Théorie des Langages 1

Recueil d'exercices

1 Induction structurelle

Exercice 1 Soit V un vocabulaire et soit un sous-ensemble $A \subseteq V$. Dans cet exercice on considérera la fonction $|\,.\,|_A : V^* \to \mathbb{N}$ qui à tout mot $w \in V^*$ associe le nombre d'occurrences d'éléments de A présents dans w. Ainsi, si $V = \{a, b, c, d\}$ et $A = \{a, b\}$, alors :

— $|cabdbacc|_A = 4$,
— $|bbaba|_A = 5$,
— $|ccdc|_A = 0$.

On pose $V = \{\wedge, \vee, \neg,), (, \bot, \top\}$, et on considère l'ensemble E des formules logiques défini par induction structurelle de la façon suivante :

Base.

1. $\top \in E$
2. $\bot \in E$

Induction. Si w, w_1 et w_2 sont dans E, alors :

3. $(\neg\, w) \in E$,
4. $(w_1 \wedge w_2) \in E$,
5. $(w_1 \vee w_2) \in E$.

Noter que les parenthèses font ici partie du vocabulaire.

▷ Question 1 Est-ce que les deux mots suivants sont dans E ? On justifiera en quelques mots les réponses.

1. $(\top \wedge \neg)$
2. $(\top \wedge \top \wedge \top)$

▷ Question 2 Montrer que $((\top \wedge \top) \vee (\bot \vee \top)) \in E$.

On définit les ensembles de symboles suivants :

$$U = \{\neg\}, \qquad B = \{\wedge, \vee\},$$
$$S = \{\top, \bot\}, \qquad N = U \cup B \cup S.$$

▷ Question 3 Soit $w_1 = (\top \wedge ((\neg\bot) \vee (\top \wedge (\bot \vee (\neg\top)))))$. Calculer $|w_1|_U$, $|w_1|_B$ et $|w_1|_S$.

▷ Question 4 Soit P la propriété sur E définie[1] par $P[w] \stackrel{\text{def}}{=} |w|_S = |w|_B + 1$. Démontrer que tout $w \in E$ vérifie $P[w]$ par induction structurelle sur w.

▷ Question 5 **[Avancé]** Soit $w \in E$. Exprimer $|w|$ en fonction de $|w|_B$ et $|w|_U$.

▷ Question 6 **[Avancé]** Utiliser la question 5 pour justifier formellement les réponses à la question 1.

Solution de l'Exercice 1.

▷ Question 1 Le symbole ¬ ne peut apparaître qu'après « (» et il manque des parenthèses pour le deuxième mot (pour chaque ∧ il doit y avoir une parenthèse ouvrante).

1. J'utilise les crochets [_] car les parenthèses font partie du vocabulaire V.

▷ Question 2 Pour montrer qu'un terme appartient à un ensemble défini par induction, il faut le construire en utilisant les constructeurs et les cas de base. Notons $u = (\top \wedge \top)$ et $v = (\bot \vee \top)$. On a $u \in E$ par application du deuxième constructeur inductif à deux fois le premier cas de base. De même, $v \in E$ par application du troisième constructeur inductif au deuxième et premier cas de base. Au final, le mot $((\top \wedge \top) \vee (\bot \vee \top))$ appartient à E par application du troisième constructeur inductif à u et v.

▷ Question 3 On a $|w_1|_U = 2$, $|w_1|_B = 4$ et $|w_1|_S = 5$.

▷ Question 4 **Rappel :** Pour faire une démonstration par induction structurelle d'une propriété P sur un ensemble E défini inductivement, on doit démontrer P pour les cas de base et démontrer que pour chaque constructeur inductif κ, si on suppose P pour tous les arguments $u_1, \ldots, u_k$ de κ (d'arité k), alors on peut démontrer P sur le mot $\kappa(u_1, \ldots, u_k)$.

Ici, on a deux cas de base et trois constructeurs inductifs, donc cinq cas à considérer.

Montrons la propriété $P[w] = |w|_S = |w|_B + 1$ par induction structurelle sur E :

— Pour $\top$: $|\top|_S = 1 = 0 + 1 = |\top|_B + 1$.

— Pour $\bot$: $|\bot|_S = 1 = 0 + 1 = |\bot|_B + 1$.

— Pour $(\neg u)$: Soit u dans E et supposons $P[u]$. On a

$$\begin{aligned} |(\neg u)|_S &= |u|_S \\ &\overset{HI}{=\!=} |u|_B + 1 \\ &= |(\neg u)|_B + 1 \end{aligned}$$

Ainsi, on a montré $P[(\neg u)]$.

— Pour $(u \wedge v)$: Soient u, v dans E et supposons $P[u]$ et $P[v]$. On a

$$\begin{aligned} |(u \wedge v)|_S &= |u|_S + |v|_S \\ &\overset{HI}{=\!=} (|u|_B + 1) + (|v|_B + 1) \\ &= (1 + |u|_B + |v|_B) + 1 \\ &= |(u \wedge v)|_B + 1 \end{aligned}$$

Ainsi, on a montré $P[(u \wedge v)]$.

— Pour $(u \vee v)$: Soient u, v dans E et supposons $P[u]$ et $P[v]$. On a

$$\begin{aligned} |(u \vee v)|_S &= |u|_S + |v|_S \\ &\overset{HI}{=\!=} (|u|_B + 1) + (|v|_B + 1) \\ &= (|u|_B + |v|_B + 1) + 1 \\ &= |(u \vee v)|_B + 1 \end{aligned}$$

Ainsi, on a montré $P[(u \vee v)]$.

▷ Question 5 On a $|w| = 4|w|_B + 3|w|_U + 1$. En effet, $|w| = |w|_N + |w|_{\{),(\}} = (2|w|_B + |w|_U + 1) + (2|w|_B + 2|w|_U)$.

▷ Question 6 L'égalité de la question précédente n'est pas vérifiée pour les mots de la question 1, ils ne font donc pas partie de E.

Exercice 2 Soit V un vocabulaire quelconque. Définir la fonction $|\,.\,|_A : V^* \to \mathbb{N}$ de l'exercice 1 par induction structurelle sur V^*.

Solution de l'Exercice 2.

Base : $|\varepsilon|_A = 0$.

Induction : Pour tout mot w et tout symbole x,

$$|xw|_A = \begin{cases} 1 + |w|_A & \text{si } x \in A \\ |w|_A & \text{sinon} \end{cases}$$

Exercice 3 Définitions inductives d'ensembles.

▷ Question 1 Donner des définitions inductives des langages suivants :

1. L'ensemble L_1 des mots sur $\{a, b\}$ de longueur paire.
2. L'ensemble L_2 des mots sur $\{a, b\}$ ne contenant pas deux a consécutifs.
3. L'ensemble L_3 des palindromes sur $\{a, b\}$.
4. **[Avancé]** L'ensemble L_4 des mots sur $\{a, b\}$ contenant un nombre pair de a.
5. **[Avancé]** L'ensemble L_5 des mots sur $\{a, b\}$ contenant autant de a que de b.

▷ Question 2 **[Avancé]** Prouver que ces définitions inductives sont correctes.

Solution de l'Exercice 3.

▷ Question 1

1. Soit M_1 l'ensemble défini par induction de la façon suivante :

   Base : $\varepsilon \in M_1$.

   Induction : si $w \in M_1$, alors aaw, abw, baw et bbw sont également dans M_1.

   Remarque : Pour les cas inductifs, on aurait pu choisir de placer les deux lettres derrière w (waa, wab, wba, wbb) voire une de chaque côté (awa, awb, bwa, bwb).
2. Soit M_2 l'ensemble défini par induction de la façon suivante :

   Base : $\{\varepsilon, a\} \subseteq M_2$.

