DM de Bases formelles du TAL

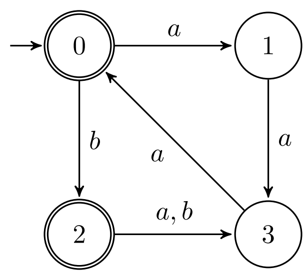
Pierre-Léo Bégay

À me rendre le 1er mai 2020

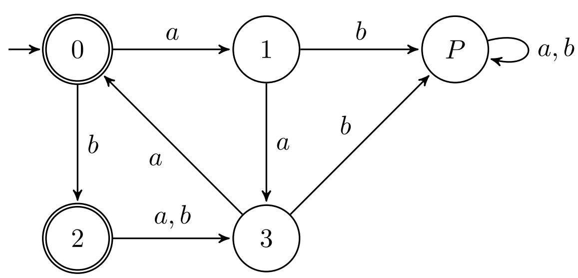
1 Automates

1.1 Complétion

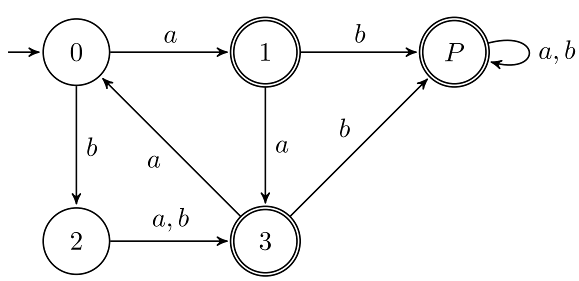
Question Donnez un automate qui reconnaît le complémentaire du langage reconnu par celui-ci :



Correction L'algorithme de complémentation d'un automate ne s'applique qu'à un automate complet. On s'intéresse donc, après application de l'algorithme classique (ajout d'un état poubelle), à l'automate suivant :

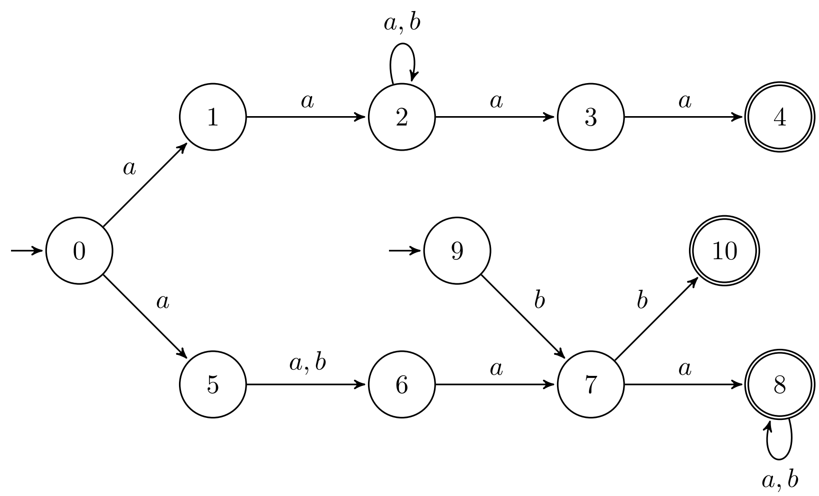


On inverse alors les états terminaux et non-terminaux pour obtenir l'automate reconnaissant le langage complémentaire du précédent :

