Bases formelles du TAL
corrections d'exercices

Pierre-Léo Bégay

February 28, 2020

Contents

Chapter 1

Langages

1.1 Mots

Exercice 1.1.1. *Combien de préfixes et suffixes admet un mot w quelconque ?*

Correction. *Soit un mot w qui se décompose en $c_1c_2...c_n$ avec $\forall i \in [1-n], c_i \in \Sigma$. Il y a $n+1$ préfixes : $\epsilon, c_1, c_1c_2, c_1c_2c_3, ...,$ et $c_1...c_n$ entier. Pareil pour les suffixes avec $\epsilon, c_n, c_{n-1}c_n, ..., c_1...c_n$.*

Pour tout mot w, il y a donc $|w|+1$ préfixes et suffixes.

Exercice 1.1.2. *Donner l'ensemble des facteurs du mot abbba.*

Correction. *On note en rouge les facteurs :*

- abbba (ϵ)
- abbba
- abbba
- abbba
- abbba
- abbba
- abbba
- abbba
- abbba
- abbba
  - *On saute abbba et abbba*
- abbba
  - *On saute abbba*
- abbba
  - *On saute abbba*

Exercice 1.1.3. *(*) Donner la borne la plus basse possible du nombre de facteurs d'un mot w. Donner un mot d'au moins 3 lettres dont le nombre de facteurs est exactement la borne donnée.*

Correction. *Soit $w = c_1...c_n$. L'ensemble des facteurs de w est l'ensemble des $c_1...c_{i-1}c_i...c_jc_{j+1}...c_n$ ainsi qu'ϵ. Le nombre de ces facteurs non-nuls est borné par*

$$|\{(i,j) \mid 0 \leq i < j \leq n\}| =^{1} 1 + 2 + 3 + ... + n =^{2} \frac{n(n+1)}{2} \in O(n^2)$$

Il ne s'agit que d'une borne, puisqu'il y aura des répétitions à partir du moment où une même lettre apparaît deux fois (cf. l'exercice précédent). Dualement, le mot abc par exemple contient bien $1 + \frac{3\times4}{2} = 7$ facteurs.

Exercice 1.1.4. *Montrer que tout facteur d'un mot en est également un sous-mot. A l'inverse, montrer qu'un sous-mot n'est pas forcément un facteur.*

Correction. *Soit f facteur d'un mot w. D'après la définition, ça veut dire qu'il existe v_1 et v_2 tels que $w = v_1.f.v_2$. On a alors bien $w = v_0.s_0.v_1$ avec $s_0 = f$.*

Soit $w = abc$. ac en est clairement un sous-mot, alors qu'il n'en est pas un facteur.

Exercice 1.1.5. *Donner toutes les façons de voir $abba$ comme sous-mot de $baaabaabbaa$.*

Correction. *On a la liste (beaucoup trop longue (22 éléments !)) suivante :*

- $b\underline{a}aa\underline{b}aa\underline{b}b\underline{a}a$
- $b\underline{a}aa\underline{b}aa\underline{b}ba\underline{a}$
- $b\underline{a}aa\underline{b}aab\underline{b}\underline{a}a$
- $b\underline{a}aa\underline{b}aab\underline{b}a\underline{a}$
- $b\underline{a}aabaa\underline{b}\underline{b}\underline{a}a$
- $b\underline{a}aabaa\underline{b}\underline{b}a\underline{a}$
- $ba\underline{a}a\underline{b}aa\underline{b}b\underline{a}a$
- $ba\underline{a}a\underline{b}aa\underline{b}ba\underline{a}$
- $ba\underline{a}a\underline{b}aab\underline{b}\underline{a}a$
- $ba\underline{a}a\underline{b}aab\underline{b}a\underline{a}$
- $ba\underline{a}abaa\underline{b}\underline{b}\underline{a}a$
- $ba\underline{a}abaa\underline{b}\underline{b}a\underline{a}$
- $baa\underline{a}\underline{b}aa\underline{b}b\underline{a}a$
- $baa\underline{a}\underline{b}aa\underline{b}ba\underline{a}$
- $baa\underline{a}\underline{b}aab\underline{b}\underline{a}a$
- $baa\underline{a}\underline{b}aab\underline{b}a\underline{a}$
- $baa\underline{a}baa\underline{b}\underline{b}\underline{a}a$

[1] Quand i vaut 0, il y a n possibilités pour j. Quand i vaut i, il y a $n-1$ possiblités pour j. ... et quand i vaut $n-1$, il y a une possibilité pour j.

[2] Prouvable assez facilement par induction sur n (bon entraînement si vous n'avez pas l'habitude)

- $baa\underline{a}baa\underline{bb}a\underline{a}$
- $baaab\underline{a}a\underline{bba}a$
- $baaab\underline{a}a\underline{bb}a\underline{a}$
- $baaaba\underline{abba}a$
- $baaaba\underline{abb}a\underline{a}$

Exercice 1.1.6. *Donner l'ensemble des sous-mots de abba*

Correction. *On a la liste suivante de sous-mots :*

- abba (ϵ)
- abba
- abba
- abba
  - *On saute abba et abba*
- abba
- abba
  - *On saute abba*
- abba
- abba
- abba
  - *On saute abba et abba*
- abba
- abba

Exercice 1.1.7. *(*)*

Donner la borne la plus basse possible du nombre de sous-mots d'un mot w. Donner un mot dont le nombre de sous-mots est exactement la borne donnée.

Correction. *Pour construire l'ensemble des sous-mots d'un mot w, on choisit de garder ou non chaque lettre du mot. On a donc $2^{|w|}$ choix. On a d'ailleurs, pour énumérer l'ensemble des sous-mots de l'exercice précédent, "généré" la suite des séries de 5 bits (00000, 00001, 00010, 00011, etc) qu'on a collées sur abbba.*

Il peut y avoir des répétitions, comme dans l'exercice précédent, ce $2^{|w|}$ n'est donc qu'une borne maximale, cependant atteinte par un mot comme abc.

Exercice 1.1.8. *(*) Dans l'exercice 1.1.1, on demande le nombre exact de préfixes et suffixes d'un mot, alors que dans les exercices 1.1.3 et 1.1.7, on demande une borne, pourquoi ?*

Correction. *Le nombre de préfixes et de suffixes est toujours le même, puisqu'on n'y trouve pas de problèmes de répétitions, contrairement aux facteurs et sous-mots (cf. les exercices associés).*

1.2 Langage

Chapter 2

Expressions régulières

2.1 Lexique et idée générale

2.1.1 Les lettres et ϵ, la base

2.1.2 ., la concaténation

2.1.3 *, l'itération

Exercice 2.1.1. *Donner 5 autres mots appartenant au langage dénotée par l'expression* $ab(bab)^*b(ca)^*b$.

Correction. *abbabbabbb (première étoile instanciée à* 2 *et seconde à* 0*), abbabbabbcab (*2 *et* 1*), abbabbabbcacacab (*2 *et* 3*), abbabbabbabbcacacab (*3 *et* 3*) et abbabbabbabbabbabbcacacacacab (*5 *et* 5*).*

Exercice 2.1.2. *Pourquoi le changement de formulation dans les exemples 2.1.5 et 2.1.6 par rapport aux exemples précédents ("c'est à dire* $\{x, y, z...\}$*" qui devient "contenant notamment* x*,* y *ou* z*") ?*

Correction. *Parce qu'on se met à étudier des* regex *qui dénotent des langages infinis, et dont il est donc assez peu pratique de faire une liste exhaustive des mots.*

2.1.4 +, la disjonction

Exercice 2.1.3. *Donner un mot acceptant deux dérivations avec la regex* $(aa)^* + (bb)^*$ *(justifier en donnant les dérivations). Existe-t-il un autre mot admettant plusieurs dérivations ?*

Correction. *On a les deux dérivations suivantes du mot* ϵ *:*

$$\underbrace{\underbrace{\underbrace{\epsilon}_{(bb)^0}}_{(bb)^*}}_{(aa)^* + (bb)^*} \qquad\qquad \underbrace{\underbrace{\underbrace{\epsilon}_{(aa)^0}}_{(aa)^*}}_{(aa)^* + (bb)^*}$$

Il n'existe pas d'autre mot acceptant plusieurs dérivations : dans une dérivation, on choisit d'abord si le mot sera composé de a ou de b, puis on choisit sa longueur (paire). Deux dérivations différentes généreront donc deux mots qui diffèrent par leur longueur ou les lettres qui le composent, et donc deux mots qui seront forcément différents à moins que la longueur soit 0.

