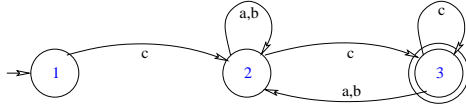


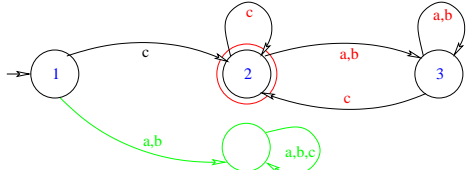
1. Soit  $\Sigma = \{a, b, c\}$ .

(a) Proposer un automate déterministe (pas nécessairement complet) qui reconnaît le langage sur  $\Sigma^*$  de tous les mots qui commencent par  $c$  et se terminent par  $c$ .

Si on considère que le  $c$  qui commence et le  $c$  qui termine ne sont pas les mêmes, on peut proposer l'automate :



Mais l'énoncé n'exclut pas que ce soit le même  $c$ , et dans ce cas, l'automate doit reconnaître aussi le mot  $c$ . Cela donne l'automate (complété ici) :

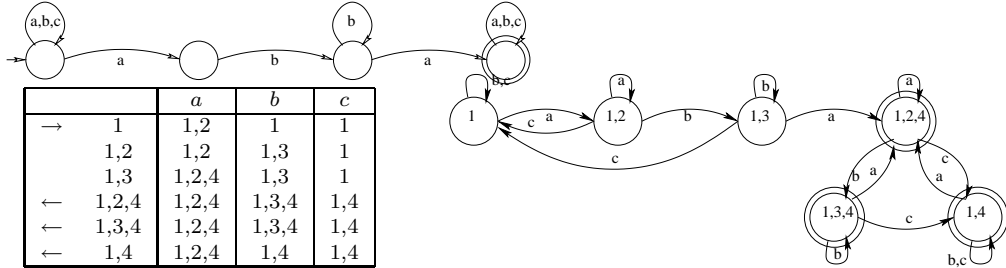


	a	b	c
→ 1	0	0	2
2	2	2	3
← 3	2	2	3
	0	0	0

On pouvait aussi passer par une version non déterministe à déterminer avec la méthode habituelle, cela donne le même résultat.

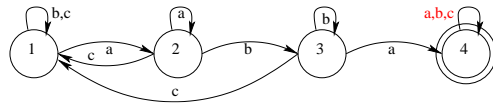
(b) Proposer un automate déterministe qui reconnaît les mots de  $\Sigma^*$  qui comprennent  $abb^*a$ .

En partant d'une version non déterministe, on obtient une version un peu compliquée :



	a	b	c
→ 1	1,2	1	1
1,2	1,2	1,3	1
1,3	1,2,4	1,3	1
← 1,2,4	1,2,4	1,3,4	1,4
← 1,3,4	1,2,4	1,3,4	1,4
← 1,4	1,2,4	1,4	1,4

Une version déterministe « à main levée » est plus simple :

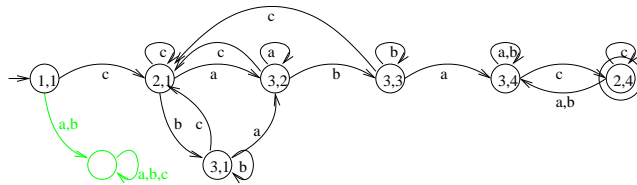


	a	b	c
→ 1	2	1	1
2	2	3	1
3	4	3	1
← 4	4	4	4

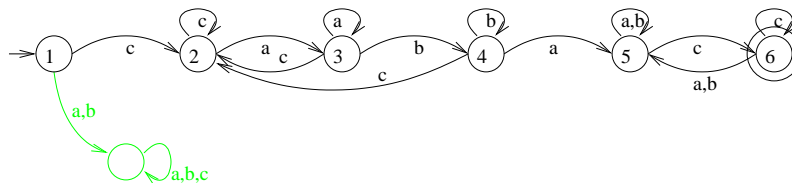
(c) Proposer un automate (pas nécessairement complet) qui reconnaît les mots de  $\Sigma^*$  qui comprennent le motif  $abb^*a$  et commencent et se terminent par  $c$ .