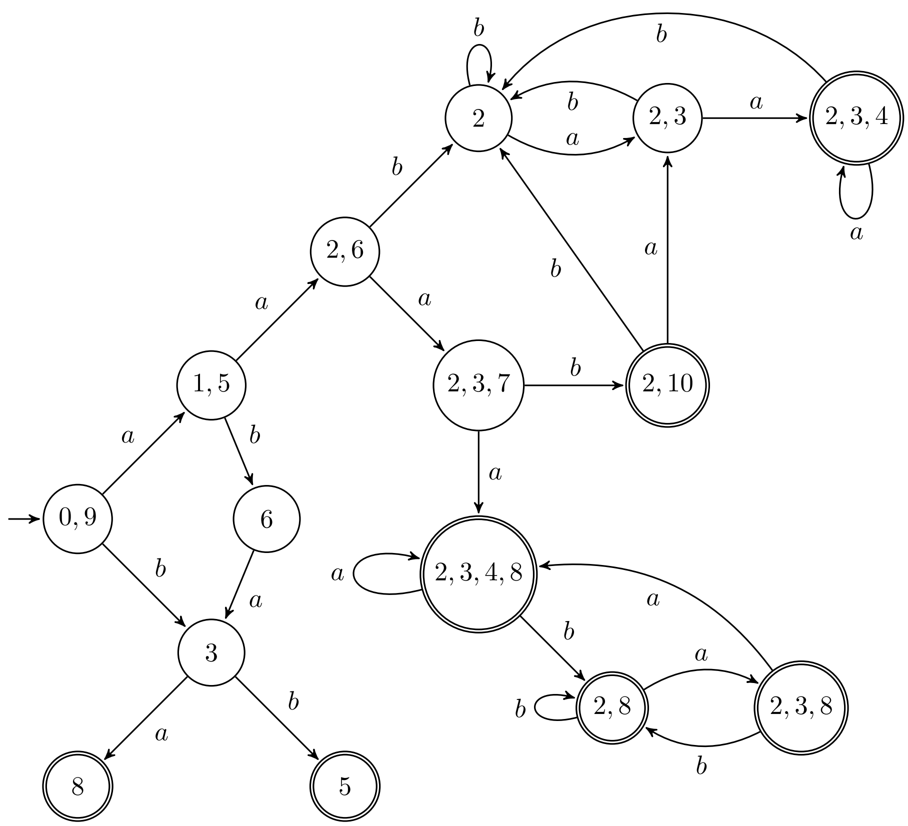


1.2 Déterminisation

Question 1 Déterminisez l'automate suivant :



Correction En appliquant l'algorithme vu en cours, on obtient



Question 2 Donnez une expression rationnelle décrivant le langage reconnu par l'automate.

Correction L'automate étant relativement linéaire[1], pas la peine d'utiliser McNaughton et Yamada. Tout parcours acceptant va de 0 à 4, 0 à 10, 0 à 8, 9 à 10 ou 9 à 8 (4 n'est pas accessible depuis 9). Respectivement, ça donne

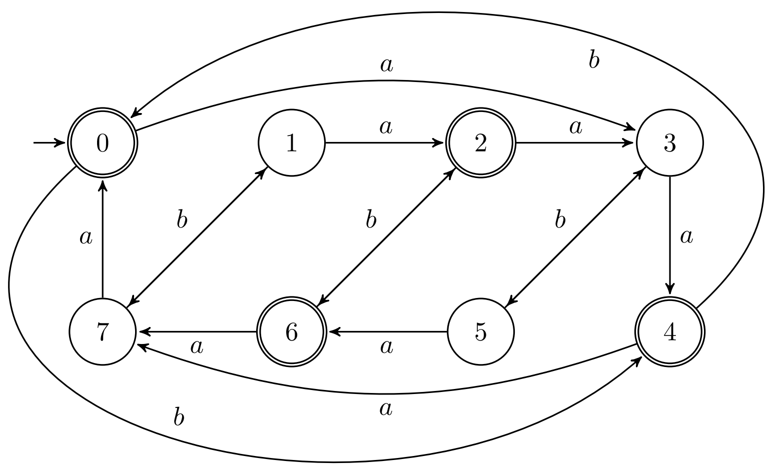
$$aa\Sigma^*aa + a(a+b)ab + a(a+b)aa\Sigma^* + bb + ba\Sigma^*$$

On peut factoriser ça en

$$aa\Sigma^*aa + (a(a+b)a+b)(b+a\Sigma^*)$$

1.3 Minimisation

Question 1 Minimisez l'automate suivant :



Correction On divise les états en deux classes, selon qu'ils soient terminaux ou non :

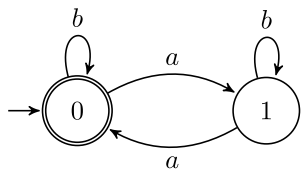
- $C_T = \{0, 2, 4, 6\}$
- $C_{nT} = \{1, 3, 5, 7\}$

On remarque que

- en lisant un a,
  - les états de C_T envoient vers un état de C_{nT}
  - les états de C_{nT} envoient vers un état de C_T
- en lisant un b,
  - les états de C_T envoient vers un état de C_T
  - les états de C_{nT} envoient vers un état de C_{nT}

[1]Dans le sens où on repère les quelques "couloirs" qui le composent

Il n'y a donc pas moyen de "casser" (raffiner) les classes d'états. On fusionne donc 0, 2, 4 et 6 d'une part, et 1, 3, 5 et 7 d'autre part :



Question 2 Quel est le langage reconnu par l'automate ?