Exercice 2.1.4. *Existe-t-il un mot acceptant plusieurs dérivations pour la regex $(aa + bb)^*$?*

Correction. *Non, dans cette* regex *on choisit d'abord la longueur (paire) du mot, puis les lettres qui le composent. On n'a donc plus le cas de l'exercice précédent.*

Exercice 2.1.5. *Donner un mot accepté par la regex $(aa + bb)^*$ mais pas $(aa)^* + (bb)^*$. Est-il possible de trouver un mot qui, à l'inverse, est accepté par la deuxième mais pas la première ?*

Correction. *La première* regex *accepte par exemple $bbaa$, qui ne fait pas partie du langage dénoté par la seconde.*

Tout mot accepté par la seconde regex *sera accepté par la première : si on a une dérivation de la forme $(aa)^* + (bb)^* \to (aa)^* \to (aa)^n$, alors on a également $(aa + bb)^* \to (aa + bb)^n \to^n (aa)^n$ (le $\to^n$ indique qu'il y a n étapes de dérivation, en l'occurrence n fois le choix de aa dans $aa + bb$). Même raisonnement si on part sur les b.*

Exercice 2.1.6. *(*) Exprimer, en langue naturelle et de façon concise, le langage dénoté par la regex $(a^*b^*)^*$. Traduire ensuite ce langage en une regex non-ambiguë, c'est-à-dire où il n'y aura qu'une dérivation pour chaque mot.*

Correction. *La* regex *permet d'engendrer n'importe quel mot. En effet, soit le mot $c_1c_2...c_n$, où $c_i \in \{a, b\}$ pour tout i, on peut par exemple commencer la dérivation par $(a^*b^*)^* \to (a^*b^*)^n$ et, quand $c_i = a$ (resp. b), instancier le i^{eme} facteur par a^1b^0 (resp. a^0b^1).*

Le langage accepté, Σ^*, est plus simplement reconnu par la regex *$(a + b)^*$.*

2.2 Syntaxe

2.3 Sémantique

2.3.1 Les cas de base

2.3.2 Sémantique de la concaténation

2.3.3 Sémantique de la disjonction

2.3.4 Sémantique de l'itération

2.4 Mise en application

2.4.1 Quelques astuces

Exercice 2.4.1. *Donner une regex pour les mots qui commencent par a.*

Correction. $a\Sigma^*$

Exercice 2.4.2. *Donner une regex pour les mots qui finissent par b.*

Correction. Σ^*b

Exercice 2.4.3. *Donner une regex pour les mots qui commencent par a finissent par b.*

Correction. $a\Sigma^*b$

Exercice 2.4.4. *Donner une regex pour les mots de longueur paire.*

Correction. $(\Sigma\Sigma)^*$

Exercice 2.4.5. *Donner une regex pour les mots de longueur impaire qui contiennent au moins 4 lettres.*

Correction. $\Sigma^4(\Sigma\Sigma)^*\Sigma$

Exercice 2.4.6. *Donner une regex pour les mots de longueur impaire, qui contiennent au moins 4 lettres, commencent par a et finissent par b.*

Correction. $a\Sigma^3(\Sigma\Sigma)^*b$

2.4.2 Syntaxe en pratique

2.4.3 Calcul de l'appartenance

Chapter 3

Automates finis

3.1 Automates finis déterministes

3.1.1 Principe général

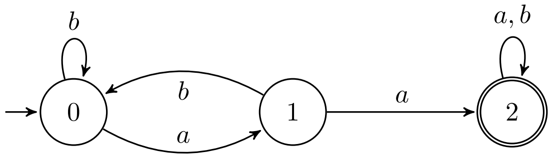


Figure 3.1: Un premier automate

Exercice 3.1.1. *Les mots abbaba, ababbaab et abba sont-ils acceptés par l'automate de la figure 3.1 ?*

Correction. *On a les parcours suivant dans l'automate :*

- *abbaba*

  $0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{a} 1 \notin \{2\}$, *donc non.*

- *ababbaab*

  $0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 2 \in \{2\}$, *donc oui.*

- *abba*

  $0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \notin \{2\}$, *donc non.*

Exercice 3.1.2. *Quel est le* **langage reconnu***, cad. l'ensemble des mots acceptés, par l'automate de la figure 3.1 ? Donner la réponse en français et sous forme d'expression rationnelle.*

Correction. *L'état* 0 *correspond à "on n'a pas croisé de aa", l'état* 1 *est "on vient de croiser un a isolé" et* 2 *est "On a croisé plus tôt aa". Le langage accepté est celui des mots ayant aa comme facteur, cad.* $\Sigma^* aa \Sigma^*$.

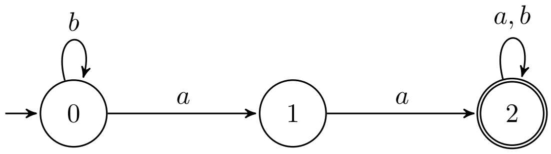


Figure 3.2: Un automate incomplet

Exercice 3.1.3. *Les mots bbbaababbaaba, bbabaab et baaaaaab sont-ils acceptés par l'automate de la figure 3.2 ?*

Correction. *On a les parcours suivant dans l'automate :*

- *bbbaababbaaba*

  $0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{babbaaba} 2 \in \{2\}$, *donc oui.*

- *bbabaab*

  $0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1 \overset{b}{\not\to}$, *donc non.*

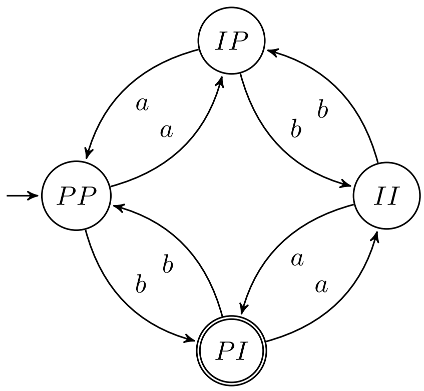
- *baaaaaab*

  $0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{aaaab} 2 \in \{2\}$, *donc oui.*

Exercice 3.1.4. *Quel est le langage reconnu par l'automate de la figure 3.2 ? Donner la réponse en français et sous forme d'expression rationnelle.*

Correction. *On reste dans l'état* 0 *tant qu'on a lu* b^**. On passe en* 1 *dès qu'on lit un premier* a*, et seul un autre* a *tombant immédiatement permet de passer en* 2*, où on accepte tout. Le langage reconnu est celui des mots qui contiennent des* a *et dont le premier est immédiatement suivi d'un deuxième, cad.* $b^*aa\Sigma^*$.

Exemple 3.1.1. *On veut écrire un automate reconnaissant le langage* $L = \{w \in \Sigma^* \mid |w|_a \text{ pair et } |w|_b \text{ impair}\}$*, cad. l'ensemble des mots avec un nombre pair de* a *et impairs de* b.



Exercice 3.1.5. *(**) En reprenant l'exemple 3.1.1, montrer que* $\forall w, w \in L \leftrightarrow$ *l'automate accepte* w*. Vous pouvez procéder par induction sur* w*, en utilisant un objectif un peu plus précis que celui fourni.*

Correction. *On est obligé, pour cette preuve, d'affiner la propriété qu'on veut prouver, en* *précisant le rôle des différents états :*

$$\begin{array}{ll} \forall w, & ((PP \xrightarrow{w} PP \text{ ssi. } |w|_a \text{ pair et } |w|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{w} IP \text{ ssi. } |w|_a \text{ impair et } |w|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{w} PI \text{ ssi. } |w|_a \text{ pair et } |w|_b \text{ impair}) \\ \wedge & (PP \xrightarrow{w} II \text{ ssi. } |w|_a \text{ impair et } |w|_b \text{ impair})) \end{array}$$

On procède par induction (droite) sur w. Dans le cas où $w = \epsilon$, la formule devient

$$\begin{array}{ll} & ((PP \xrightarrow{\epsilon} PP \text{ ssi. } |\epsilon|_a \text{ pair et } |\epsilon|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{\epsilon} IP \text{ ssi. } |\epsilon|_a \text{ impair et } |\epsilon|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{\epsilon} PI \text{ ssi. } |\epsilon|_a \text{ pair et } |\epsilon|_b \text{ impair}) \\ \wedge & (PP \xrightarrow{\epsilon} II \text{ ssi. } |\epsilon|_a \text{ impair et } |\epsilon|_b \text{ impair})) \end{array}$$