Il était important de remarquer que le langage concerné était l'intersection des deux langages précédents. En appliquant l'algorithme d'intersection (on simplifie en supprimant les paires d'états dont l'un est un puits) :

	a	b	c
→ 1,1			2,1
2,1	2,2	2,1	3,1
2,2	2,2	2,3	3,1
3,1	2,2	2,3	3,1
2,3	2,4	2,3	3,1
2,4	2,4	2,4	3,4
← 3,4	2,4	2,4	3,4



On pouvait aussi proposer une version « à main levée » :

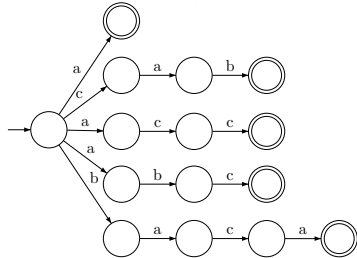


2. Soit  $L = \{a, bac, cca, abc, baca\}$

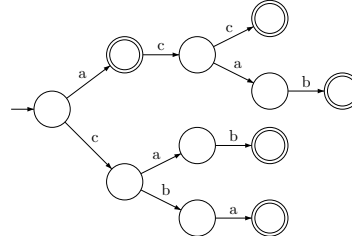
(a) Proposer un automate qui reconnaît  $\bar{L}$ , le langage qui contient les mots « renversés » de  $L$  ( $\{a, cab, acc, cba, acab\}$ ).

On peut proposer directement un automate :

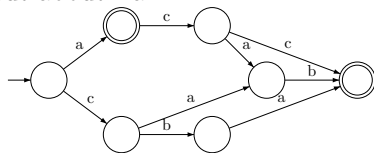
Version « brutale », non déterministe :



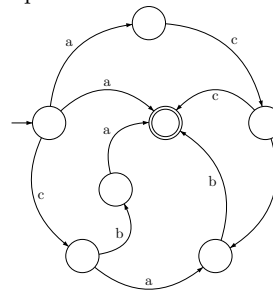
Version plus maline, où on détermine :



... et si on essaie d'économiser les états, on peut aboutir à :

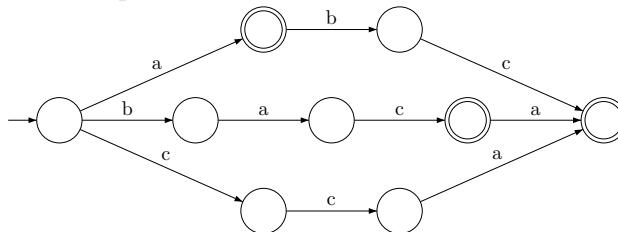


On pouvait aussi n'obtenir qu'un état initial, mais au prix d'un retour du non-déterminisme :

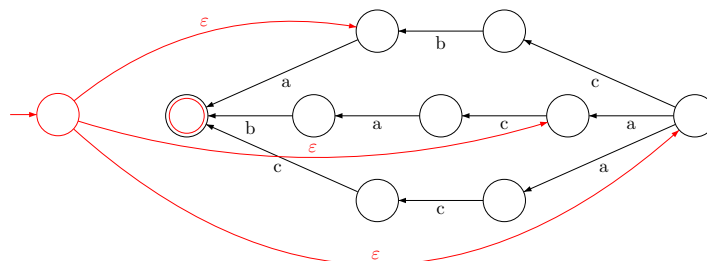


Mais on pouvait aussi, comme le suggérait le reste de l'exercice, partir de l'automate reconnaissant  $L$  ( $\{a, bac, cca, abc, baca\}$ ).