Correction Sur l'automate minimisé, on observe facilement que le langage reconnu est celui des mots contenant un nombre pair de a, décrit notamment par l'expression $b^*(ab^*ab^*)^*$ ou, de façon plus bourrine, $(b^*ab^*ab^*)^*$.

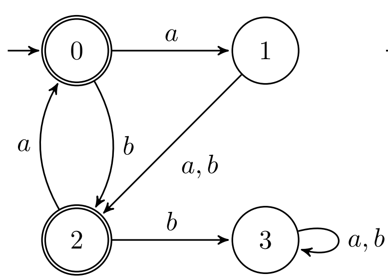
Bonus Essayez d'expliquer comment l'automate non-minimal reconnaissait le langage, et en quoi il n'était pas optimal

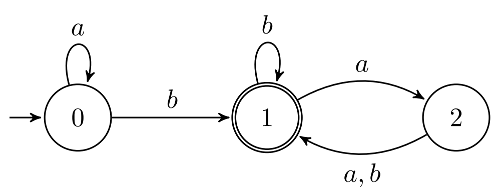
Correction Comme on l'a déjà vu dans la minimisation, les états 0, 2, 4 et 6 d'une part, et 1, 3, 5 et 7 d'autre part sont équivalents. En effet, au court de la lecture d'un mot, on est dans un état pair si et seulement si on a lu un nombre pair de a, tandis qu'on est dans un état impair si et seulement si on a lu un nombre impair de a.

Dans l'automate original, on s'amuse donc à sauter de façon très gratuite entre états équivalents sans grand sens. Par exemple, quand on lit un b en 1 on va en 7 et inversement au lieu de rester en place, parce que pourquoi pas. Il fonctionne donc comme quatre copies de sa version minimale mélangées de façon chaotique.

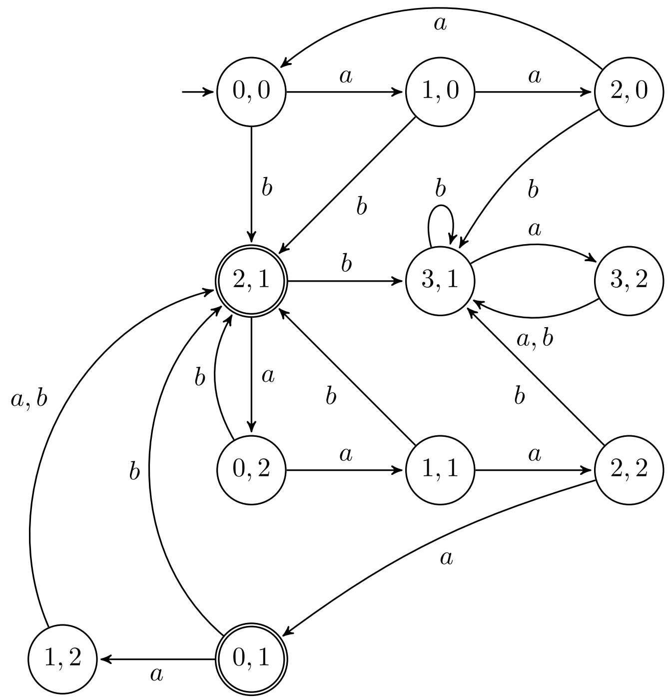
1.4 Automate produit

Question Donnez un automate qui reconnaît l'intersection des langages reconnus par les deux automates suivant :





Correction En appliquant le produit d'automates, on obtient :



2 Langages intrinsèquement ambigus

Dans cet exercice, on s'intéressera **uniquement** aux **grammaires de type 2** (ou grammaires algébriques). On rappelle que ces grammaires n'acceptent que les règles de la forme $A \to \gamma$, avec $\gamma \in (\Sigma \cup V)^*$.

Vos réponses devront être accompagnées d'une justification légère (de l'ordre d'une ou deux phrases) expliquant comment la grammaire donnée génère le langage de la question.

Indice Il n'est pas interdit de penser au cours sur les propriétés de clôture des langages réguliers.