Qui se réécrit en

$$\begin{array}{ll} & \top \text{ ssi. } \top \\ \wedge & \bot \text{ ssi. } \bot \\ \wedge & \bot \text{ ssi. } \bot \\ \wedge & \bot \text{ ssi. } \bot \end{array}$$

qui s'évalue à $\top$. Le cas de base est donc vrai. Pour la récursion, on suppose que la propriété est vraie pour w, et on veut vérifier qu'elle est vraie pour $w.a$ et $w.b$. On se concentre, sans perte de généralité, sur $w.a$. On a 4 cas à étudier : $|w|_a$ pair et $|w|_b$ pair, $|w|_a$ impair et $|w|_b$ pair, $|w|_a$ pair et $|w|_b$ impair, et enfin $|w|_a$ impair et $|w|_b$ impair. On se concentre, encore une fois sans perte de généralité, sur le premier cas. On veut donc prouver

$$\begin{array}{ll} & ((PP \xrightarrow{wa} PP \text{ ssi. } |wa|_a \text{ pair et } |wa|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{wa} IP \text{ ssi. } |wa|_a \text{ impair et } |wa|_b \text{ pair}) \\ \wedge & (PP \xrightarrow{wa} PI \text{ ssi. } |wa|_a \text{ pair et } |wa|_b \text{ impair}) \\ \wedge & (PP \xrightarrow{wa} II \text{ ssi. } |wa|_a \text{ impair et } |wa|_b \text{ impair})) \end{array}$$

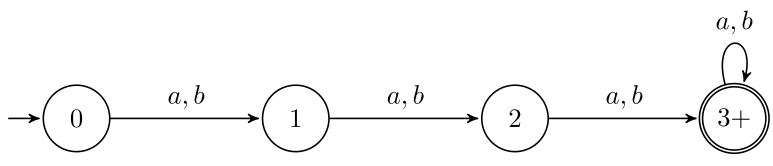
Puisqu'on est dans le cas $|w|_a$ pair et $|w|_b$ pair, $|wa|_a$ impair et $|wa|_b$ pair. De plus, l'hypothèse de récurrence nous dit que $PP \xrightarrow{w} PP$. Puisque $PP \xrightarrow{a} IP$, on a $PP \xrightarrow{wa} IP$. La propriété reste donc vraie en passant de w à $w.a$.

En généralisant cette preuve à tous les cas ignorés[1], on montre la propriété pour tout mot. Puisque le seul état terminal est PI, la correction de l'automate est un corrolaire.

Dans la série d'exercices qui suit, on utilisera comme alphabet $\Sigma = \{a, b\}$.

Exercice 3.1.6. *Donner un automate qui reconnaît le langage $\{w \in \Sigma^* \mid |w| \geq 3\}$.*

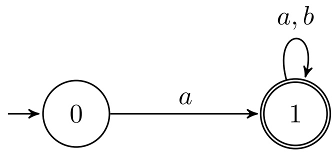
Correction. *On vérifie simplement la présence de 3 premières lettres avant d'autoriser n'importe quoi. Le nom des états représente le nombre de lettres lues :*



[1] Oui, la preuve formelle est un long et terrible chemin de croix parcouru par des héros et héroïnes du quotidien.

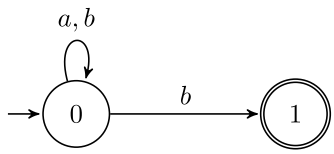
Exercice 3.1.7. *Donner un automate pour les mots qui commencent par a.*

Correction. *On vérifie simplement la présence d'un a au début du mot :*



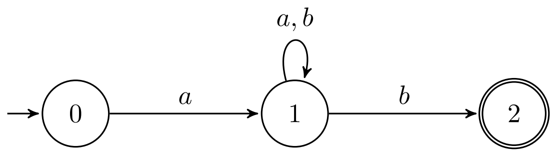
Exercice 3.1.8. *Donner un automate pour les mots qui finissent par b.*

Correction. *On vérifie simplement la présence d'un b qui ne peut être suivi par rien :*



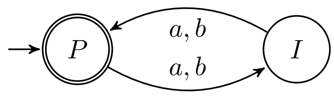
Exercice 3.1.9. *Donner un automate pour les mots qui commencent par a finissent par b.*

Correction. *En mélangeant les automates précédant, on obtient*



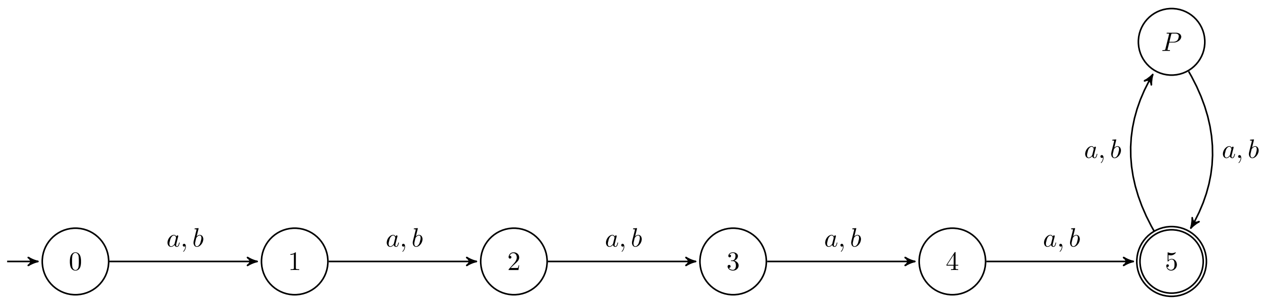
Exercice 3.1.10. *Donner un automate pour les mots de longueur paire.*

Correction. *On suit juste la parité de la longueur du mot donné :*

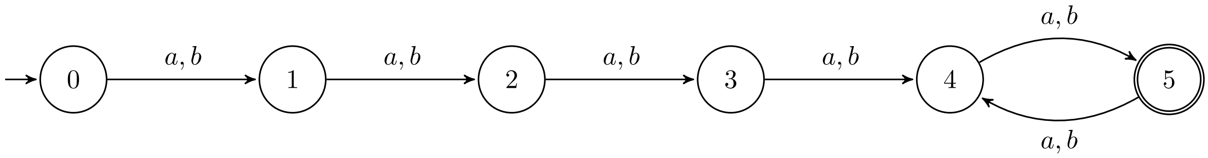


Exercice 3.1.11. *Donner un autuomate pour les mots de longueur impaire qui contiennent au moins 4 lettres.*

Correction. *On vérifie d'abord qu'on a 5 lettres, puis qu'on ajoute un nombre pair de lettres :*

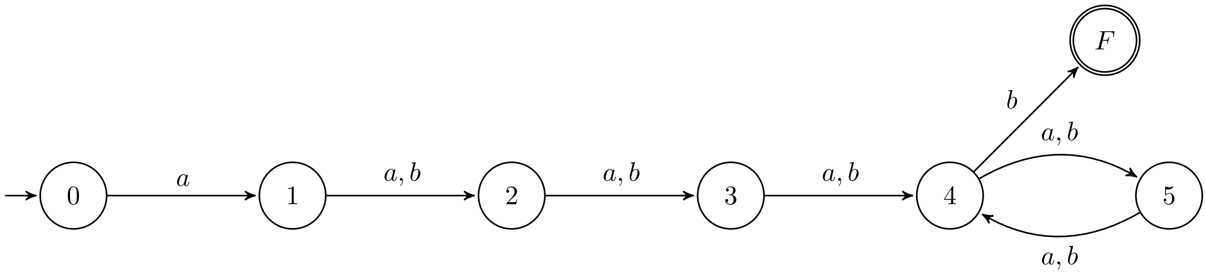


Notez qu'on peut ici fusionner les états 4 et I :



Exercice 3.1.12. *Donner un automate pour les mots de longueur impaire, qui contiennent au moins 4 lettres, commencent par a et finissent par b.*

Correction. *On contraint un peu l'automate précédent en forçant un a au début, et en ajoutant un état (on ne peut plus avoir 5 terminal, car sinon on va soit permettre de finir avec a, soit trop contraindre l'intérieur des mots) :*



3.1.2 Formalisation et implémentation

Exercice 3.1.13. *Donner la formalisation de l'automate de l'exemple 3.1.1 et vérifier qu'il accepte le mot $aababba$.*