   Induction : si $w \in M_2$, alors bw et abw sont également dans M_2.

   Remarque : Idem, on aurait pu prendre wb, wba pour les cas inductifs.
3. Soit M_3 l'ensemble défini par induction de la façon suivante :

   Base : $\{\varepsilon, a, b\} \in M_3$.

   Induction : si $w \in M_3$, alors awa et bwb sont également dans M_3.
4. Soit M_4 l'ensemble défini par induction de la façon suivante :

   Base : $\varepsilon \in M_4$.

   Induction : si $w \in M_4$, alors bw, wb et awa sont également dans M_4.
5. Soit M_5 l'ensemble défini par induction de la façon suivante :

   Base : $\varepsilon \in M_5$.

   Induction : si w_1 et w_2 sont des éléments de M_5, alors aw_1b, bw_1a et w_1w_2 sont également dans M_5.
   Variante (celle présentée en cours) : si w_1 et w_2 sont de éléments de M_5, alors aw_1bw_2 et bw_1aw_2 sont dans M_5

▷ Question 2 Les preuves que les M_i sont inclus dans les L_i se font par induction structurelle. Seule la preuve pour $i = 1$ est détaillée ici, les autres étant très similaires. Il s'agit donc de prouver que $M_1 \subseteq L_1$ par induction structurelle.

— On montre tout d'abord que la propriété est vérifiée pour tous les cas de base. On a $|\varepsilon| = 0$, donc $\varepsilon \in L_1$.

— On montre ensuite que la propriété est préservée par les constructeurs inductifs. Soit $w \in M_1$, et supposons que $w \in L_1$. Alors $|aaw| = |w| + 2$ est pair, donc aaw est bien élément de L_1. On prouve de la même manière que abw, baw et bbw sont dans L_1.

Les preuves que les définitions inductives engendrent bien tous les mots des langages correspondants se feront par récurrence bien fondée sur la longueur des mots.

1. Prouvons que $L_1 \subseteq M_1$.
   Soit $w \in L_1$ de longueur n, et supposons que pour tout $w' \in L_1$, si $|w'| < n$, alors $w' \in M_1$. Si $n = 0$, alors $w = \varepsilon$ et il est clair que $w \in M_1$. Il est également clair que si $w \in L_1$, alors on ne peut pas avoir $n = 1$. On suppose maintenant que $w = uvw'$, où u, v sont des lettres dans $\{a, b\}$. Sans perte de généralité, on peut supposer que $u = a$ et $v = b$, les autres cas sont similaires. Comme $w' \in L_1$ est nécessairement de longueur paire et que $|w'| = |w| - 2 < |w|$, par hypothèse d'induction, $w' \in M_1$. Donc, le mot abw' est également dans M_1 d'après les règles de construction de M_1, ce qui prouve que $w \in M_1$.
   On en déduit que $L_1 \subseteq M_1$.
2. Prouvons que $L_2 \subseteq M_2$.
   Soit $w \in L_2$ de longueur n, et supposons que pour tout $w' \in L_2$, si $|w'| < n$, alors $w' \in M_2$.
   Si $n \in \{0, 1\}$, alors $w \in \{\varepsilon, a, b\}$, et il est aisé de vérifier que $w \in M_2$. Supposons maintenant $n > 1$ et distinguons deux cas.
   — Si $w = bw'$, alors w' ne peut pas contenir deux a consécutifs et est donc élément de L_2. Comme $|w'| = n - 1$, on en déduit que $w' \in M_2$, puis que $bw' \in M_2$.
   — Sinon $w = aw'$, et nécessairement $w' = bw''$ car $n > 1$ par hypothèse. Donc $w = abw''$ et w'' ne peut pas contenir deux a consécutifs. Donc $w'' \in M_2$, et $abw'' \in M_2$.
3. Prouvons que $L_3 \subseteq M_3$.
   Soit $w \in L_3$ de longueur n, et supposons que pour tout $w' \in L_3$, si $|w'| < n$, alors $w' \in M_3$.
   Si $n \in \{0, 1\}$, alors il est aisé de vérifier que $w \in M_3$.
   Supposons maintenant que la première lettre de w soit un a, le cas où cette première lettre est un b est similaire. Alors comme w est un palindrôme, sa dernière lettre est également un a, donc w est de la forme $aw'a$. Comme w' est nécessairement un palindrôme, de longueur strictement inférieure à n, on a $w' \in M_3$, et donc, $aw'a \in M_3$.
4. Prouvons que $L_4 \subseteq M_4$.
   Soit $w \in L_4$ de longueur n, et supposons que pour tout $w' \in L_4$, si $|w'| < n$, alors $w' \in M_4$.
   Il est clair que si $w = \varepsilon$, alors $w \in M_4$.

— Si $w = bw'$, alors w' doit contenir un nombre pair de a, donc par hypothèse d'induction $w' \in M_4$ et on en déduit que $bw' \in M_4$.
— Si $w = w'b$, alors le même raisonnement que précédemment prouve que $w'b \in M_4$.
— Supposons maintenant que $w = aw'a$. Alors une fois encore, w' doit contenir un nombre pair de a et est bien élément de M_4 ; on en déduit que $aw'a \in M_4$.

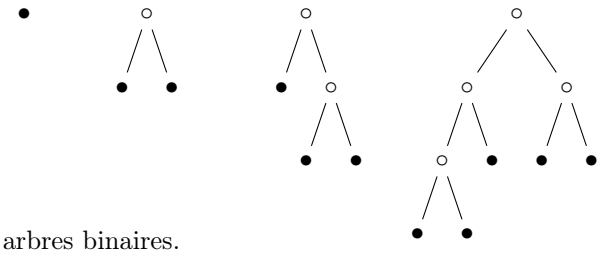
5. Prouvons que $L_5 \subseteq M_5$.
Soit $w \in L_5$ de longueur n, et supposons que pour tout $w' \in L_5$, si $|w'| < n$, alors $w' \in M_5$.
Pour la première version : il est clair que si $w = \varepsilon$, alors $w \in M_5$. Supposons maintenant que la première lettre de w est a, la preuve dans le cas où la première lettre est b est similaire.
— Si w est de la forme aw_1b, alors nécessairement, w_1 contient autant de a que de b. Ce mot étant de longueur $n-2$, on en déduit que $w_1 \in M_5$, et qu'on a bien $aw_1b \in M_5$.
— Supposons que w est de la forme aw_1a, et considérons l'ensemble

$$A \stackrel{\text{def}}{=} \{w' \mid w' \text{ est un préfixe strict de } w \text{ et } |w'|_a \leq |w'_b|\}.$$

L'ensemble A est non-vide car aw_1 en est un élément. Prenons l'élément de A de longueur minimale ; cet élément est de la forme aw_2, où $w_2 \in \{a,b\}^*$, et w est de la forme aw_2w_3a. Comme aw_2 est de longueur minimale dans A, on en déduit que $|aw_2|_a = |aw_2|_b$ (sinon en enlevant sa dernière lettre, on obtient toujours un élément de A). Nécessairement, on a aussi $|w_3a|_a = |w_3a|_b$, puisque $|w|_a = |w|_b$. Ceci signifie que aw_2 et w_3a sont tous deux éléments de L_5, et comme $|aw_2| < |w|$ et $|w_3a| < |w|$, ces mots sont également éléments de M_5. Par la suite, aw_2w_3a est nécessairement dans M_5 d'après la dernière règle de construction de M_5.
Pour la variante (preuve non présentée en cours) : si $|w| = 0$, alors $w = \varepsilon \in M_5$. Pour $|w| > 0$, supposons $w = aw'$ (le cas $w = bw'$ est identique, en remplaçant tous les a par b). Soit x le plus court des préfixes y de w' tels que $|y|_b > |y|_a$ (forcément $x \neq \varepsilon$). Noter que x existe car $|w'|_b > |w'|_a$. On a donc $w = axw_2$. x ne peut pas terminer par a, sinon $x = za$ et $|z|_b > |z|_a$ avec z plus court que x. Donc x termine par b : $x = w_1b$. Maintenant, $|x|_b = |x|_a + 1$ car sinon $|w_1|_b > |w_1|_a$, avec w_1 plus court que x. On en déduit $|w_1|_b = |w_1|_a$, donc $w_1 \in L_5$, et donc aussi $w_2 \in L_5$. On a donc $w = aw_1bw_2$. L'HI nous dit que w_1 et w_2 sont dans M_5, donc par induction w aussi.