Question 0

Donnez une grammaire qui reconnaît le langage $L_0 = \{a^n b^n \mid n \in \mathbb{N}\}$

Correction On pose $G_0 = \left\langle \{a, b\}, \{S\}, S, \{S \to aSb \mid \epsilon \text{ (règles 1 et 2)}\} \right\rangle$
Toute dérivation dans cette grammaire sera de la forme

$$S \to aSb \to aaSbb \to aaaSbbb \to ...$$

Plus précisément :

$$S \to_1^n a^n S b^n \to_2 a^n \epsilon b^n = a^n b^n$$

Où $\to_1^n$ représente n applications de la règle 1 ($S \to aSb$) et $\to_2$ correspond à une application de la règle 2 ($S \to \epsilon$)

Question 1

Donnez une grammaire qui reconnaît le langage $L_1 = \{a^n b a^n \mid n \in \mathbb{N}\}$

Correction L'idée est à peu près la même que pour la question précédente. On *pousse* un a de chaque côté du S à chaque étape de la dérivation et on finit avec un b au milieu (en lieu et place du ϵ).

On pose donc $G_1 = \big\langle \{a, b\}, \{S\}, S, \{S \to aSa \mid b \text{ (règles 1 et 2)}\}\big\rangle$

Toute dérivation dans cette grammaire sera de la forme

$$S \to aSa \to aaSaa \to aaaSaaa \to ...$$

Plus précisément :

$$S \to_1^n a^n S a^n \to_2 a^n b a^n$$

Question 2

Donnez une grammaire qui reconnaît le langage $L_2 = \{a^n b a^m \mid n, m \in \mathbb{N} \text{ et } n \geq m\}$

Correction On peut tout d'abord observer que $\{a^n b a^m \mid n, m \in \mathbb{N} \text{ et } n \geq m\} = \{a^n a^k b a^n \mid n, k \in \mathbb{N}\}$. Dit autrement, L_2 contient exactement les mots de L_1 avec des a qui se seraient *glissés* entre le b central et les a de gauche.

C'est ce qu'on va traduire en règles, en *prolongeant* la grammaire précédente : au lieu de produire uniquement un b, la deuxième règle de S laissera également derrière elle un nouveau non-terminal appelé A, qui peut générer un nombre arbitraire de a.

On pose donc $G_2 = \big\langle \{a, b\}, \{S, A\}, S, \{$

$S \to aSa \mid Ab$ (règles 1 et 2),

$A \to aA \mid \epsilon$ (règles 3 et 4)$\}\big\rangle$

Toute dérivation dans cette grammaire sera de la forme

$$S \to_1^n a^n S a^n \to_2 a^n A b a^n \to_3^k a^n a^k b A a^n \to_4 a^n a^k b \epsilon a^n = a^n a^k b a^n = a^{k+n} b a^n$$

Avec la remarque quelques lignes plus haut, on obtient bien le langage L_2

Remarque On aurait pu obtenir le même langage avec une grammaire à 1 non-terminal en faisait $S \to aSa \mid aS \mid \epsilon$. Cependant, l'introduction du A permet d'obtenir une grammaire non-ambiguë, ce que la grammaire de la phrase précédente n'est pas.

Question 3

Donnez une grammaire qui reconnaît le langage

$$L_3 = \{a^n b a^m b a^p b a^q \mid n, m, p, q \in \mathbb{N}, n \geq m \text{ et } p \geq q\}$$

Correction On commence par la remarque suivante : $L_3 = \{ubv \mid u, v \in L_2\}$. Dit autrement, tout mot de L_3 est composé de deux mots (indépendants) de L_2 séparés par un b. Or, on a déjà une grammaire pour générer les mots de L_2, il serait donc dommage de ne pas en profiter.

On pose donc $G_3 = \big\langle \{a, b\}, \{S', S, A\}, S', \{$

$S \to aSa \mid bA$ (règles 1 et 2),

$A \to aA \mid \epsilon$ (règles 3 et 4)

$S' \to SbS$ (règle 5)$\}\big\rangle$

On notera trois différences entre G_3 et G_2 :

L'introduction du nouveau non-terminal S'

Le changement d'axiome, qui est désormais S'. S est donc maintenant un non-terminal 'comme les autres'

L'introduction de la nouvelle règle 5, qui permet de *lancer* deux exécutions de de G_2, séparées par un b

Toute dérivation dans cette grammaire sera de la forme

$$S' \to_5 SbS \to^*_{1,2,3,4} a^n a^k b a^n b a^{n'} a^{k'} b a^{n'}$$

Remarque On commence à voir se dessiner la logique de l'exercice, où les grammaires (enfin, certaines) vont être écrites *au-dessus* d'une autre. Pour prouver que chaque grammaire G_i engendre bien le langage L_i, on peut s'appuyer sur le fait que les grammaires précédentes étaient correctes (les miracles de la récursivités). C'est pour ça qu'on se permet, juste au-dessus, d'affirmer que SbS se dérive, après un certain nombre d'applications des règles 1,2,3 et 4, en un mot de la forme $a^n a^k b a^n b a^{n'} a^{k'} b a^{n'}$.