Correction. *On a le 5-uplet suivant :*

- $Q = \{PP, IP, PI, II\}$
- $\Sigma = \{a, b\}$
- $q_0 = PP$
- $F = \{PI\}$
- $\delta(PP, a) = IP; \delta(PP, b) = PI; \delta(IP, a) = PP; \delta(IP, b) = II; \delta(PI, a) = II; \delta(PI, b) = PP; \delta(II, a) = PI; \delta(II, b) = IP.$

On a alors le calcul suivant :

$$\begin{aligned}
& \delta^*(PP, aababba) \\
= \; & \delta^*(\delta(PP, a), ababba) \\
= \; & \delta^*(IP, ababba) \\
= \; & \delta^*(\delta(IP, a), babba) \\
= \; & \delta^*(PP, babba) \\
= \; & \delta^*(PI, abba) \\
= \; & \delta^*(II, bba) \\
= \; & \delta^*(IP, ba)
\end{aligned}$$

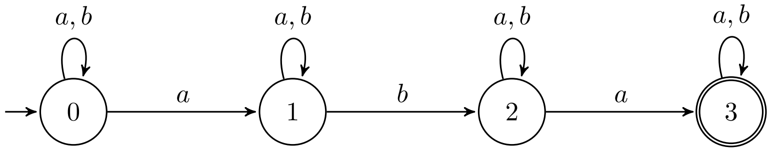
$= \delta^*(II, a)$
$= \delta^*(PI, \epsilon)$
$= PI \in \{PI\}$, *donc oui.*

3.2 Automates finis non-déterministes

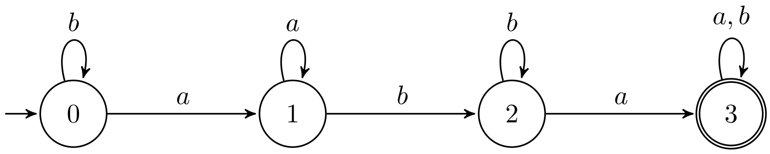
3.2.1 Principe général

Exercice 3.2.1. *Donner une regex et un automate fini pour le langage* $L = \{w \mid aba$ *est un sous-mot de* $w\}$.

Correction. *On peut fournir une regex hautemennt non-déterministe :* $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$. *L'automate non-déterministe équivalent est*



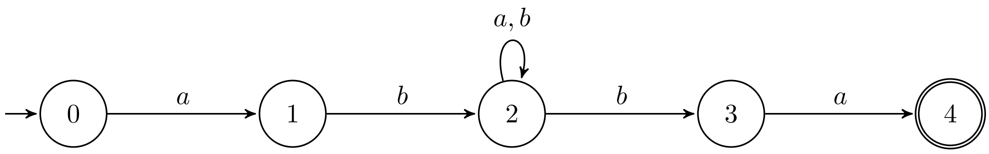
Assez étonnamment, on peut concevoir un automate déterministe qui reconnaît le même langage avec le même nombre d'états.



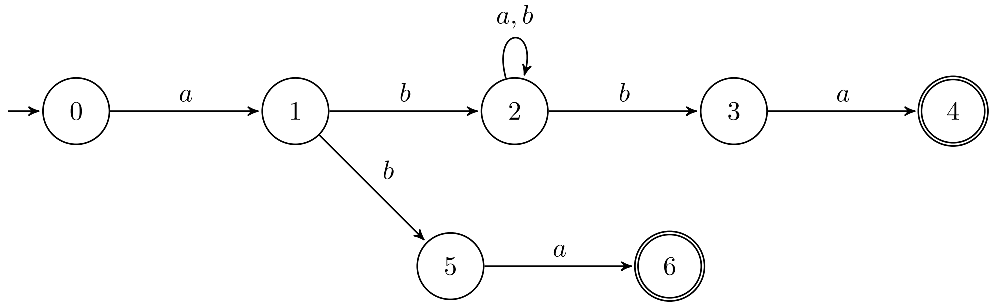
Dans cette version, on enlève certaines transitions pour forcer que les transitions horizontales soient prises dès que possible. Cet automate permet de déduire une regex moins claire car moins ambigüe, et donc plus efficace : $b^* a a^* b b^* a \Sigma^*$.

Exercice 3.2.2. *Donner une regex et un automate fini pour le langage des mots qui commencent par* ab *et finissent par* ba.

Correction. *On pourrait d'abord avoir envie de répondre*



Cependant, cet automate n'accepte pas aba, *qui fait pourtant partie du langage. Il s'agit cependant de la seule exception, qu'on peut rajouter à la main (l'état* 5 *représente le choix que le premier* b *est à la fois celui du début et de la fin, et donc que le mot lu est* aba*). On aurait pu être encore plus flemmard et rajouter deux états, dont un initial, plutôt que de partager* 0 *et* 1.



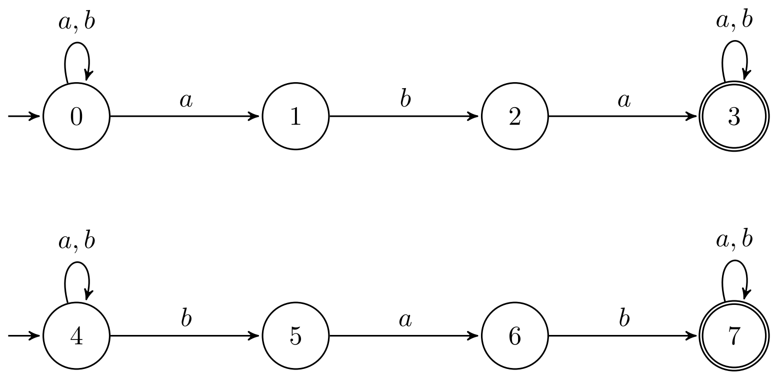
La regex correspondant à cet automate est $a(b\Sigma^*ba + ba)$. On aurait pu plus simplement utiliser la version défactorisée $ab\Sigma^*ba + aba$.

3.2.2 Formalisation et implémentation

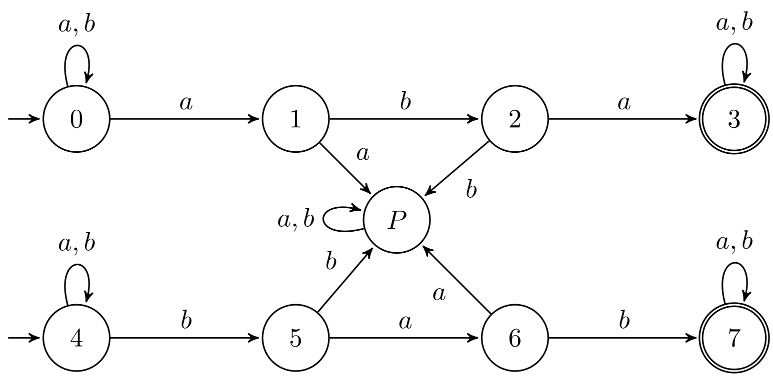
3.3 Transformation d'automates

3.3.1 Complétion

Exercice 3.3.1. *Compléter l'automate suivant :*



Correction. *On obtient*



Exercice 3.3.2. *Donner la formalisation de la complétion d'un automate non-déterministe.*

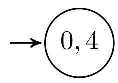
Correction. *Soit un automate non-déterministe* $\langle Q, \Sigma, I, F, \delta \rangle$*, sa version complétée est* $\langle Q \cup \{P\}, \Sigma, I, F, \delta' \rangle$*, avec*

$$\begin{cases} \delta'(q,a) = \delta(q,a) & \text{si } \delta(q,a) \neq \emptyset, \\ \delta'(q,a) = \{P\} & \text{si } \delta(q,a) = \emptyset, \\ \delta'(P,a) = \{P\} & \text{pour tout } a \in \Sigma \end{cases}$$

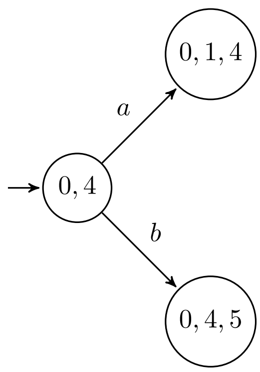
3.3.2 Déterminisation

Exercice 3.3.3. *Déterminiser l'automate de l'exercice 3.3.1.*

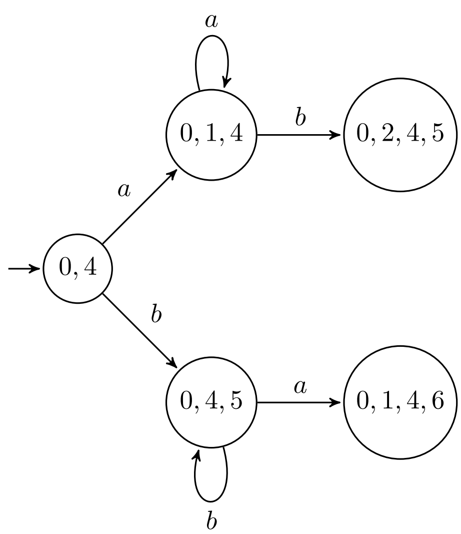
Correction. *L'automate à déterminiser a deux états initiaux : 0 et 4. On commence donc en créant un état initial représentant 0 et 4 :*