Exercice 4

On s'intéresse aux arbres binaires, mais pour simplifier, on considère qu'ils ne contiennent pas de données. On rappelle qu'un arbre binaire est un arbre dont tous les noeuds internes ont exactement deux fils.
En voici ci-contre quatre exemples représentés graphiquement :



▷ Question 1 Donner le vocabulaire et la définition inductive des arbres binaires.

▷ Question 2 La formule d'énumération d'un ensemble inductif formé à partir d'un ensemble de cas de base B et d'un ensemble de constructeurs inductifs K est :

$$\begin{aligned} E_0 &\stackrel{\text{def}}{=} B, \\ E_{n+1} &\stackrel{\text{def}}{=} E_n \cup \{\kappa_i(e_1, \ldots, e_{k_i}) \mid \kappa_i \in K, e_1, \ldots, e_{k_i} \in E_n\} \end{aligned}$$

Donner les trois premiers ensembles E_n pour la définition des arbres binaires de la question précédente. Quelle est la hauteur des arbres contenus dans ces ensembles ? On rappelle que la hauteur d'un arbre est la longueur maximale d'un chemin de la racine (le nœud tout en haut de l'arbre) vers une feuille de l'arbre.

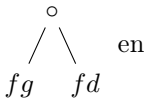
▷ Question 3 Montrer par induction structurelle que dans un arbre binaire le nombre n_i de nœuds internes $\circ$ et le nombre n_f de feuilles (nœuds sans fils) $\bullet$ satisfont la relation suivante : $n_f = n_i + 1$.

▷ Question 4 **[Avancé]** Soit H_n l'ensemble des arbres binaires de hauteur inférieure ou égale à n. Montrer l'égalité $E_n = H_n$ par récurrence sur n.

Solution de l'Exercice 4.

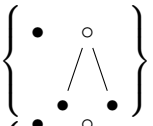
▷ Question 1 On fixe le vocabulaire à $V = \{\bullet, \circ, \backslash, /\}$.

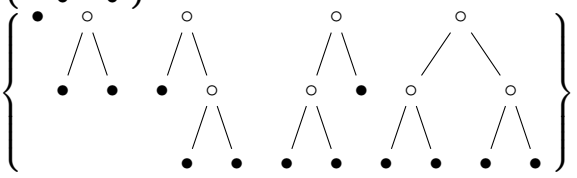
Base une feuille, notée ici $\bullet$

Induction Si fg et fd sont des arbres binaires,  en est un.

▷ QUESTION 2

$E_0 = \{ \bullet \}$

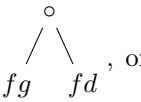
$E_1 =$ 

$E_2 =$ 

Les arbres de E_n ont une hauteur inférieure ou égale à n.

▷ QUESTION 3

Base : $n_f = 1$ et $n_i = 0$ donc on a bien $n_f = n_i + 1$.

Induction : Soit fg et fd des arbres binaires satisfaisant la relation $n_f = n_i + 1$. Pour l'arbre  , on a $n_f = n_f(fg) + n_f(fd)$ et $n_i = n_i(fg) + n_i(fd) + 1$. Par Hyp. Ind. sur fg et fd, on en déduit $n_f = n_f(fg) + n_f(fd) = (n_i(fg) + 1) + (n_i(fd) + 1) = (n_i(fg) + n_i(fd) + 1) + 1 = n_i + 1$.

▷ QUESTION 4 $n = 0$ On a $E_0 = \{\bullet\}$. Comme la hauteur de $\bullet$ est 0, on a $E_0 \subseteq H_0$. Réciproquement, $\bullet$ est le seul arbre de hauteur 0 donc on a bien $E_0 = H_0$.

$n \mapsto n + 1$ Soit $n \in \mathbb{N}$. Supposons $E_n = H_n$ et montrons l'égalité $E_{n+1} = H_{n+1}$.

$\subseteq$ Soit $t \in E_{n+1}$. Par définition de E_{n+1}, ou bien $t \in E_n$, auquel cas par hypothèse de récurrence, $t \in H_n \subset H_{n+1}$; ou bien t contient une racine dont les deux fils fg et fd sont dans $E_n = H_n$. La hauteur de t vaut alors $1 + \max(\mathsf{hauteur}(fg), \mathsf{hauteur}(fd))$. Par hypothèse de récurrence, $\mathsf{hauteur}(fg) \leq n$ et $\mathsf{hauteur}(fg) \leq n$ d'où le résultat.

$\supseteq$ Soit $t \in H_{n+1}$. Si $\mathsf{hauteur}(t) \leq n$, par hypothèse de récurrence, $t \in E_n \subset E_{n+1}$. Sinon, $\mathsf{hauteur}(t) = n + 1$ et la racine de t n'est pas une feuille. Ses fils gauche et droit sont de hauteurs inférieures ou égales à n donc par hypothèse de récurrence, sont dans E_n. Ainsi, $t = \kappa(fg, fd)$ avec $fg, fd \in E_n$ donc $t \in E_{n+1}$.

Exercice 5 [Avancé] On considère un langage E_p d'expressions préfixées *sur des chiffres*. Pour cela, on définit les ensembles suivants :

$$\begin{aligned} \text{NUM} &= \{\texttt{Zr}, \texttt{Un}, \texttt{De}, \texttt{Tr}, \texttt{Qu}, \texttt{Ci}, \texttt{Si}, \texttt{Sp}, \texttt{Hu}, \texttt{Ne}\}, \\ \text{OP} &= \{+, -, \times\}. \end{aligned}$$

L'ensemble E_p est alors un langage sur le vocabulaire $V = \text{NUM} \cup \text{OP}$, dont les éléments sont des expressions arithmétiques sur des chiffres, où les opérateurs sont placés avant les opérandes.

Par exemple, l'expression préfixée correspondant à $1 + 2$ est `+ Un De` ; l'expression préfixée correspondant à $(2 + 3) \times (3 - 1)$ est `× + De Tr − Tr Un`.

▷ QUESTION 1 Donner une définition inductive du langage E_p.

On considère la fonction **eval** : $E_\text{p} \to \mathbb{N}$, qui calcule la valeur d'une expression préfixée. Par exemple, si $w =$ `× + De Tr − Tr Un` alors $\textbf{eval}(w) = 10$.

▷ QUESTION 2 Donner une définition inductive de la fonction **eval**.

Pour un mot $w \in E_\text{p}$ donné, on note $|w|_\text{NUM}$ le nombre d'éléments de NUM présents dans w, et on note $|w|_\text{OP}$ le nombre d'éléments de OP présents dans w.

▷ QUESTION 3 Prouver par induction structurelle que pour tout $w \in E_\text{p}$, on a $|w|_\text{NUM} = |w|_\text{OP} + 1$.

Solution de l'Exercice 5.

▷ Question 1

Base : $\textsc{Num} \subseteq E_{\text{p}}$.
Induction : si $e_1, e_2 \in E_p$, alors $\{+\ e_1\ e_2,\ -\ e_1\ e_2,\ \times\ e_1\ e_2\} \subseteq E_{\text{p}}$.