Question 4

Donnez une grammaire qui reconnaît le langage

$$L_4 = \{a^n b a^m b a^p b a^q \mid n, m, p, q \in \mathbb{N}, n \geq q \text{ et } m \geq p\}$$

Correction Ici, on a des relations imbriquées entre les variables :

$$a^n b a^m b a^p b a^q$$

On va pouvoir retrouver cette structure *pyramidale* avec deux dérivations de G_2 *imbriquées*. On pose donc $G_4 = \big\langle \{a, b\}, \{S'', S''', A\}, S'', \{$

$S'' \to aS''a \mid AbS'''b$ (règles 1 et 2),

$A \to aA \mid \epsilon$ (règles 3 et 4),

$S''' \to aS'''a \mid Ab$ (règles 5 et 6)$\}\big\rangle$

Notez qu'on a échangé S contre S'' et S''', le premier étant l'axiome, mais qu'on a gardé A (vous devriez comprendre pourquoi à la question suivante). Dans l'idée, S'' sert à générer la partie *extérieure* du mot (le a^n et le a^q) et à *lancer* la dérivation partant de S''', qui va elle-même générer a^m et a^p. Plus formellement, toute dérivation sera de la forme

$$
\begin{aligned}
& S'' \\
\to_1^q\ & a^q S'' a^q \\
\to_2\ & a^q A b S''' b a^q \\
\to_3^k\ & a^q a^k A b S''' b a^q \\
\to_4\ & a^q a^k b S''' b a^q \\
\to_5^p\ & a^q a^k b a^p S''' a^p b a^q \\
\to_6\ & a^q a^k b a^p A b a^p b a^q \\
\to_3^r\ & a^q a^k b a^p a^r A b a^p b a^q \\
\to_4\ & a^q a^k b a^p a^r b a^p b a^q
\end{aligned}
$$

En posant $k = n - q$ et $r = m - p$, on obtient bien exactement les mots de L_4.

Question 5

Donnez une grammaire qui reconnaît le langage

$$L_5 = \{a^n b a^m b a^p b a^q \mid n, m, p, q \in \mathbb{N} \text{ et } ((n \geq m \text{ et } p \geq q) \text{ ou } (n \geq q \text{ et } m \geq p))\}$$

Correction On remarque tout d'abord que $L_5 = L_3 \cup L_4$. On peut donc engendrer l'ensemble des mots de L_5 en combinant les grammaires G_3 et G_4. Dans l'idée, on a simplement besoin d'ajouter un nouveau symbole, qui servira aussi d'axiome, et qui permettra de choisir quelle grammaire (G_3 ou G_4) sera *appelée*. Formellement, on a $G_5 = \langle \{a, b\}, \{S_f, S, S', S'', S''', A\}, S_f, \{$

$$
\begin{aligned}
& S_f \to S' \mid S'', \\
& S' \to SbS, \\
& S \to aSa \mid bA, \\
& A \to aA \mid \epsilon, \\
& S'' \to aS''a \mid AbS'''b, \\
& S''' \to aS'''a \mid Ab\}\rangle
\end{aligned}
$$

