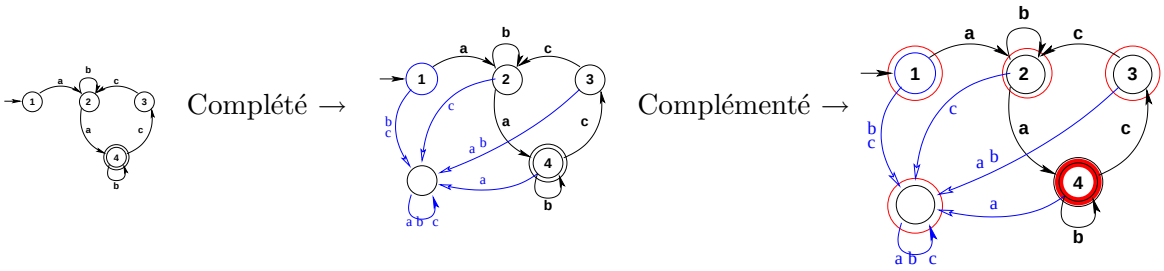
1.3.3.3 Complémentation

Le complément d'un langage $\mathcal{L}_1$ est l'ensemble de tous les mots du monoïde qui n'appartiennent pas à ce langage. En terme d'automate, il s'agit donc de tous les mots qui n'ont pas de chemin aboutissant à un état final dans l'automate reconnaissant $\mathcal{L}_1$.

L'algorithme pour construire le complément d'un automate est relativement intuitif : il suffit que tous les états d'échec de l'automate initial deviennent des états de réussite, et réciproquement. Pratiquement, il suffit de rendre terminaux les états non terminaux et réciproquement (on échange Q et $Q \setminus F$). Mais attention, il est nécessaire que tous les chemins possibles soient présent dans l'automate, et donc qu'il soit **complet** ; de même il est nécessaire que l'automate initial soit **déterministe**.

exemple :



En exercice, le lecteur est invité à se figurer ce qui se produit lorsque ces contraintes ne sont pas vérifiées, et que l'on applique l'algorithme (échange de Q et $Q \setminus F$).

1.3.3.4 Intersection

Théorie : on sait que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. On pourrait donc utiliser les algorithmes précédents. Mais il y a une autre méthode, moins fastidieuse (ne pas oublier que la complémentation nécessite d'abord une déterminisation).

Intuitivement, l'idée est de parcourir "en parallèle" les deux automates, et de ne garder que les chemins qui existent dans les deux automates. Pour cela, les états du nouvel automate sont des couples (q_i, q_j), où q_i appartient au premier automate et q_j au second. Pour chaque lettre de transition, on crée le nouvel état-couple atteint, et on continue.

	a	b
→ 1	2	4
2	4	3
← 3	3	3
4	4	4

	a	b
↔ 1	2	5
2	5	3
3	4	5
4	1	4
5	5	5

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

Les mêmes contraintes que précédemment s'appliquent : on part de deux automates **déterministes complets**. À noter aussi qu'un tel algorithme, comme l'algorithme de déterminisation, a le mérite de ne pas conserver les états non atteint depuis l'état initial.