▷ Question 2

Base : $\mathbf{eval}(\texttt{Zr}) \stackrel{\text{def}}{=} 0$, ..., $\mathbf{eval}(\texttt{Ne}) \stackrel{\text{def}}{=} 9$.
Induction : si $\circ \in \{+, -, \times\}$, alors $\mathbf{eval}(\circ\ e_1\ e_2) \stackrel{\text{def}}{=} \mathbf{eval}(e_1) \circ \mathbf{eval}(e_2)$.

On notera qu'on utilise ici des notations identiques pour les opérateurs $+, -, \times$ et pour les opérations mathématiques associées (i.e., pour le langage et le méta-langage).

▷ Question 3

Base : Pour tout $w \in \textsc{Num}$, on a $|w|_{\textsc{Num}} = 1$ et $|w|_{\textsc{Op}} = 0$.
Induction : Soit $\circ \in \{+, -, \times\}$. On a les égalités suivantes :

$$\begin{aligned} |\circ\ e_1\ e_2|_{\textsc{Num}} &= |e_1|_{\textsc{Num}} + |e_2|_{\textsc{Num}} \\ &= 2 + |e_1|_{\textsc{Op}} + |e_2|_{\textsc{Op}} \text{ (hyp. d'induction)} \\ &= 1 + |\circ\ e_1\ e_2|_{\textsc{Op}} \end{aligned}$$

2 Langages

Exercice 6 [A savoir faire] Soit L un langage. Montrer que les propositions suivantes sont équivalentes :

(i) $\varepsilon \in L$
(ii) $\forall i \geq 0$, $\varepsilon \in L^i$
(iii) $\forall i \geq 0$, $L^i \subseteq L^{i+1}$

Solution de l'Exercice 6.

(i) $\Rightarrow$ (ii) Supposons que $\varepsilon \in L$. On prouve par récurrence sur i qu'on a alors pour tout $i \geq 0$, $\varepsilon \in L^i$.
Base $L^0 = \{\varepsilon\}$ donc $\varepsilon \in L^0$
Induction Soit un entier i et supposons que $\varepsilon \in L^i$. Montrons $\varepsilon \in L^{i+1}$.
Comme $\varepsilon \in L$, on a $\varepsilon = \varepsilon.\varepsilon \in L.L^i = L^{i+1}$.
(ii) $\Rightarrow$ (iii) Supposons que $\forall i \geq 0$, $\varepsilon \in L^i$. Alors en particulier, $\varepsilon \in L$. Soit $n \geq 0$, montrons que $L^n \subseteq L^{n+1}$. Par définition, $L^{n+1} = L.L^n$, et comme $\varepsilon \in L$, on a $L^n = \{\varepsilon\}.L^n \subseteq L^{n+1}$.
(iii) $\Rightarrow$ (i) Supposons que $\forall i \geq 0$, $L^i \subseteq L^{i+1}$. Alors en particulier, $L^0 \subseteq L^1$, i.e., $\varepsilon \in L$.

Exercice 7 On considère dans cet exercice des langages définis sur $V = \{a, b\}$.

▷ Question 1 Définir **par concaténation et itération** le langage L_1 des mots constitués d'une séquence de a suivie d'une séquence de b. Les séquences peuvent éventuellement être vides.

▷ Question 2 Définir **par induction** le langage L_2 des mots de la forme $a^n b^n$ avec $n > 0$.

▷ Question 3 Définir **à l'aide des opérations ensemblistes classiques** et des langages précédents le langage L_3 des mots de la forme $a^i b^j$ avec $i \neq j$.

Solution de l'Exercice 7.

▷ Question 1 $L_1 = \{a\}^*\{b\}^*$.

▷ Question 2
Base : $ab \in L_2$.
Induction : Si $w \in L_2$ alors $awb \in L_2$.

▷ Question 3 $L_3 = L_1 \setminus (\{\varepsilon\} \cup L_2)$.

Exercice 8 Soit V un vocabulaire. Etant donnés deux mots $x, z \in V^*$, on dit que x est conjugué à z s'il existe un mot $y \in V^*$ tel que $xy = yz$. On souhaite prouver que x est conjugué à z si et seulement s'il existe deux mots $u, v \in V^*$ tels que $x = uv$ et $z = vu$.

▷ Question 1 Montrer que s'il existe deux mots $u, v \in V^*$ tels que $x = uv$ et $z = vu$ alors x est conjugué à z.

▷ Question 2 On suppose que x est conjugué à z et que $x = \varepsilon$. Déterminer u et v tels que $x = uv$ et $z = vu$.

On suppose maintenant que x est conjugué à z et que $x \neq \varepsilon$. Soit alors y **de longueur minimale** tel que $xy = yz$.

▷ Question 3 Montrer qu'on a nécessairement $|x| \geq |y|$.

▷ Question 4 En déduire l'existence de u et v tels que $x = uv$ et $z = vu$.

▷ Question 5 **[Avancé]** Montrer que la relation de conjuguaison est une relation d'équivalence.

Solution de l'Exercice 8.

▷ Question 1 Posons $y \stackrel{\text{def}}{=} u$. Alors $xy = (uv)u = u(vu) = uz = yz$. Le mot x est bien conjugué à z.

▷ Question 2 Si $x = \varepsilon$, alors $xy = y = yz$, donc nécessairement, $z = \varepsilon$. Il suffit de prendre $u = v = \varepsilon$, et on a $x = uv$ et $z = vu$.

▷ Question 3 Supposons que $|x| < |y|$. Comme $xy = yz$, x doit nécessairement être un préfixe strict de y ; il existe donc un mot $u \in V^+$ tel que $y = xu$. On a donc $xxu = xuz$, d'où $xu = uz$. Comme $|u| < |y|$, on a une contradiction : y ne peut pas être de longueur minimale. On en déduit que nécessairement, $|x| \geq |y|$.

▷ Question 4 Supposons que $|x| = |y|$. Comme $xy = yz$, on en déduit que $x = y$ et $y = z$; donc $x = z$. On peut alors prendre $u \stackrel{\text{def}}{=} x$ et $v \stackrel{\text{def}}{=} \varepsilon$.
Si $|x| > |y|$, alors comme $xy = yz$, on en déduit qu'il existe $w \in V^+$ tel que $x = yw$. On a donc $ywy = yz$, d'où $wy = z$. On peut donc prendre $u \stackrel{\text{def}}{=} y$ et $v \stackrel{\text{def}}{=} w$.

▷ Question 5 La relation de conjugaison est :

Réflexive : il suffit de prendre $y \stackrel{\text{def}}{=} \varepsilon$.

Symétrique : d'après ce qui précède, si x est conjugué à z, alors il existe u, v tels que $x = uv$ et $z = vu$. En intervertissant u et v, on en déduit que z est conjugé à x.

Transitive : supposons que x est conjugué à z et que z est conjugué à w. Par définition, il existe donc y tel que $xy = yz$, et il existe y' tel que $zy' = y'w$. On en déduit que $xyy' = yzy' = yy'w$, ce qui prouve que x est conjugué à w.

Exercice 9 [A savoir faire] Etant donné un ensemble E, on rappelle qu'une relation ρ sur E est un sous-ensemble de $E \times E$. La relation ρ est :

- Réflexive si pour tout $x \in E$, on a $(x, x) \in \rho$.
- Symétrique si pour tout $x, y \in E$, on a $(x, y) \in \rho$ si et seulement si $(y, x) \in \rho$.
- Antisymétrique si pour tout $x, y \in E$, si $(x, y) \in \rho$ et $(y, x) \in \rho$, alors $x = y$.
- Transitive si pour tout $x, y, z \in E$, si $(x, y) \in \rho$ et $(y, z) \in \rho$, alors $(x, z) \in \rho$.