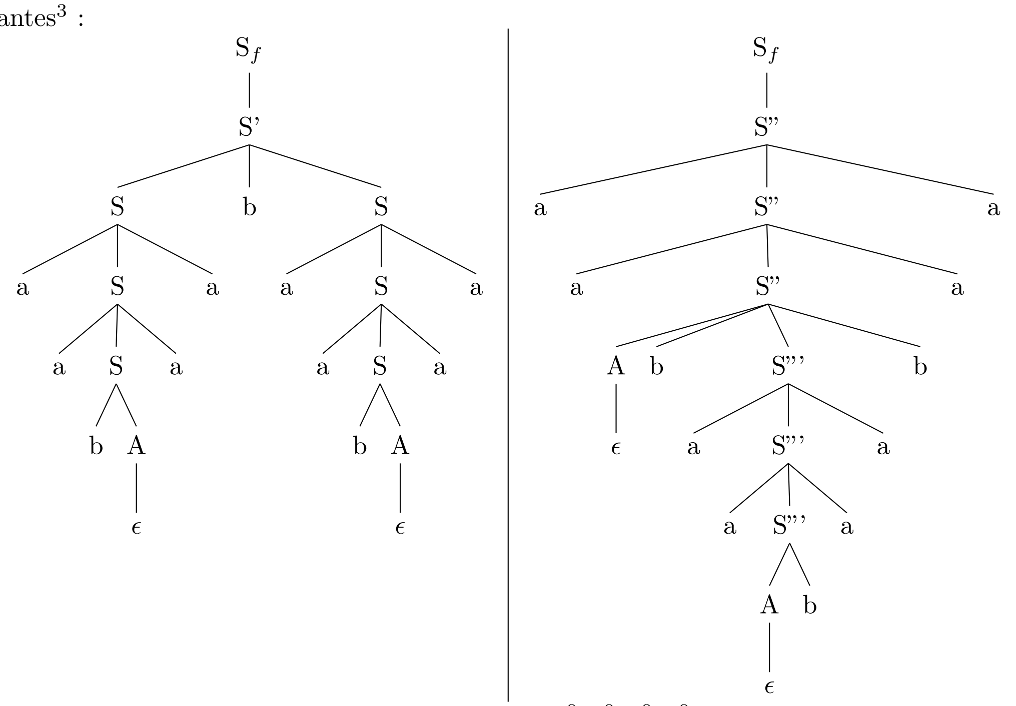
Dans les lignes 2, 3 et 4, on reconnaît les règles de G_4. Quant aux lignes 4, 5 et 6, on y retrouve les règles G_3 (notez que la règle de A est *partagée*, puisqu'elle apparaissait à l'identique dans les deux grammaires). Enfin, les deux règles de la première ligne, qui concernent l'axiome S_f, permettent de choisir si on veut effectuer une dérivation *à la* G_3 ou *à la* G_4. G_5 génère donc $L_3 \cup L_4 = L_5$.

Question 6

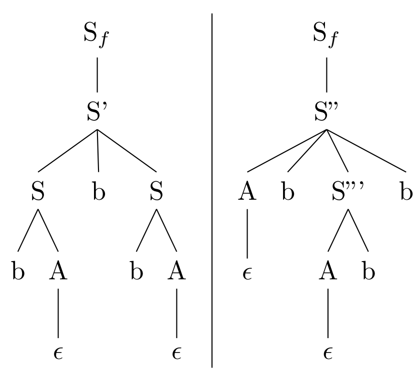
Donnez, dans la grammaire de la question 5, deux dérivations différentes d'un même mot de L_5 (pas de justification demandée)

Correction Soient $\phi(n,m,p,q) \equiv (n \geq m \wedge$[2] $p \geq q)$ et $\psi(n,m,p,q) \equiv (n \geq q \wedge m \geq p)$. Les grammaires G_3 et G_4 n'étant pas ambigües, on ne risque pas de trouver plusieurs dérivations pour un mot $a^n ba^m ba^p ba^q$ correspondant *strictement* à $\phi(n,m,p,q)$ ou $\psi(n,m,p,q)$ (cad rendant l'un vrai et l'autre faux). On veut donc un mot rendant les deux conditions vraies.

On prend par exemple $u = a^2ba^2ba^2ba^2$, qu'on peut obtenir par les deux dérivations suivantes[3] :



On pourrait même être plus radical et prendre $u = a^0ba^0ba^0ba^0 = bbb$



[2] $\wedge$ = ‘et‘

[3] Notez qu'on n'a pas besoin d'avoir le même nombre de a dans chaque segment. Le mot $a^4ba^3ba^2ba^1$ marche aussi par exemple

3 Grammaire mystère

Soit la grammaire de type 0[4] suivante : $\Big\langle \{a, b, \#\}, \{S, S', A, B, \$\}, S, \{$

1. $S \to \$_G S' \$_D$
2. $S' \to aAS'$
3. $S' \to bBS'$
4. $S' \to \epsilon$
5. $Aa \to aA$
6. $Ab \to bA$
7. $Ba \to aB$
8. $Bb \to bB$
9. $\$_G a \to a\$_G$
10. $\$_G b \to b\$_G$
11. $A\$_D \to \$_D a$
12. $B\$_D \to \$_D b$
13. $\$_G\$_D \to \#\} \Big\rangle$

Expliquez, en des termes très simples et *naturels*, le langage décrit par cette grammaire. Justifiez votre réponse en expliquant succinctement le fonctionnement de la grammaire (au moins les grandes étapes).

