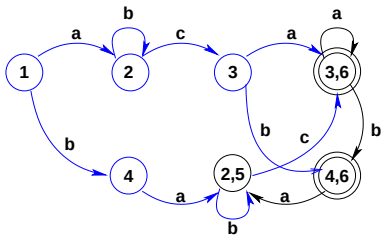
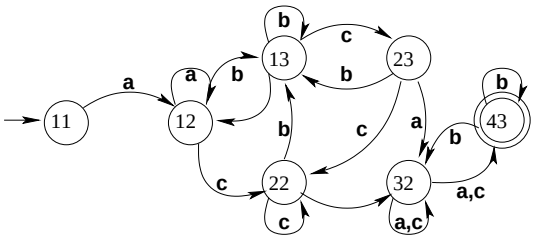
1. En appliquant l'algorithme de déterminisation, qui consiste, en partant de l'état initial, à créer au fur et à mesure du parcours des états correspondant à l'union des chemins possibles, on aboutit à l'automate représenté graphiquement par la figure suivante :



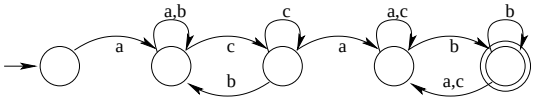
	a	b	c
→ 1	2	4	/
2	/	2	3
4	(2,5)	/	/
3	(3,6)	(4,6)	/
(2,5)	/	(2,5)	(3,6)
← (3,6)	(3,6)	(4,6)	/
← (4,6)	(2,5)	/	/

2. On cherche à reconnaître les mots appartenant aux deux langages (et seulement ceux-là), il s'agit donc d'une **intersection**.
   (a) Une première approche consiste à tenter de retrouver, à partir des automates, une caractérisation des deux langages en jeu, pour ensuite caractériser le langage correspondant à leur intersection. Ici, on voit que le *premier* automate reconnaît *tous les mots qui comprennent le motif ca, et qui se terminent par b*, le *second* automate reconnaît *tous les mots qui commencent par a et se terminent par b*. (Les deux automates sont déterministes, ce qui explique leur apparente complexité.)
   Alors l'intersection est caractérisée comme l'ensemble de *tous les mots qui commencent par a, comprennent le motif ca, et se terminent par b*. Dès lors, il suffit de construire l'automate correspondant à cette description, ce qui est demandé à la question suivante.
   (b) L'autre approche consiste à appliquer un algorithme de création de l'automate intersection : on fait un parcours simultané des deux automates, en créant au fur et à mesure du parcours des états formé d'un couple (q_a, q_b) où q_a est un état du premier automate, et q_b un état du second. Voici l'automate résultant :

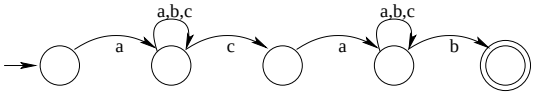
	a	b	c
→ (1,1)	(1,2)	/	/
(1,2)	(1,2)	(1,3)	(2,2)
(1,3)	(1,2)	(1,3)	(2,3)
(2,2)	(3,2)	(1,3)	(2,2)
(2,3)	(3,2)	(1,3)	(2,2)
(3,2)	(3,2)	(4,3)	(3,2)
← (4,3)	(3,2)	(4,3)	(3,2)



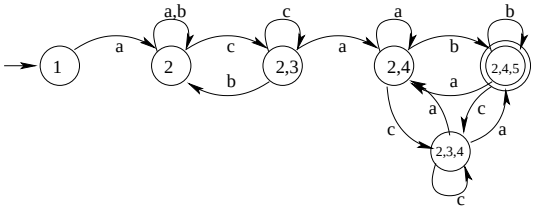
3. (a) Si on a répondu à la question précédente, il est inutile d'en dire plus ici.
   (b) Sinon, on peut soit construire "à la main" l'automate (déterministe) demandé, cela donne l'automate suivant :



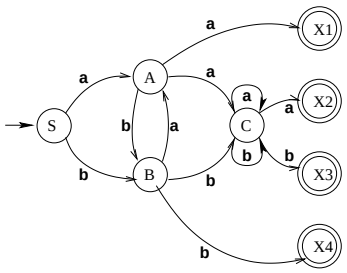
   (c) Ou bien on peut passer par une version non déterministe, plus simple à imaginer :

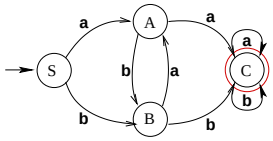


   qu'il faut ensuite déterminiser :



4. La méthode de correspondance donnée en cours donne directement l'automate, dont les états correspondent aux non-terminaux de la grammaire. La seule (petite) difficulté quand on applique strictement l'algorithme vient du fait que les productions en $N \to x$ créent un nouvel état terminal. Une application stricte de l'algorithme donnerait donc l'automate représenté à gauche. Cependant, il est facile de montrer que cet automate reconnaît le même langage que la version de droite, qui comprend 4 états. On peut remarquer pour retrouver ce résultat autrement que la grammaire initiale est équivalente à la grammaire avec ε-production suivante, à partir de laquelle l'algorithme de correspondance donnerait exactement l'automate de droite.





$$\begin{array}{lcl} S & \to & aA \mid bB \\ A & \to & bB \mid aC \\ B & \to & aA \mid bC \\ C & \to & aC \mid bC \mid \varepsilon \end{array}$$

5. Deux méthodes possibles pour cet exercice :
   (a) Construction d'une expression par raisonnement sur l'automate. Une expression possible est $a(b^*|ab^*cc)^*ab^*$. Le raisonnement est le suivant :
   (1) tous les mots commencent par a (seule transition possible) : aX ; (2) tous les mots se terminent par a suivi d'un nombre quelconque de b (c'est le seul moyen d'arriver à l'état 4) : $aXab^*$; il faut nécessairement passer par l'état 2, mais on remarque qu'il est possible de passer plusieurs fois par cet état. Quelles sont les possibilités ? (i) soit la boucle de b (donc b^*), (ii) soit la boucle (4, 3, 2) (donc ab^*cc, le b^* vient ici de la boucle sur 4). On a donc $(b^*|ab^*cc)$, et cela peut se répéter, l'étoile s'applique donc à cette expression, ce qui donne bien $a(b^*|ab^*cc)^*ab^*$.
   D'autres raisonnements, tout aussi valides, ont conduit à d'autres propositions trouvées dans vos copies : $a((b^*|a)b^*cc)^*ab^*$
   $ab^*(a|ab^*(ccb^*ab^*)^*)$
   (b) On peut aussi appliquer l'algorithme vu en cours (automate généralisé puis suppression de tous les états 1 à 1), ce qui donne, par exemple en suivant l'ordre illusté ci-après, l'expression $ab^*a(ccb^*a|b)^*$.