Soit $E = \{a, b, c, d, e\}$, et considérons la relation ρ définie par :

$$\rho \ = \ \{(a,a),\ (a,b),\ (a,c),\ (b,d),\ (d,e)\}.$$

Construire les relations suivantes :

1. La fermeture réflexive de ρ (i.e. la plus petite relation réflexive contenant ρ).
2. La fermeture symétrique de ρ (i.e. la plus petite relation symétrique contenant ρ).
3. La fermeture transitive de ρ (i.e. la plus petite relation transitive contenant ρ).

Solution de l'Exercice 9.

1. $\rho \cup \{(b,b),\ (c,c),\ (d,d),\ (e,e)\}$.
2. $\rho \cup \{(b,a),\ (c,a),\ (d,b),\ (e,d)\}$.
3. $\rho \cup \{(a,d),\ (a,e),\ (b,e)\}$.

Exercice 10 Soient L et M des langages sur un vocabulaire V. Montrer que si $L \subseteq M$ alors $L^* \subseteq M^*$. La réciproque est-elle vraie ? Justifier.

Solution de l'Exercice 10. Soit $w \in L^*$. Par définition de L^*, w est de la forme $w_1 \cdots w_n$, avec pour tout $i \in \{1, \ldots, n\}$, $w_i \in L$. Comme $L \subseteq M$, chaque w_i est dans M, donc $w \in M^*$. D'où $L^* \subseteq M^*$.
L'inclusion réciproque est fausse : si $L = \{aa\}$ et $M = \{a\}$, alors $L^* \subseteq M^*$, mais $L \not\subseteq M$.

Exercice 11 Pour chacune des égalités suivantes identifier celles qui sont vraies et celles qui sont fausses. Pour la première catégorie on donnera une intuition de la preuve. Pour la seconde catégorie on donnera des contre-exemples et on identifiera les inclusions uni-directionnelles.

1. $L^* = (L^*)^*$
2. $L^* \cup M^* = (L \cup M)^*$
3. $(L^* \cup M^*)^* = (L \cup M)^*$
4. $L^+ = L^* \setminus \{\varepsilon\}$
5. $(LM)^* = L^*M^*$
6. $(L \cup M)^* = (L^*M^*)^*$

Solution de l'Exercice 11. Note : Les preuves de cet exercice sont nettement plus lisibles en utilisant la propriété $L \subseteq M \implies L^* \subseteq M^*$ qui est démontrée à l'exercice 10. C'est ce qu'on fait ici.

1. L'égalité $L^* = (L^*)^*$ est vraie.
   L'inclusion $L^* \subseteq (L^*)^*$ est facile à démontrer car pour tout langage M, on a $M \subseteq M^*$, soit en prenant une suite de longueur 1 de mots de M, soit en utilisant la définition de $M^* = \bigcup_{i \geqslant 0} M^i$.
   Réciproquement, considérons un mot de $(L^*)^*$. Ce mot est de la forme $w_1 \cdots w_n$, où pour tout $i \in \{1, \ldots, n\}$, $w_i \in L^*$. Par définition, chaque w_i est de la forme $x_{i,1} \cdots x_{i,n_i}$, où pour tout $j = 1 \ldots, n_i$, $x_{i,j} \in L$. On en déduit que w est une concaténation finie d'éléments de L, les $x_{i,j}$, et est donc dans L^*.
2. Comme $L \subseteq (L \cup M)$, on a $L^* \subseteq (L \cup M)^*$, et de même, $M^* \subseteq (L \cup M)^*$. Donc $L^* \cup M^* \subseteq (L \cup M)^*$.
   L'inclusion réciproque est fausse : si $L = \{a\}$ et $M = \{b\}$, alors $ab \in (L \cup M)^*$, mais $ab \notin L^* \cup M^*$.
3. L'égalité $(L \cup M)^* = (L^* \cup M^*)^*$ est vraie.
   On a $(L \cup M)^* \subseteq (L^* \cup M^*)^*$. En effet, comme $L \subseteq L^*$ et $M \subseteq M^*$, on a $(L \cup M) \subseteq (L^* \cup M^*)$, d'où $(L \cup M)^* \subseteq (L^* \cup M^*)^*$.
   Réciproquement, comme $L^* \subseteq (L \cup M)^*$ et $M^* \subseteq (L \cup M)^*$, on en déduit que $L^* \cup M^* \subseteq (L \cup M)^*$ puis que $(L^* \cup M^*)^* \subseteq ((L \cup M)^*)^* = (L \cup M)^*$ (question 1).
4. On a $L^* \setminus \{\varepsilon\} \subseteq L^+$ car $L^* = \bigcup_{i \geqslant 0} L^i = L^0 \cup \bigcup_{i>0} L^i = \{\varepsilon\} \cup L^+$.
   L'inclusion réciproque est fausse : par exemple si $L = \{\varepsilon\}$, alors $L^+ = \{\varepsilon\}$ mais $L^* \setminus \{\varepsilon\} = \emptyset$. Plus généralement, elle est fausse ssi $\varepsilon \in L$.
5. Aucune des deux inclusions n'est vraie.
   Posons $L = \{a\}$ et $M = \{b\}$. Alors $abab \in (LM)^*$, mais ce mot n'est pas dans L^*M^*. Réciproquement, $aa \in L^*M^*$ mais $aa \notin (LM)^*$.
6. L'égalité $(L \cup M)^* = (L^*M^*)^*$ est vraie.
   Comme $L \subseteq L^*M^*$ et $M \subseteq L^*M^*$, on a $(L \cup M) \subseteq L^*M^*$ et donc $(L \cup M)^* \subseteq (L^*M^*)^*$.
   Réciproquement, comme $L^* \subseteq (L \cup M)^*$ et $M^* \subseteq (L \cup M)^*$, on a $L^*M^* \subseteq ((L \cup M)^*)^* = (L \cup M)^*$, et $(L^*M^*)^* \subseteq ((L \cup M)^*)^* = (L \cup M)^*$

3 Automates finis et modélisation

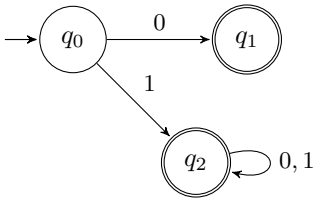
Exercice 12 Construire des automates reconnaissant les langages suivants :

1. L'ensemble des nombres binaires sans zéro inutile en tête.
2. Les mots sur $\{a, b\}$ contenant deux a et/ou deux b consécutifs.
3. Les mots sur $\{a, b\}$ contenant un nombre pair de a et un nombre pair de b.

Solution de l'Exercice 12.

1. **Première méthode :**

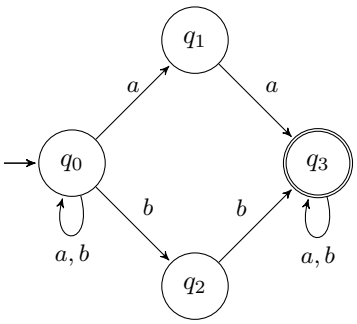
Q	0	1	initial/final
q_0	q_1	q_2	I
q_1	$\times$	$\times$	F
a_2	q_2	q_2	F



Deuxième méthode : On peut procéder par union : « zéro », un automate à 2 états, et « autres », un automate à 2 états. L'union fait le reste : 2 états initiaux, 2 états acceptants, union des relations de transitions.

2.

