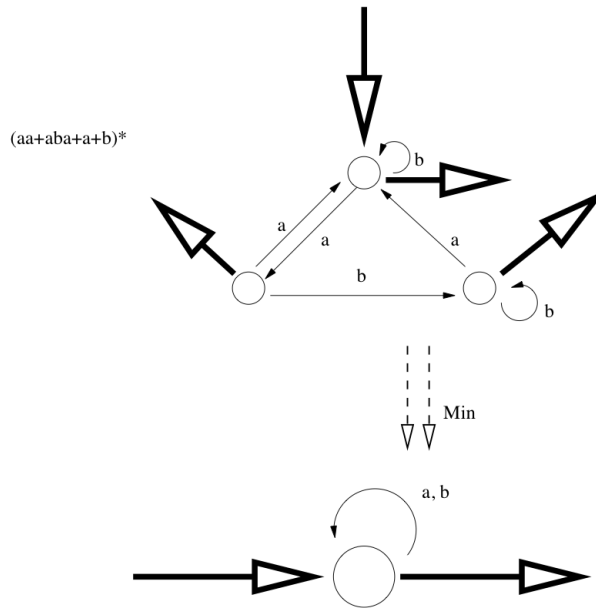


FIG. 11 – Corrigé Exercice 4, automate 1

Expression 2 : ici, pas “d’astuce” comme précédemment. On construit donc un automate reconnaissant $(ab)^*$, $(baa)^*$ et b^* (il est plus facile et direct de le faire que pour l’expression 1), compléter, minimiser. Voir Fig. 12.

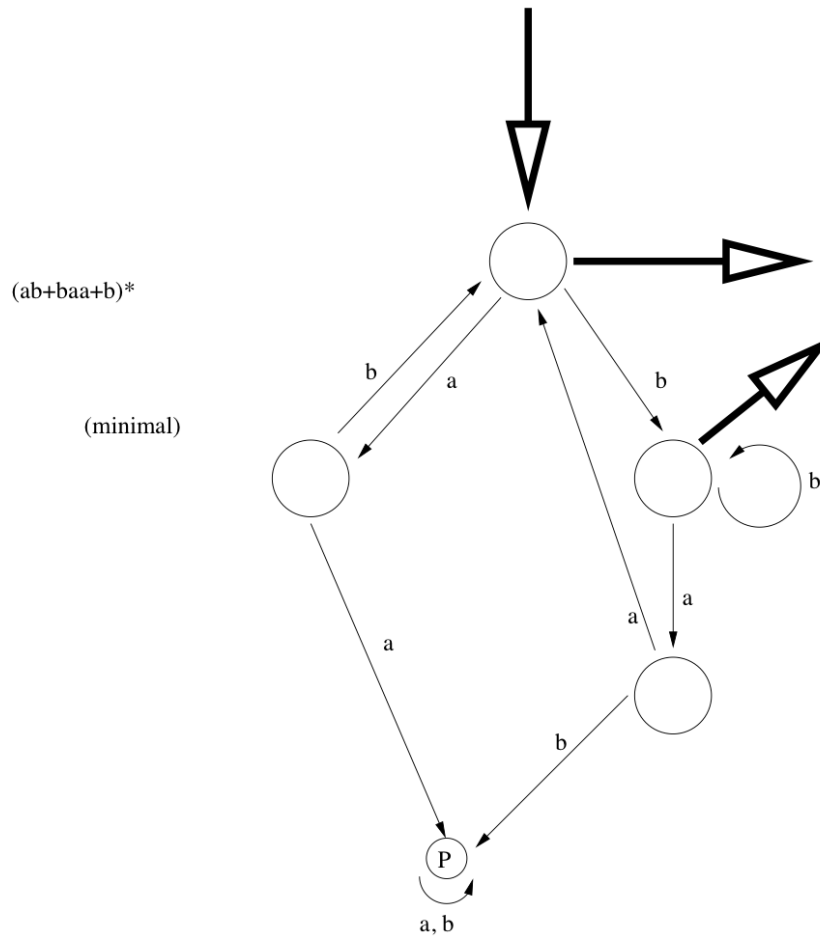


FIG. 12 – Corrigé Exercice 4, automate 2

Expression 3 : là aussi : $b \in L(a^*b)$, et donc $((aba)^* + aa^*)^* + a^*b + a)^* = (a + b)^*$, on peut procéder comme pour l'expression 1 (mais c'est moins visible à l'œil nu). Il est parfaitement envisageable de prendre le temps de construire des automates, d'abord non déterministes, reconnaissant des parties du langage, de les réunir, déterminer, compléter et minimiser le résultat, mais c'est très long. Pour s'en sortir dans ce cas là, commencer par faire un automate reconnaissant $(aba)^*$, $(aa)^*$, a^* et b^* à la fois. Abrégé en Fig. 12.