Indice Plutôt que d'essayer de deviner le langage engendré en fixant longuement les règles, dérivez[5] quelques mots au hasard et voyez s'ils n'ont pas l'air de partager une propriété intéressante. C'est beaucoup plus facile de vérifier qu'une grammaire a une propriété donnée que de l'inférer.

Correction Si on s'amuse à faire quelques dérivations, on se retrouve toujours avec des mots de la forme $u\#u$, comme par exemple $abba\#abba$:

$$
\begin{aligned}
& S \\
\to_1\ & \$_G S' \$_D \\
\to_2\ & \$_G aAS' \$_D \\
\to_3^2\ & \$_G aAbBbBS' \$_D \\
\to_2\ & \$_G aAbBbBaAS' \$_D \\
\to_4\ & \$_G aAbBbBaA\$_D
\end{aligned}
$$

[4]Le langage décrit est lui-même de type 1, ce qui veut dire qu'on pourrait le faire avec une grammaire contextuelle, mais c'est plus *simple* en s'accordant le luxe d'une type 0

[5]Notez que pour les grammaire de type 0 (et 1), la traduction des dérivations en arbre n'est plus possible. Vous devrez donc faire des dérivations 'plates', comme on faisait au début du chapitre sur les grammaires

$$\begin{aligned}
&\rightarrow_7 \; \$_G aAbBbaBA\$_D \\
&\rightarrow_8 \; \$_G aAbbBaBA\$_D \\
&\rightarrow_7 \; \$_G aAbbaBBA\$_D \\
&\rightarrow^3_{5+6} \; \$_G abbaABBA\$_D \\
&\rightarrow^4_{9+10} \; abba\$_G ABBA\$_D \\
&\rightarrow^4_{11+12} \; abba\$_G\$_D abba \\
&\rightarrow_{13} \; abba\#abba
\end{aligned}$$

Le langage engendré est en effet celui des mots ‘doublés‘ (avec un # qui sert de séparateur à la fin[6]). On peut justifier cette réponse en étudiant l'ensemble des dérivations possibles :

- Dans un premier temps, l'axiome S passe la main à un ‘second axiome‘ en encadrant le mot par des \$ qui serviront plus loin à repérer le milieu du mot engendré
- Ensuite, on utilise les règles 2 et 3 permettent de générer le mot qui va être doublé (le u de $u\#u$). Dans l'exemple ci-dessus, on a utilisé R2, R3, R3 puis R2 pour générer les terminaux abba (dans cet ordre-là). Les terminaux sont cependant pour l'instant accompagnés des non-terminaux correspondant.

  A supposer qu'on veuille générer le mot $u\#u$ avec $u = c_1c_2...c_n$, à ce point de la dérivation on a $\$_G c_1C_1c_2C_2...c_nC_nS'\$_D$
- La règle 4 fait ensuite disparaître le S'

  On a $\$_G c_1C_1c_2C_2...c_nC_n\$_D$
- Les règles 5 à 8 permettent ensuite aux non-terminaux de *remonter* le mot (en allant vers la droite). L'astuce ici est que les non-terminaux peuvent *enjamber* les terminaux, mais pas les autres non-terminaux. L'ordre entre ces derniers est donc préservé.

  On en est donc à $\$_G c_1c_2...c_nC_1C_2...C_n\$_D$
- Les règles 9 et 10 font quant à elles remonter le $\$_G$ jusqu'au début des non-terminaux

  Ce qui donne $c_1c_2...c_n\$_G C_1C_2...C_n\$_D$
- Dualement, les règles 11 et 12 font *descendre* le $\$_D$. Cependant, ce dernier transforme tous les non-terminaux en le terminal correspondant (ie. A en a et B en b)

  On se retrouve avec $c_1c_2...c_n\$_G\$_D c_1c_2...c_n$
- Enfin, la règle 13 transforme la paire nouvellement crée de \$ en #. On finit donc bien avec $c_1c_2...c_n\#c_1c_2...c_n \in \Sigma^*$

[6]Notez que c'est uniquement pour repérer plus facilement le milieu, on aurait tout à fait pu changer la règle 13 en \$\$ $\rightarrow \epsilon$ pour faire des ‘purs doublons‘