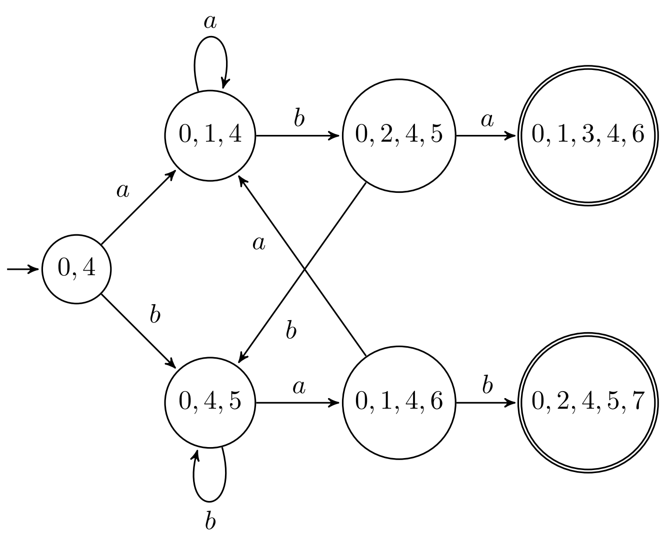
La lettre a permet de passer de 0 à 0 ou 1, et fait boucler sur 4. Au total, quand on est en 0, 4 et qu'on lit un a, on a le choix entre 0, 1 et 4. De même, en lisant un b, on peut aller en 0, 4 ou 5. On obtient donc les premières transtitions suivantes :



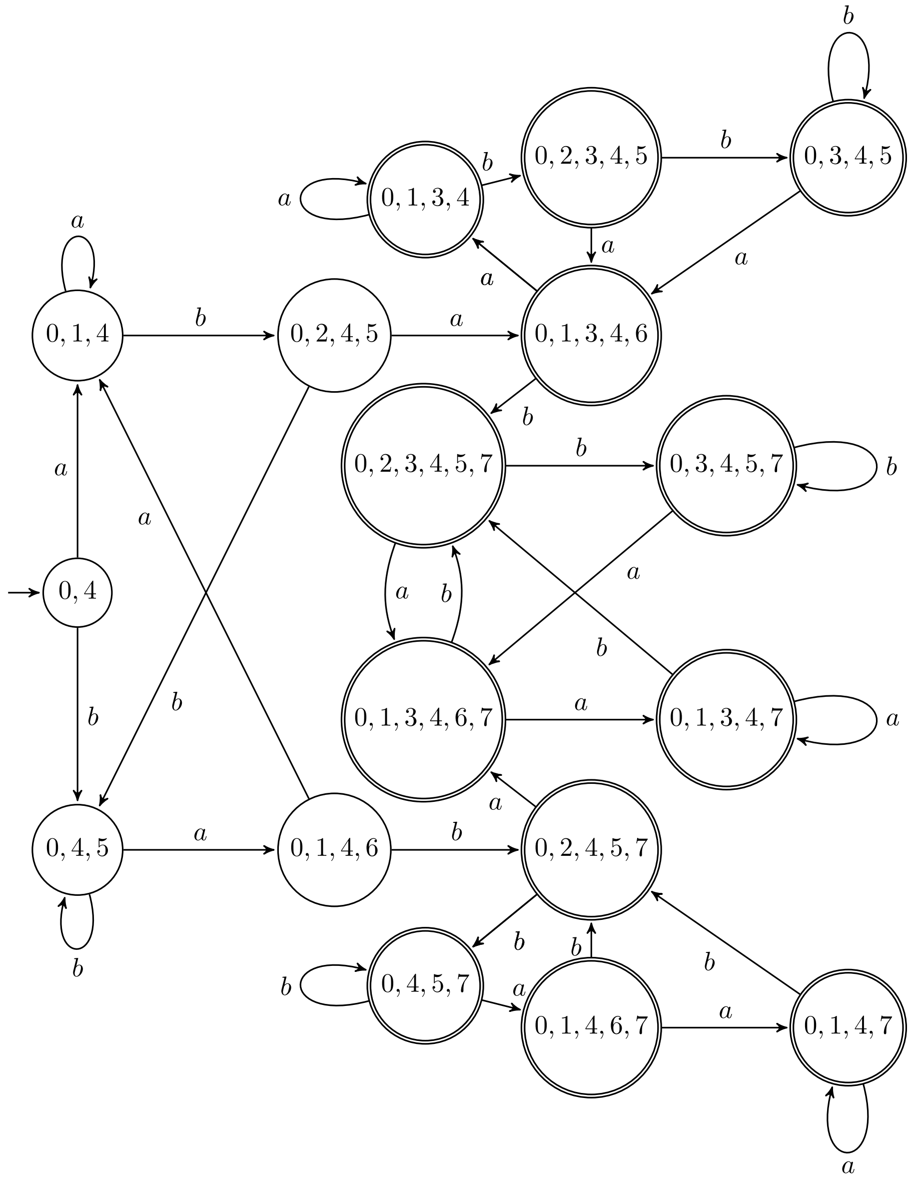
On itère :



Quand on lit un a depuis 0, 2, 4 *ou* 5*, on peut atteindre* 0, 1, 3, 4 *ou* 6*, on crée donc l'état correspondant.* 3 *étant un état terminal dans l'automate initial,* $0, 1, 3, 4, 6$ *l'est aussi. Un* b *par contre nous emmène seulement en* 0, 4 *ou* 5*. Cette état existant déjà dans notre automate, on pointe dessus. En faisant le même raisonnement avec* $0, 1, 4, 6$*, on obtient*

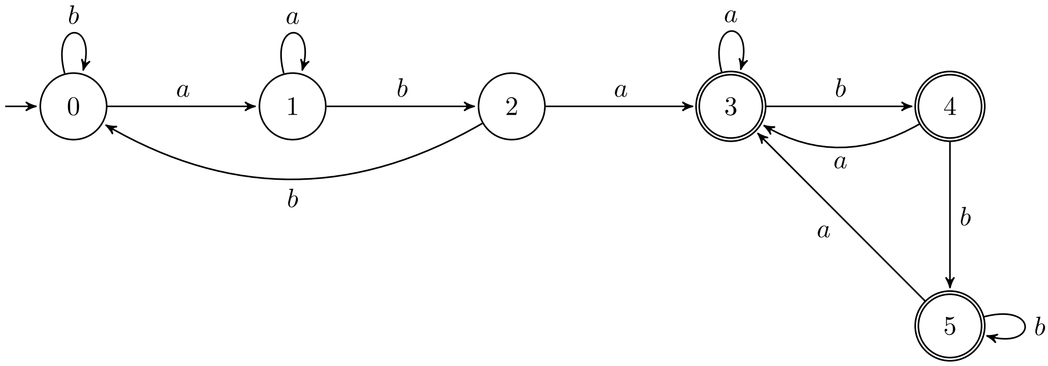


En continuant comme ça, on arrive à



3.3.3 Minimisation

Exercice 3.3.4. *Minimiser (en utilisant l'algorithme) l'automate*



Correction. *On a* $N = \{0, 1, 2\}$ *d'un côté, et* $F = \{3, 4, 5\}$ *de l'autre. On va d'abord s'intéresser à* N.

- *0 vs 1*
  - $\delta(0, a) = 1 = \delta(1, a)$
  - $\delta(1, b) = 0 \in N$ *et* $\delta(1, b) = 2 \in N$

  $\Rightarrow$ *0 et 1 sont d'accord*
- *0 vs 2*
  - $\delta(0, a) = 1 \in N$ *et* $\delta(2, a) = 3 \in F$

  $\Rightarrow$ *0 et 2 ne sont pas d'accord, on va devoir les séparer*
- *Pas besoin de tester 1 vs 2, puisqu'on sait déjà que 1 va être avec 0, et donc séparé de 2.*

On peut donc séparer N *en* $N_1 = \{0, 1\}$ *et* $N_2 = \{2\}$. *On a maintenant le choix d'étudier* N_1 *ou* F. *On choisit (un peu bêtement)* F.