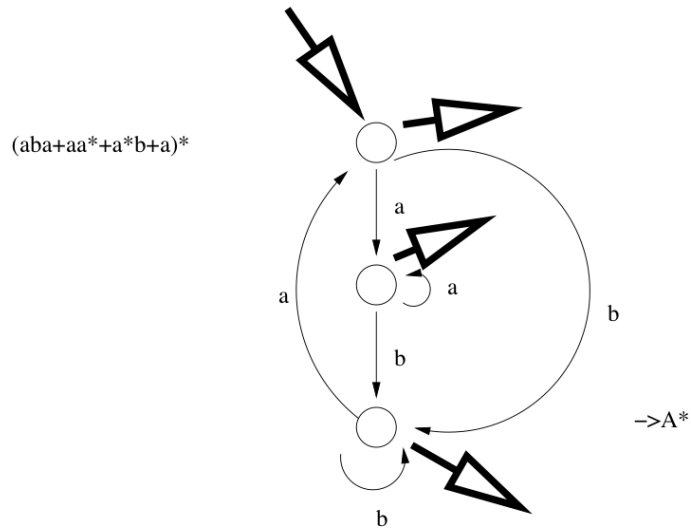


FIG. 13 – Corrigé Exercice 4, automate 3

Expression 4 : pas d’astuce non plus, l’automate reconnaissant $(abb)^*$, $(ba)^*$, $(aaa)^*$, b^* et $(aab)^*$ est assez rapide à mettre en place, puis à minimiser. Voir Fig. 14.

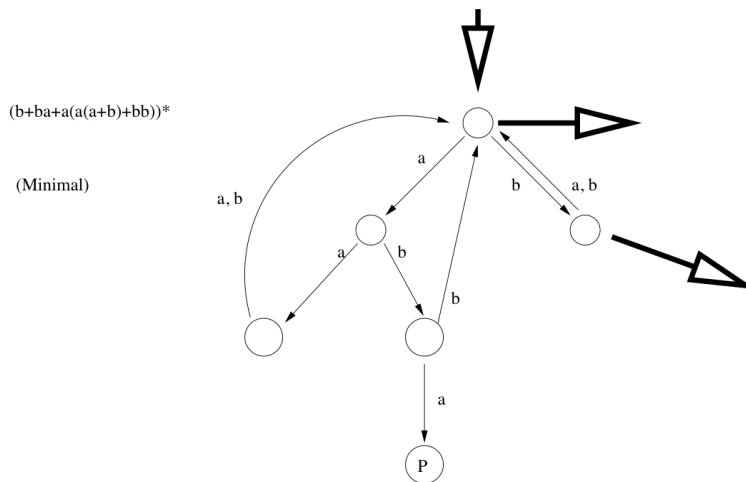


FIG. 14 – Corrigé Exercice 4, automate 4

La technique utilisée dans les schémas précédents est dite “exploration systématique” : les motifs dirigeant le langage font une, deux ou trois lettres de long. On donne donc un automate déterministe en raisonnant comme suit : si

$(aba)^*$ fait partie du langage, alors il y a trois états formant boucle sur ce motif, le premier étant initial est final. a^* également ? Dans ce cas, il y a une transition a du deuxième état, qui est également final, vers le premier, étiqueté par a. Etc... on synthétise toutes les possibilités autour d'une structure d'arbre qui garantit le déterminisme. Ensuite, on identifie les états terminaux et on complète l'automate ; il ne reste plus qu'à minimiser, étape qui doit être rapide.

Exercice 5 Soit l'alphabet $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b\}$. Donner un automate déterministe reconnaissant le langage composé des chaînes de caractères pour lesquelles la fonction pseudo-C suivante¹ renvoie 1 :

```
int stest (char *str) {
    int *i, val;
    val=sscanf(str, "%3db", i);
    return (0==val | EOF==val)?0:1;
}
```

En général, les langages décrits par les conversions de format, comme celles de `scanf` ou `printf`, sont-ils rationnels ?

Corrigé 5 Bon, les connaissances de chacun sur les modes de formatage en C n'étant manifestement pas à la hauteur de l'exercice (ni de la plupart du corps enseignant, d'ailleurs), il a été transformé en question bonus (+0.5 ou +1 point en cas de quelque chose de pertinent de près ou de loin – sans fortement être juste – décrit). Le langage reconnu par la fonction contient les mots composés d'un a , d'au moins un et au plus trois chiffres, et d'un b (automate relativement simple). Sauf dans le cas où le compilateur ignore certains types d'erreurs, selon les options, ce qui peut donner soit "exactement trois chiffres", "un certain nombre de chiffres" ou A^* dans les cas extrêmes. L'important est la dernière question : le fait que les mécanismes standards de traitement de chaînes de caractères de la plupart des langages de programmation (formats C/C++/Python/Java, regexp perl/PHP, etc.) sont des expressions rationnelles déguisées.

¹Rappels : `sscanf` prend en paramètres une chaîne de caractères et une séquence de conversion de format. Les valeurs de retours sont EOF si la chaîne est vide, le nombre de conversions réalisées sinon (0 si aucune n'est effectuée). Ne pas s'attacher aux considérations de pointeurs ou de mémoire. La dernière ligne signifie "renvoyer 1 si la chaîne correspondait au format, 0 sinon".