Q	a	b	initial/final
q_0	q_0, q_1	q_0, q_2	I
q_1	q_3	$\times$	
q_2	$\times$	q_3	
q_3	q_3	q_3	F

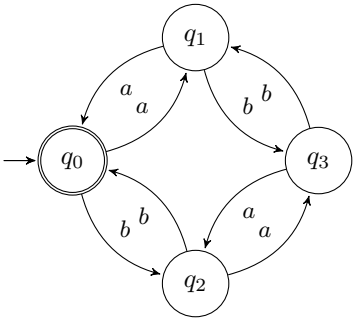


Comme précédemment, on peut procéder par union :

(a) construire un automate pour reconnaître deux a consécutifs est facile (automate non-déterministe à 3 états) ;
(b) idem pour deux b consécutifs ;
(c) faire l'union de ces deux automates (le « et » dans « et/ou » de l'énoncé est juste là pour perturber...)

3.

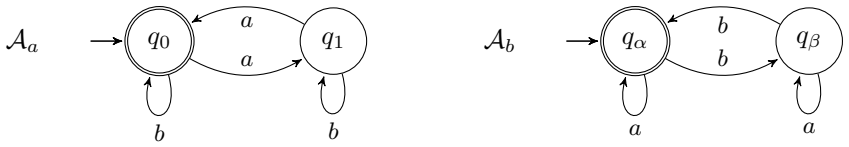
Q	a	b	initial/final
q_0	q_1	q_2	I, F
q_1	q_0	q_3	
q_2	q_3	q_0	
q_3	q_2	q_1	



Remarque : On peut voir cet automate comme le produit cartésien deux deux automates déterministes complets $\mathcal{A}_a$ et $\mathcal{A}_b$ qui reconnaissent respectivement :

(a) pour $\mathcal{A}_a$, les mots contenant un nombre pair de 'a',
(b) pour $\mathcal{A}_b$, les mots contenant un nombre pair de 'b'.

En effet, si on prend ces deux automates :



L'automate pour les mots contenant un nombre pair de 'a' et un nombre pair de 'b' est alors l'automate produit $\mathcal{A}_a \times \mathcal{A}_b$ de $\mathcal{A}_a$ et $\mathcal{A}_b$ défini par

$$(Q_{\mathcal{A}_a} \times Q_{\mathcal{A}_b}, \quad V, \quad \delta_{\mathcal{A}_a} \times \delta_{\mathcal{A}_b}, \quad (i_{\mathcal{A}_a}, i_{\mathcal{A}_b}), \quad F_{\mathcal{A}_a} \times F_{\mathcal{A}_b})$$

avec

$$\delta_{\mathcal{A}_a} \times \delta_{\mathcal{A}_b} = \{((p_1, p_2), x, (q_1, q_2)) \,|\, (p_1, x, q_1) \in \delta_{\mathcal{A}_a} \text{ et } (p_2, x, q_2) \in \delta_{\mathcal{A}_b}\} \,.$$

Autrement dit, les états de $\mathcal{A}_a \times \mathcal{A}_b$ sont des paires d'un état de chaque automate. La construction de l'automate produit permet d'exécuter deux automates déterministes complets en parallèle. Choisir $F_{\mathcal{A}_a} \times F_{\mathcal{A}_b}$ permet de reconnaître l'intersection des langage $\mathcal{L}(\mathcal{A}_a)$ et $\mathcal{L}(\mathcal{A}_b)$ mais on peut faire d'autre choix pour reconnaître l'union ou la différence.

Exercice 13 L'ensemble des littéraux numériques en Python forme un langage, formellement défini dans `https://docs.python.org/3/reference/lexical_analysis.html#numeric-literals`. Dans cet exercice, on considère un sous-ensemble des littéraux entiers et flottants écrits en base 10. Ils sont composés d'une partie entière, d'une partie décimale optionnelle et d'un exposant optionnel ; ils sont définis sur le vocabulaire

$$V \stackrel{\text{def}}{=} \{\mathbf{0}, \ldots, \mathbf{9}, \mathbf{e}, \mathbf{E}, ., +, -\}.$$

On définit les huit ensembles suivants (suite sur la page suivante) :

$$\begin{aligned}
\texttt{nonzerodigit} &\stackrel{\text{def}}{=} \{\mathbf{1}, \ldots, \mathbf{9}\} \\
\texttt{digit} &\stackrel{\text{def}}{=} \{\mathbf{0}\} \cup \texttt{nonzerodigit} \\
\texttt{integer} &\stackrel{\text{def}}{=} \texttt{nonzerodigit}\,(\texttt{digit}^*) \cup \{\mathbf{0}\}^+ \\
\texttt{dot} &\stackrel{\text{def}}{=} \{.\} \\
\texttt{pointfloat} &\stackrel{\text{def}}{=} (\texttt{digit}^*)\,\texttt{dot}\,(\texttt{digit}^+) \cup (\texttt{digit}^+)\,\texttt{dot}
\end{aligned}$$

$$\begin{aligned}
\texttt{exponent} &\stackrel{\text{def}}{=} \{\mathbf{e}, \mathbf{E}\}\,\{\varepsilon, +, -\}\,\texttt{digit}^+ \\
\texttt{exponentfloat} &\stackrel{\text{def}}{=} \big(\texttt{digit}^+ \cup \texttt{pointfloat}\big)\,\texttt{exponent} \\
\texttt{number} &\stackrel{\text{def}}{=} \texttt{integer} \cup \texttt{pointfloat} \cup \texttt{exponentfloat}
\end{aligned}$$

▷ Question 1 Parmi les mots suivants, lesquels appartiennent à `number` ? Lesquelles n'y appartiennent pas ?
.314, **.3E+4**, **0.5E−2**, **0000**, **E67**, **1E7e3**, **6E+1234**, **2E++3.4**

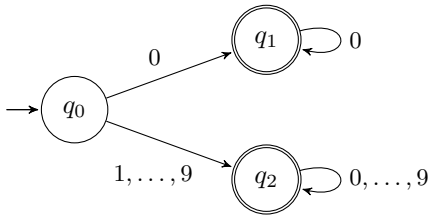
▷ Question 2 Donner un automate qui reconnaît le langage `integer`.

▷ Question 3 Donner un automate qui reconnaît le langage `number`.

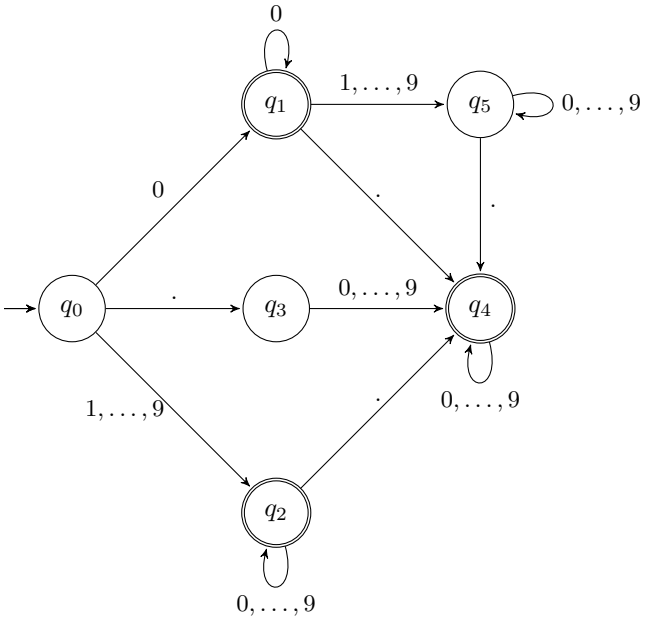
Solution de l'Exercice 13.