- *3 vs 4*
  - $\delta(3, a) = 3 = \delta(4, a)$
  - $\delta(3, b) = 4 \in F$ *et* $\delta(4, b) = 5 \in F$

  $\Rightarrow$ *3 d'accord avec 4*
- *3 vs 5*
  - $\delta(3, a) = 3 = \delta(5, a)$
  - $\delta(3, b) = 4 \in F$ *et* $\delta(5, b) = 5 \in F$

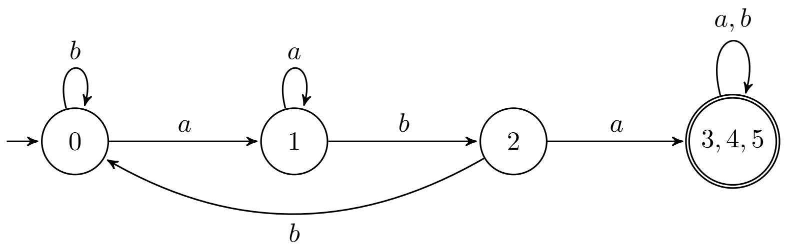
  $\Rightarrow$ *3 d'accord avec 5*
- *Pas besoin de tester 4 vs 5, puisqu'ils sont tous les deux d'accord avec 3.*

On laisse donc F *en l'étant. On s'intéresse maintenant à* N_1 *:*

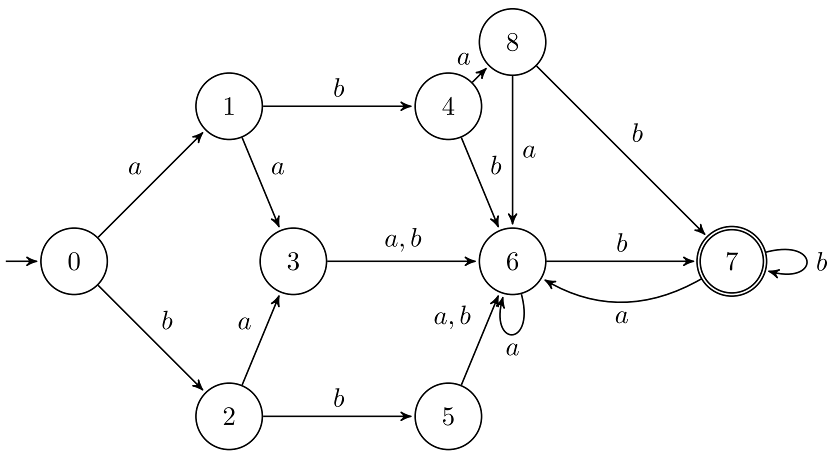
- *0 vs 1*
  - $\delta(0, a) = 1 = \delta(1, a)$
  - $\delta(0, b) = 0 \in N_1$ *et* $\delta(1, b) = 2 \in N_1$

  $\Rightarrow$ *On sépare* 0 *et* 1.

On a donc maintenant $N_3 = \{0\}$, $N_4 = \{1\}$, $N_2 = \{2\}$ et $F = \{3, 4, 5\}$. Puisqu'on a eu un changement (casser N_1 en deux), on devrait refaire la vérification pour F faite plus haut, au cas où le résultat aurait changé. Ceci dit, on peut remarquer qu'aucun état de F n'envoie directement en 0 ou 1, et donc que F resterait insécable. On peut donc en fusionner les états et obtenir



Exercice 3.3.5. *Minimiser l'automate suivant :*



Correction. *On divise donc notre ensemble d'états en $F = \{7\}$ et $N = \{0, 1, 2, 3, 4, 5, 6, 8\}$. On ne va manifestement pas pouvoir affiner F, contrairement à N. Pour ça, on va regarder toutes les paires d'états de N et, à chaque fois, vérifier si les deux états ont des désaccords en lisant a ou b :*

- *0 vs. 1*
  - $\delta(0, a) = 1$ *et* $\delta(1, a) = 3$. *Pour l'instant, 1 et 3 appartiennent tous les deux à 3, ce qui veut dire qu'on suppose que 1 accepte un mot w ssi. 3 accepte w. On peut donc conclure que, de ce qu'on sait, 0 accepte un mot $a.w$ ssi. 1 accepte $a.w$.*
  - $\delta(0, b) = 2$ *et* $\delta(1, b) = 4$. *2 et 4 appartiennent à la même classe, on suppose donc pour l'instant que 0 accepte un mot $b.w$ ssi. 1 accepte un mot $b.w$.*
  - ⇒ *Pour l'instant, on conserve notre hypothèse selon laquelle 0 et 1 acceptent le même langage. Ils sont donc dans la même classe.*
- *0 vs. 2*
  - $\delta(0, a) = 1$ *et* $\delta(2, a) = 3$, *1 et 3 sont dans la même classe.*
  - $\delta(0, b) = 2$ *et* $\delta(2, b) = 5$. *2 et 5 appartiennent à la même classe.*

⇒ *Pour l'instant, on conserve notre hypothèse selon laquelle* 0 *et* 2 *acceptent le même langage. 2 rejoint donc la classe de 0 et 1.*

Pour préserver ce qu'il me reste de santé mentale, je ne vais pas détailler tout le reste.

- *0 vs. 3 : compatibilité. 3 est donc mis avec 0, 1 et 2.*
- *0 vs. 4 : compatibilité. 4 est donc mis avec 0, 1, 2 et 3.*
- *0 vs. 5 : compatibilité. 5 est donc mis avec 0, 1, 2 et 3, 4.*
- *0 vs. 6 : incompatibilité. 6 est mis à part.*
- *0 vs. 8 : incompatibilité. 8 est mis à part.*

On sait que 6 et 8 ne vont pas dans l'ensemble $0, 1, 2, 3, 4, 5$, *mais il reste à déterminer s'ils vont ensemble ou s'ils restent chacun dans leur coin.*

- *6 vs. 8 : compatibilité, on classe* 6 *et* 8 *ensemble.*

⇒ *On sépare* N *en* $N_1 = \{0, 1, 2, 3, 4, 5\}$ *et* $N_2 = \{6, 8\}$

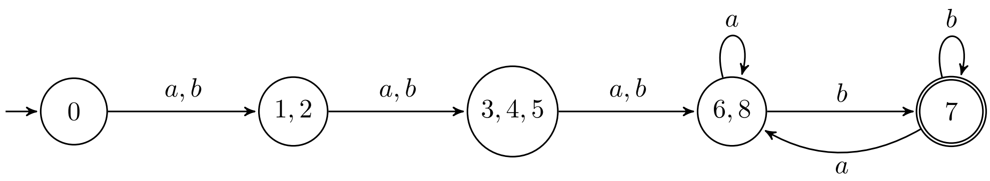
On peut noter que 6 *et* 8 *pointent exactement sur les mêmes états pour les deux lettres, on peut donc être sûr qu'on n'arrivera pas à les séparer, quoi qu'il arrive à* N_1. *On s'intéresse donc à ce dernier.*

- *Les seuls états qu'on va sortir sont ceux qui pointent vers* N_2, *cad.* $\{3, 4, 5\}$. *On peut en effet vérifier qu'ils sont tous incompatibles avec* 0, 1 *et* 2, *et qu'ils sont compatibles entre eux.*

On divise donc N_1 *en* $N_3 = \{0, 1, 2\}$ *et* $N_4 = \{3, 4, 5\}$. *On regarde* N_3 *:*

- 1 *et* 2 *vont vers* N_4 *avec* b *alors que* 0 *reste en* N_3. 1 *et* 2 *envoient vers le même état avec* a.