Par exemple, soit  $\mathcal{A}$ , qui reconnaît  $L$  :



À partir de  $\mathcal{A}$ , on peut construire  $\mathcal{A}'$  :



Cet automate, non déterministe (pas seulement à cause des  $\epsilon$ ), reconnaît  $\bar{L}$ .

(b) Comment démontrer, en général, que si  $L$  est rationnel, alors  $\bar{L}$  l'est aussi ?

Si  $L$  est rationnel, il existe un automate  $A$  qui le reconnaît.

Alors on peut construire un automate  $\bar{A}$  qui vérifie les propriétés suivantes :

- $\bar{A}$  a les mêmes états que  $A$ , plus un nouvel état initial.
- l'état initial de  $\bar{A}$  est relié par une  $\varepsilon$ -transition à tous les états finaux dans  $A$ .
- toutes les transitions de  $A$  sont inversées dans  $\bar{A}$ .
- l'état initial de  $A$  devient le seul état final de  $\bar{A}$ .

Formellement, pour tout automate  $\mathcal{A} = \langle Q, X, \delta, q_0, F \rangle$ , on peut construire  $\bar{\mathcal{A}} = \langle Q \cup \{I\}, X, \bar{\delta}, I, \{q_0\} \rangle$ , tel que :

- $\forall q \in F, (I, \varepsilon, q) \in \bar{\delta}$
- $\forall (q, x, q') \in \delta, (q', x, q) \in \bar{\delta}$

Puisque l'on peut toujours construire un tel automate, et puisqu'il reconnaît le langage inversé de l'automate initial, on peut conclure que l'inversé d'un langage  $L$  rationnel est rationnel.

(c) Même question avec le langage  $MAX(L) = \{u \in L \mid \forall v \neq \varepsilon, uv \notin L\}$ .

Pour un langage  $L$  donné,  $MAX(L)$  est le langage qui comprend les mots de  $L$  qui ne sont pas les préfixes d'autres mots de  $L$ .

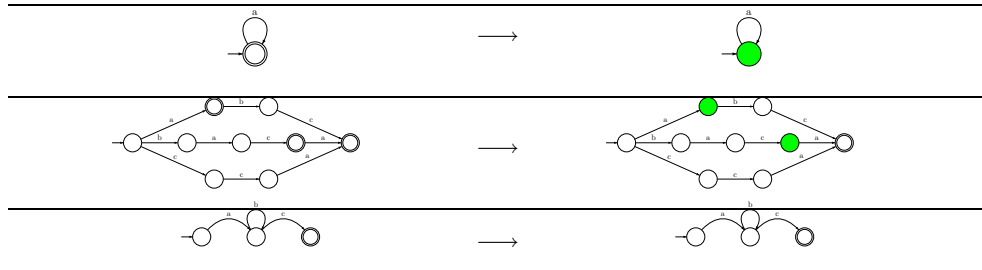
Quelques exemples pour fixer les idées :

- Si  $L = a^* = \{\varepsilon, a, aa, aaa, \dots\}$  alors  $MAX(L) = \emptyset$
- Si  $L = \{a, bac, cca, abc, baca\}$  alors  $MAX(L) = \{cca, abc, baca\}$
- Si  $L = ab^*c = \{ac, abc, abbc, abbbc, \dots\}$  alors  $MAX(L) = L$

Pour démontrer que si  $L$  est rationnel, alors  $MAX(L)$  l'est aussi, il suffit de montrer que l'on peut transformer tout automate qui reconnaît  $L$  en un automate qui reconnaît  $MAX(L)$ .

La transformation pertinente consiste à ne plus considérer comme terminaux les états qui ont des transitions sortantes utiles (*i.e.*, n'allant pas dans un puits).