▷ Question 1 Eléments dans `number` : **.314**, **.3E+4**, **0.5E−2**, **0000**, **6E+1234**

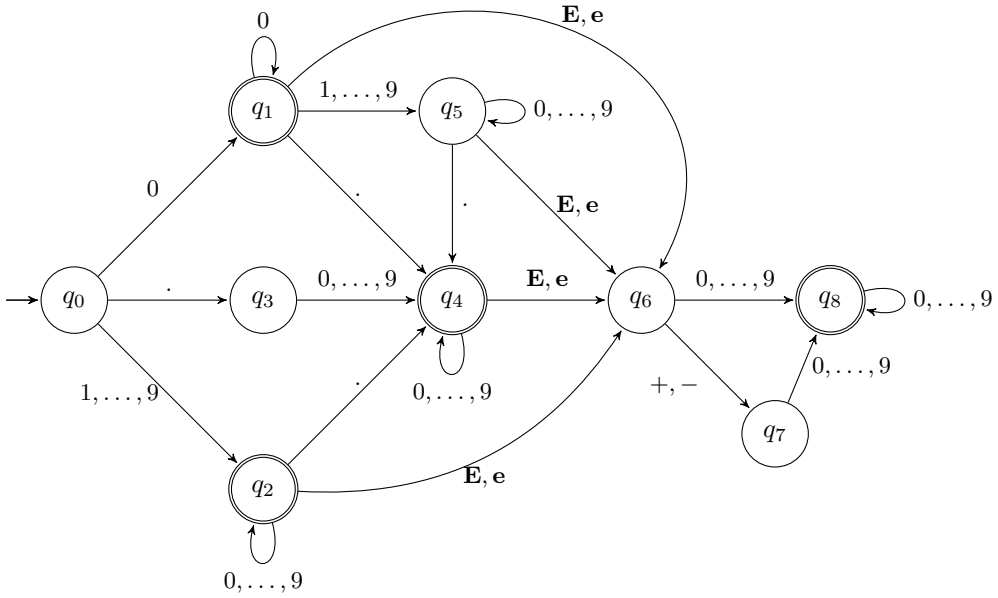
▷ Question 2 L'automate suivant reconnaît `integer` :



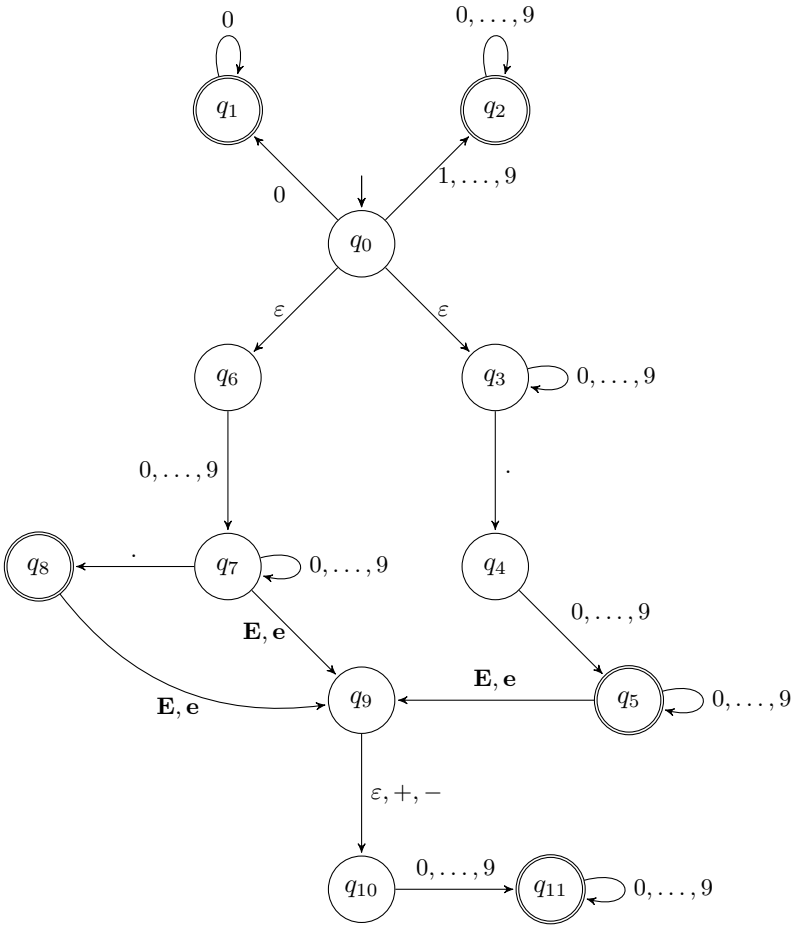
▷ Question 3 Le plus simple pour résoudre cette question est de procéder par étapes.
— L'automate suivant reconnaît les nombres décimaux :



— L'automate suivant reconnaît les littéraux décimaux en Python :



Autre solution moins compacte :



On peut aussi procéder facilement avec des unions et concaténations : un automate pour `pointfloat` (déjà l'union de 2 automates) ; un automate pour `exponent`, un automate pour `digit`$^+$, puis des unions et concaténations avec des ε-transitions, et enfin une union (en dupliquant `pointfloat`).

Exercice 14 Un fermier cherche à faire traverser une rivière à son chou, sa chèvre et son loup. Pour cela, il dispose d'une petite barque qui ne permet de transporter qu'un seul des trois à la fois (en plus de lui-même). Étant donné que

le loup mange la chèvre et que la chèvre mange le chou, le fermier doit faire attention à qui ou quoi il laisse seuls sur chacune des rives. Le fermier peut-il faire traverser la rivière au chou, à la chèvre et au loup sans qu'aucun ne se fasse dévorer ?

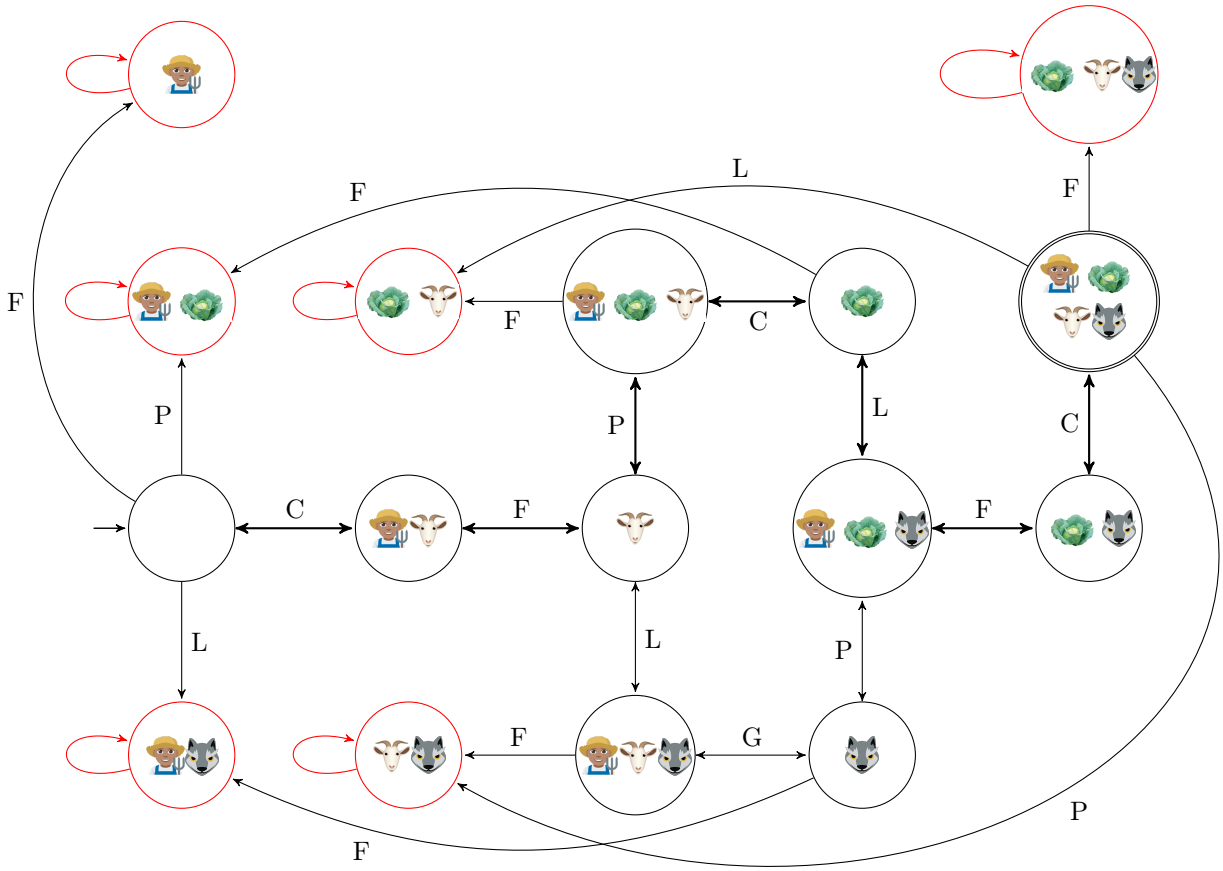
▷ QUESTION 1 Représenter le problème par un automate, en précisant le vocabulaire choisi.