On divise N_3 *en* $N_5 = \{0\}$ *et* $N_6 = \{1, 2\}$. *En plus de ces deux ensembles, on a* $N_4 = \{3, 4, 5\}$, $N_2 = \{6, 8\}$ *et* $F = \{7\}$. *En repassant sur chacun de ces ensembles, on se rend compte qu'on ne peut faire aucune séparation. On peut donc fusionner les états de chaque classe et obtenir*



Cad. l'automate acceptant les mots d'au moins 4 lettres terminant par un b. *La version initiale de l'automate était non-minimale en ce qu'elle distinguait en plusieurs états la lecture de* a *ou de* b *alors qu'au fond, on ne s'intéressait qu'à la longueur.*

3.4 Limite de la reconnaissance par automates finis

3.5 Propriétés de clôture

3.5.1 Union

3.5.2 Concaténation

3.5.3 Itération

Exercice 3.5.1. *Est-il nécessaire dans la transformation de rendre non-terminaux les états qui l'étaient dans l'automate original ?*

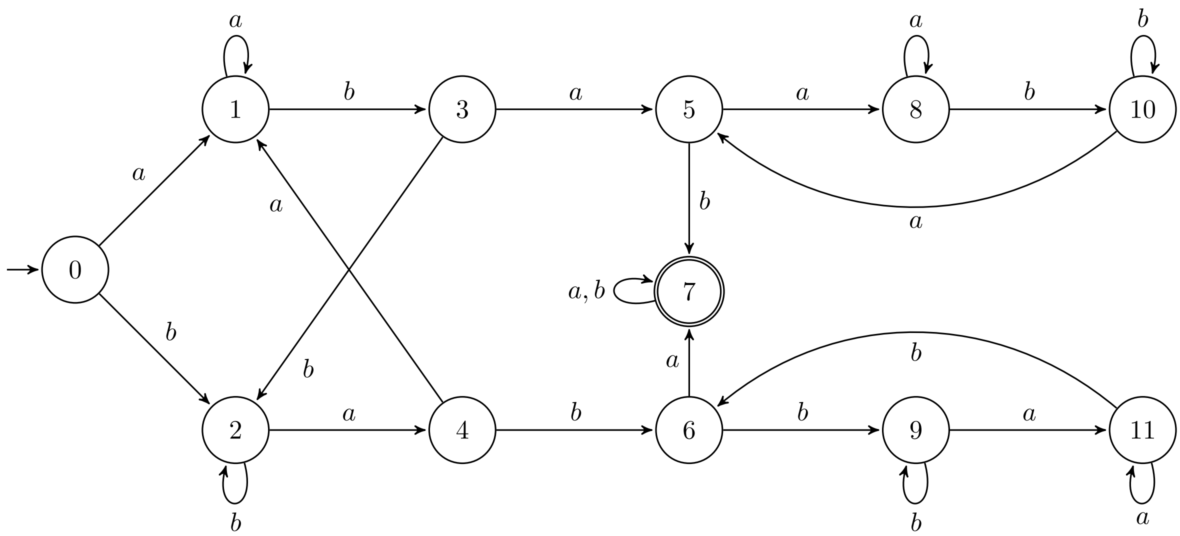
Correction. *Non, puisque toute transition vers un état anciennement terminal est copiée pour mener à S, qui l'est. Toute lecture terminant dans un état de F peut donc également s'achever dans S, ce qui implque que laisser les états de F terminaux permettrait juste de choisir son état d'acceptation.*

Exercice 3.5.2. *Pourquoi était-il indispensable d'introduire un nouvel état dans la transformation ?*

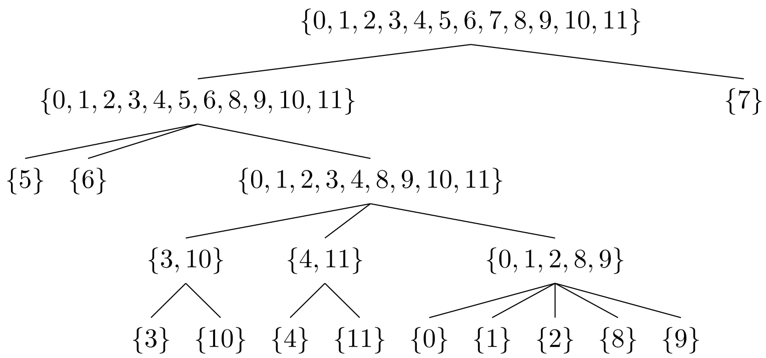
Correction. *L^* contient le mot vide pour tout langage L, on a donc besoin d'un état initial terminal. Cependant, rendre l'état initial d'origine terminal peut parasiter les calculs en faisant accepter des mots n'appartenant pas à L^*. Dans l'exemple 3.5.2 du poly, rendre PP initial plutôt que créer S ferait accepter aa, qui n'appartient pas à L^* (tout mot de L^*, à l'exception d'ϵ, contient au moins un b).*

3.5.4 Intersection

Exercice 3.5.3. *Minimiser l'automate*

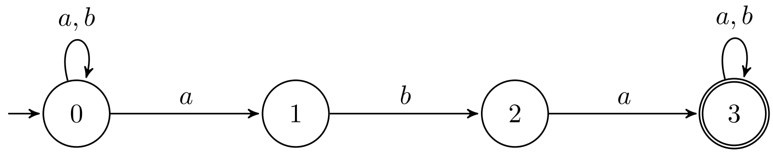


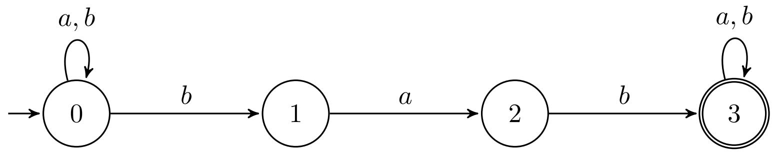
Correction. *On obtient le partitionnement suivant :*



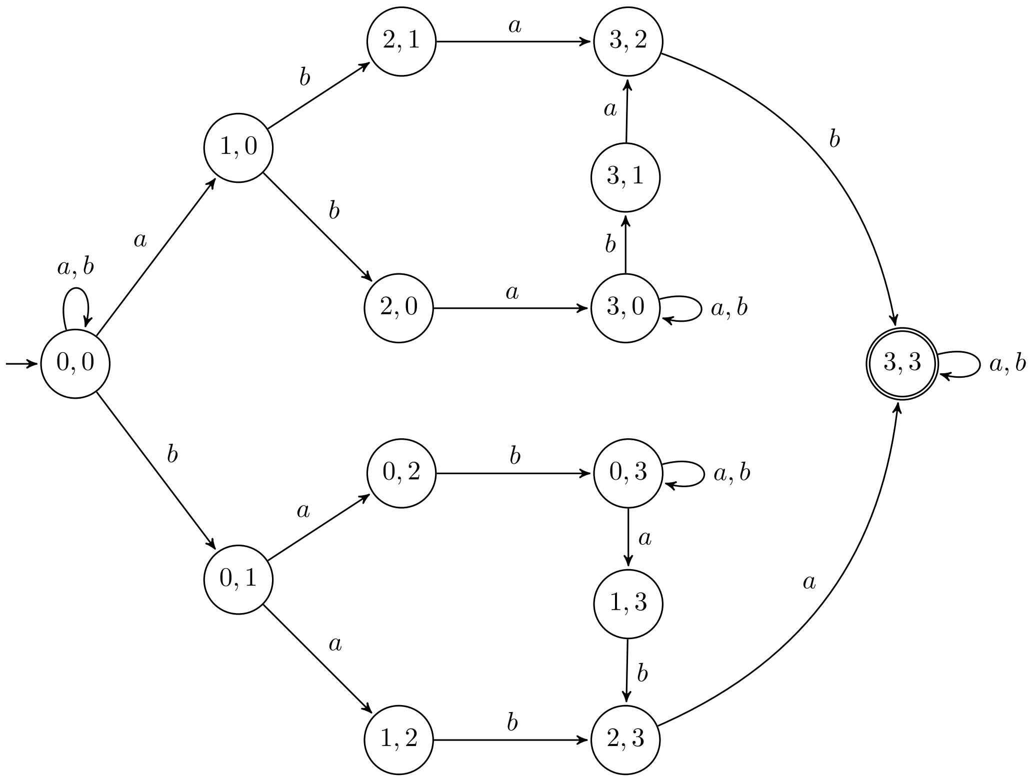
L'automate était donc déjà minimal.