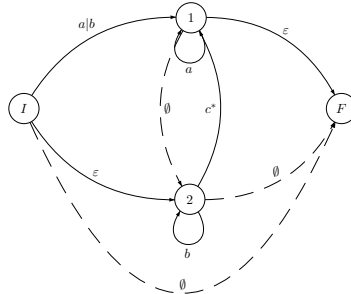
Faute d'une démonstration générale, on peut facilement vérifier que cette transformation donne bien les résultats attendus pour les trois exemples précédents :



On peut donc conclure que pour tout langage  $L$ , s'il existe un automate qui le reconnaît, il existe un automate qui reconnaît  $MAX(L)$ . Donc, si  $L$  est rationnel, alors  $MAX(L)$  l'est aussi.

3. Soit l'automate généralisé représenté dans le tableau suivant. Donnez l'expression rationnelle que l'on obtient en supprimant en premier l'état 1, puis l'état 2; et donnez celle que l'on obtient en supprimant en premier l'état 2. Qu'en déduisez-vous ?

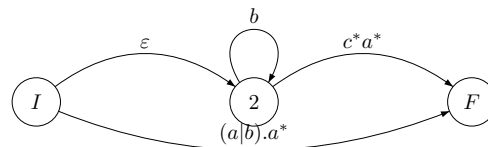
Automate généralisé initial :



↗	1	2	F
I	a b	ε	∅
1	a	∅	ε
2	c*	b	∅

Suppression de l'état 1 (puis de l'état 2)

↗	2	F
I	<del>a b.a*</del>   ∅	a b.a*.ε
	ε	∅
2	<del>c*.a*</del>   ∅	c*.a*.ε
	b	∅

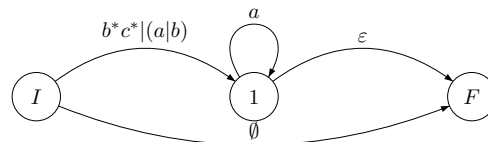


d'où, après suppression de l'état 2, l'expression

$$\boxed{\varepsilon b^* c^* a^* | (a|b).a^*}$$

Suppression de l'état 2 (puis de l'état 1)

↗	1	F
I	ε.b*.c*	<del>ε b*</del>   ∅
	(a b)	∅
1	<del>∅ b*</del>   ∅	<del>ε b*</del>   ∅
	a	ε



d'où, après suppression de l'état 1, l'expression

$$\boxed{(b^*c|(a|b))a^*}$$

- Par définition, les expressions obtenues, bien que différentes, sont équivalentes (décrivent le même langage). On pourrait imaginer utiliser cet algorithme pour démontrer des équivalences remarquables.
  - Cet algorithme garantit une expression rationnelle, mais pas une expression rationnelle canonique ou minimale.
4. On s'autorise à écrire une grammaire avec des expressions rationnelles en partie droite. Par exemple,  $S \rightarrow (AB | AA)$   
 $A \rightarrow aa^*$   
 $B \rightarrow b^*$

(a) Quel est le langage reconnu par cette grammaire ?

Le langage reconnu est décrit par l'expression  $(aa^*b | aa^*aa^*)$ . On peut simplifier cette expression de différentes manières :  $aa^*(b^*|aa^*)$ , ou  $a^+(b^*|a^+)$ ...

- (b) *Peut-on proposer une grammaire algébrique « normale » qui reconnaisse ce langage ?*

La grammaire la plus simple :

$$\begin{aligned} A &\rightarrow AB \mid AA \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid \varepsilon \end{aligned}$$

- (c) *Est-ce qu'en général, on augmente le pouvoir expressif des grammaires algébriques en autorisant des expressions rationnelles en partie droite ? Justifiez brièvement votre réponse.*

Non, le fait d'autoriser des expressions rationnelles en partie droite n'ajoute pas de pouvoir expressif.

Idée de la démonstration : pour toute expression rationnelle, on peut définir une grammaire régulière qui la reconnaît. Donc on peut remplacer une règle de la forme  $A \rightarrow \alpha$  où  $\alpha$  est une expression rationnelle par la règle  $A \rightarrow A'$ , où  $A'$  est l'axiome d'une grammaire régulière reconnaissant le même langage que  $\alpha$ .