Exercice 3.5.4. *Appliquer l'algorithme d'intersection sur les automates*





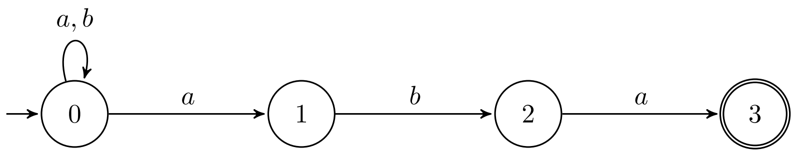
Correction. *On obtient l'automate suivant :*



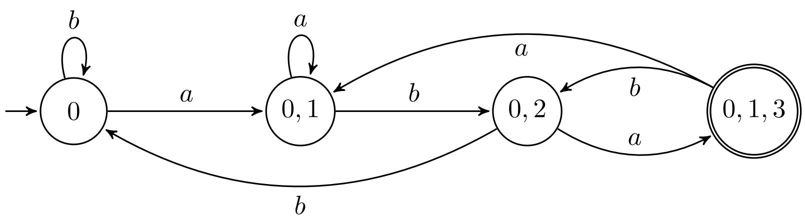
3.5.5 Complémentaire

Exercice 3.5.5. *Donner un automate des mots ne terminant pas par aba.*

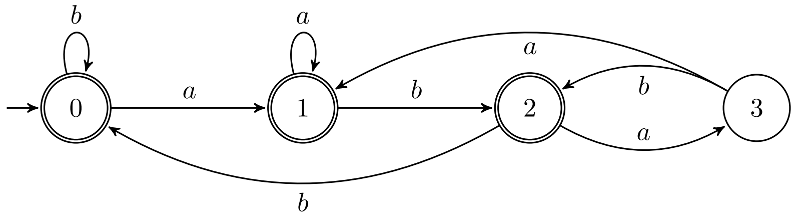
Correction. *On donne d'abord un automate des mots terminant par aba :*



Il faut d'abord déterminiser l'automate. On obtient



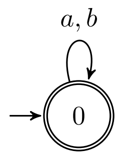
Cet automate est déjà complet, on peut donc procéder à l'inversion des états terminaux (et *renommer les états). Le langage des mots ne terminant pas par aba est donc reconnu par*



3.5.6 Conséquences pour les langages non-reconnus

Exercice 3.5.6. *Montrer que le langage $\{w \in \{a,b,c\}^* \mid |w|_a = |w|_b\}$ est non-rationnel.*

Correction. *On remarque que $L \cap \{a,b\}^*$, cad. L restraint à a et b, est égal à $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$, qui est non-rationnel. $\{a,b\}^*$ est lui rationnel, car reconnu par l'automate*



Si L était rationnel, par clôture, $L \cap \{a,b\}^* = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ devrait l'être aussi, ce qui n'est pas le cas. On en déduit donc que L n'est pas rationnnel.

Chapter 4

Théorème de Kleene

4.1 Des expressions rationnelles aux automates

4.1.1 Traduction récursive

4.1.2 Traduction linéaire

Exercice 4.1.1. *Utiliser l'algorithme de Glushkov pour traduire en automate l'expression* $e = (bb)^*(b(a+b)^*)^*$

Correction. *On commence par réécrire* e *en* $(b_1b_2)^*(b_3(a_4+b_5)^*)^*$. *On peut ensuite calculer les images des fonctions* E, D, F *et* P.

L'expression contient-elle le mot vide, et l'état initial est-il terminal ?

$$\begin{aligned} & E((b_1b_2)^*(b_3(a_4+b_5)^*)^*) \\ = \; & E((b_1b_2)^*) \wedge E((b_3(a_4+b_5)^*)^*) \\ = \; & \top \wedge \top \\ = \; & \top \end{aligned}$$

Quelles sont les premières lettres des mots de l'expression, et donc les transitions partant de l'état initial ?

$$\begin{aligned} & D((b_1b_2)^*(b_3(a_4+b_5)^*)^*) \\ = \; & D((b_1b_2)^*) \cup D((b_3(a_4+b_5)^*)^*) \\ = \; & D(b_1b_2) \cup D(b_3(a_4+b_5)^*) \\ = \; & D(b_1) \cup D(b_3) \\ = \; & \{b_1\} \cup \{b_3\} \\ = \; & \{b_1, b_3\} \end{aligned}$$

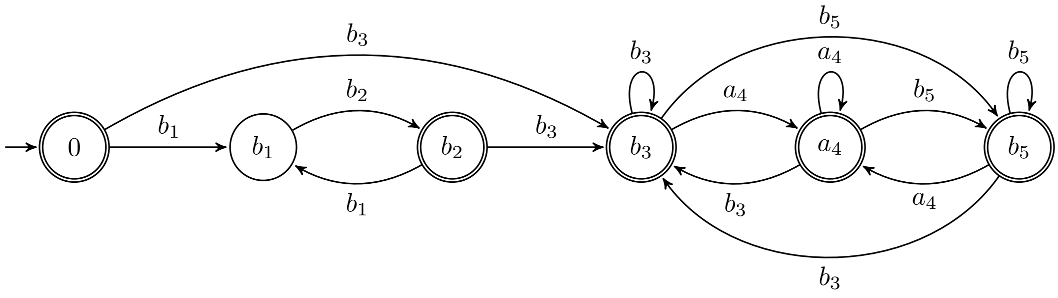
Quelles sont les dernières lettres des mots de l'expression, et donc les états terminaux ?

$$
\begin{aligned}
& F((b_1b_2)^*(b_3(a_4+b_5)^*)^*) \\
= \; & F((b_1b_2)^*) \cup F((b_3(a_4+b_5)^*)^*) \\
= \; & F(b_1b_2) \cup F(b_3(a_4+b_5)^*) \\
= \; & D(b_2) \cup F(b_3) \cup F((a_4+a_5)^*) \\
= \; & D(b_2) \cup F(b_3) \cup F(a_4+a_5) \\
= \; & D(b_2) \cup F(b_3) \cup F(a_4) \cup F(a_5) \\
= \; & \{b_2\} \cup \{b_3\} \cup \{a_4\} \cup \{a_5\} \\
= \; & \{b_2, b_3, a_4, a_5\}
\end{aligned}
$$

Et enfin, quelles lettres peuvent se suivre dans les mots de l'expression ?

$$
\begin{aligned}
& P((b_1b_2)^*(b_3(a_4+b_5)^*)^*) \\
= \; & P((b_1b_2)^*) \cup P((b_3(a_4+b_5)^*)^*) \cup F((b_1b_2)^*).D((b_3(a_4+b_5)^*)^*) \\
= \; & P((b_1b_2)^*) \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2\}.\{b.3\} \\
= \; & P((b_1b_2)^*) \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2.b_3\} \\
= \; & P(b_1b_2) \cup F(b_1b_2).D(b_1b_2) \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2.b_3\} \\
= \; & P(b_1b_2) \cup \{b_2.b_1\} \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2.b_3\} \\
= \; & P(b_1) \cup P(b_2) \cup F(b_1).D(b_2) \cup \{b_2.b_1\} \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2.b_3\} \\
= \; & \emptyset \cup \emptyset \cup \{b_1.b_2\} \cup \{b_2.b_1\} \cup P((b_3(a_4+b_5)^*)^*) \cup \{b_2.b_3\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3\} \cup P((b_3(a_4+b_5)^*)^*) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3\} \cup P(b_3(a_4+b_5)^*) \cup F(b_3(a_4+b_5)^*).D(b_3(a_4+b_5)^*) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3\} \cup P(b_3(a_4+b_5)^*) \cup \{b_3, a_4, b_5\}.\{b_3\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3\} \cup P(b_3(a_4+b_5)^*) \cup \{b_3.b_3, a_4.b_3, b_5.b_3\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, a_4.b_3, b_5.b_3\} \cup P(b_3(a_4+b_5)^*) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, a_4.b_3, b_5.b_3\} \cup P(b_3) \cup P((a_4+b_5)^*) \cup D(b_3).F((a_4+b_5)^*) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, a_4.b_3, b_5.b_3\} \cup \emptyset \cup P((a_4+b_5)^*) \cup \{b_3\}.\{a_4, b_5\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, b_3.a_4, b_3.b_5, a_4.b_3, b_5.b_3\} \cup P((a_4+b_5)^*) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, b_3.a_4, b_3.b_5, a_4.b_3, b_5.b_3\} \cup P(a_4+b_5) \cup F(a_4+b_5).D(a_4+b_5) \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, b_3.a_4, b_3.b_5, a_4.b_3, b_5.b_3\} \cup P(a_4) \cup P(b_5) \cup \{a_4, b_5\}.\{a_4, b_5\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, b_3.a_4, b_3.b_5, a_4.b_3, b_5.b_3\} \cup \emptyset \cup \emptyset \cup \{a_4.a_4, a_4.b_5, b_5.a_4, b_5.b_5\} \\
= \; & \{b_1.b_2, b_2.b_1, b_2.b_3, b_3.b_3, b_3.a_4, b_3.b_5, a_4.b_3, b_5.b_3, a_4.a_4, a_4.b_5, b_5.a_4, b_5.b_5\}
\end{aligned}
$$

On obtient donc l'automate suivant :



qui se "déspécialise" en