5. Soit la grammaire suivante :

$$\begin{aligned} S &\rightarrow aSB \mid BA \\ A &\rightarrow Ac \mid \varepsilon \\ B &\rightarrow bAa \end{aligned}$$

- (a) *Proposer une grammaire équivalente sans  $\varepsilon$ -productions.* Puisque  $A$  peut « s'effacer », pour enlever la règle  $A \rightarrow \varepsilon$ , il faut envisager dans toutes les règles le cas où  $A$  s'effacerait :

$$\begin{aligned} S &\rightarrow aSB \mid BA \mid B \\ A &\rightarrow Ac \mid c \\ B &\rightarrow bAa \mid ba \end{aligned}$$

- (b) *Proposer une grammaire équivalente sans productions singulières.*

Il y a une seule production singulière directe ( $S \rightarrow B$ ), et pas de productions singulières indirectes. Pour l'éliminer, on « saute » une étape de dérivation, et on remplace  $B$  par tout ce que donnerait  $B$  dans la règle  $S \rightarrow B$ .

$$\begin{aligned} S &\rightarrow aSB \mid BA \mid ba \mid bAa \\ A &\rightarrow Ac \mid c \\ B &\rightarrow bAa \mid ba \end{aligned}$$

- (c) *Proposer une grammaire équivalente sans récursivité gauche.*

Une seule règle récursive gauche directe, la règle  $A \rightarrow Ac$ . Pas de récursivité indirecte.

En appliquant l'algorithme à la lettre, on obtient à la place de  $A \rightarrow Ac$  :

$$\begin{aligned} A &\rightarrow cA' \\ A' &\rightarrow cA' \mid \varepsilon \end{aligned}$$

Ce qui donne la grammaire (sans  $\varepsilon$ ) :

$$\begin{aligned} S &\rightarrow aSB \mid BA \mid ba \mid bAa \\ A &\rightarrow cA' \mid c \\ A' &\rightarrow cA' \mid c \\ B &\rightarrow bAa \mid ba \end{aligned}$$

En observant que le non terminal  $A$ , source de la règle à dérécursiver, engendre exactement le langage  $c^+$ , on peut proposer directement une grammaire non récursive :

$$\begin{aligned} S &\rightarrow aSB \mid BA \mid ba \mid bAa \\ A &\rightarrow cA \mid c \\ B &\rightarrow bAa \mid ba \end{aligned}$$

- (d) *Proposer une grammaire équivalente en forme normale de Greibach.*

Ici, on peut appliquer l'algorithme, et ordonner les non terminaux. La seule contrainte est d'avoir  $S < B$ . On ne trouve comme règle à transformer que la règle  $S \rightarrow BA$ . On va la remplacer par  $S \rightarrow bAaA \mid baA$ . Cela donne, selon la grammaire d'origine :

$$\begin{aligned} S &\rightarrow aSB \mid bAaA \mid baA \mid ba \mid bAa \\ A &\rightarrow cA' \mid c \\ A' &\rightarrow cA' \mid c \\ B &\rightarrow bAa \mid ba \end{aligned} \quad \left| \quad \begin{aligned} S &\rightarrow aSB \mid bAaA \mid baA \mid ba \mid bAa \\ A &\rightarrow cA \mid c \\ B &\rightarrow bAa \mid ba \end{aligned}$$

Il reste à remplacer les terminaux en partie droite, ce qui donne, selon la grammaire d'origine :

$$\begin{aligned} S &\rightarrow aSB \mid bAXA \mid bXA \mid bX \mid bAX \\ A &\rightarrow cA' \mid c \\ A' &\rightarrow cA' \mid c \\ B &\rightarrow bAX \mid bX \\ X &\rightarrow a \end{aligned} \quad \left| \quad \begin{aligned} S &\rightarrow aSB \mid bAXA \mid bXA \mid bX \mid bAX \\ A &\rightarrow cA \mid c \\ B &\rightarrow bAX \mid bX \\ X &\rightarrow a \end{aligned}$$