▷ QUESTION 2 Comment déterminer une stratégie à partir de l'automate ? Quelles sont les stratégies optimales ?

Solution de l'Exercice 14.

▷ QUESTION 1 On note le fermier , le chou , la chèvre et le loup . Les états de l'automate sont les différents états du problème : de quels côtés sont le fermier, le chou, la chèvre et le loup ? Il y a donc 16 possibilités, qui représentent les parties de l'ensemble { , , , }.
Les transitions correspondent aux actions possibles du fermier : qui faire traverser. On peut les noter par le légume ou l'animal qui change de côté : P pour le chou, C pour la chèvre, L pour le loup et F lorsque le fermier traverse seul. Lorsqu'un état comporte d'un côté le fermier et de l'autre soit le chou et la chèvre, soit la chèvre et le loup, le jeu est perdu et par convention ces états sont des puits, noté de plus en rouge ci-dessous.

Voici l'automate que l'on obtient :



▷ QUESTION 2 Les stratégies gagnantes sont les mots du langage reconnu par cet automate. Pour en identifier une, il suffit de faire un calcul d'accessibilité pour déterminer si l'état acceptant est accessible. Un parcours en largeur donne alors les deux solutions optimales en nombre de coups : C F P C L F C, l'autre étant obtenue en inversant L et P.

Exercice 15 [A savoir faire] Donner des automates reconnaissant les langages suivants :

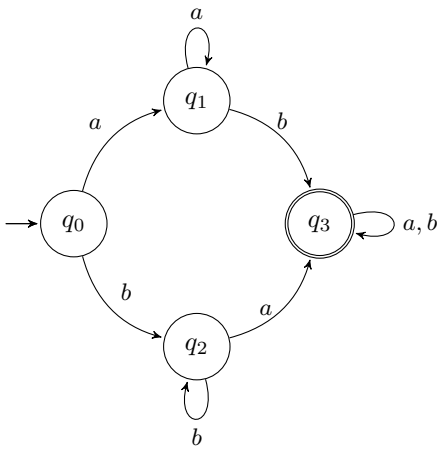
1. $L_1 = \{\omega \in \{a,b\}^* \,|\, \omega \text{ contient au moins un } a \text{ et un } b\}$.
2. $L_2 = \{\omega \in \{a,b\}^* \,|\, \omega \text{ ne contient pas deux } a \text{ consécutifs}\}$.

3. $L_3 = \{\omega \in \{a,b\}^* \,|\, \omega$ ne contient pas *plus de* deux a consécutifs$\}$.
4. $L_4 = \{\omega \in \{a,b\}^* \,|\, \omega$ a bab pour suffixe$\}$.
5. $L_5 = \{\omega \in \{a,b\}^* \,|\,$ la cinquième lettre de ω est un $a\}$.

Solution de l'Exercice 15.

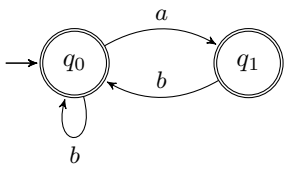
1.

Q	a	b	initial/final
q_0	q_1	q_2	I
q_1	q_1	q_3	
q_2	q_3	q_2	
q_3	q_3	q_3	F



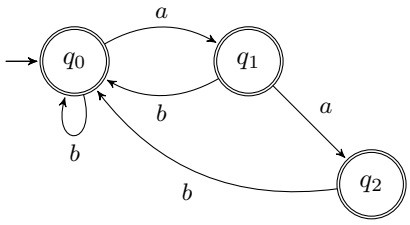
2.

Q	a	b	initial/final
q_0	q_1	q_0	I, F
q_1	$\times$	q_0	F



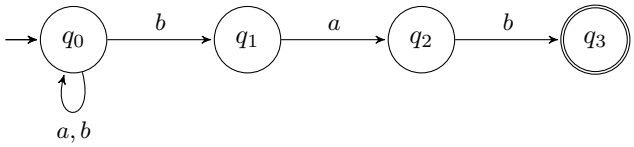
3.

Q	a	b	initial/final
q_0	q_1	q_0	I, F
q_1	q_2	q_0	F
q_2	$\times$	q_0	F



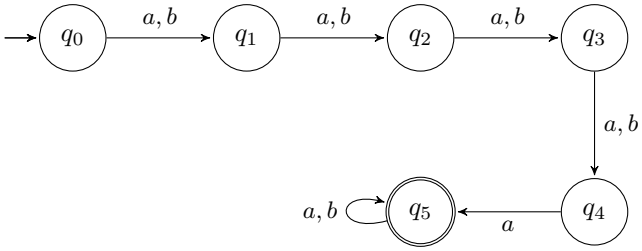
4.

Q	a	b	initial/final
q_0	q_0	q_0, q_1	I
q_1	q_2	$\times$	
q_2	$\times$	q_3	
q_3	$\times$	$\times$	F



5.

Q	a	b	initial/final
q_0	q_1	q_1	I
q_1	q_2	q_2	
q_2	q_3	q_3	
q_3	q_4	q_4	
q_4	q_5	$\times$	
q_5	q_5	q_5	F



Exercice 16 [A savoir faire] Soit L_2 le langage défini inductivement de la façon suivante :
- $\varepsilon \in L_2$.
- Si $w \in L_2$, alors $bw \in L_2$.
- Si $w_1, w_2, w_3 \in L_2$, alors $w_1 a w_2 a w_3 \in L_2$.

▷ Question 1 Définir L_2 par compréhension.

▷ Question 2 Donner un automate qui reconnaît ce langage.

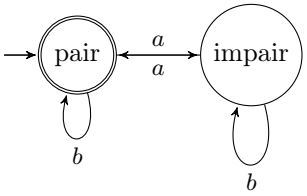
Solution de l'Exercice 16.

▷ Question 1 Posons $L_2' \stackrel{\text{def}}{=} \{w \in \{a,b\}^* \,|\, |w|_a \text{ est pair}\}$, et montrons que $L_2 = L_2'$.
- On prouve que $L_2 \subseteq L_2'$ par induction structurelle : le résultat est clair pour ε. Soit $w \in L_2$, et supposons que $w \in L_2'$. Alors comme $|bw|_a = |w|_a$, on en déduit qu'on a aussi $bw \in L_2'$. De même, pour $w_1, w_2, w_3 \in L_2$, si ces trois mots sont dans L_2', alors il est aisé de vérifier que $w_1 a w_2 a w_3$ contient un nombre pair de a, et est également dans L_2'.
- Avant de démontrer que $L_2' \subseteq L_2$, prouvons par récurrence sur la longueur des mots que $\{b\}^* \subseteq L_2$. Soit $w \in \{b\}^*$ de longueur n, et supposons que tout mot de $\{b\}^*$ de longueur $n-1$ est dans L_2.
  - Si $n = 0$, alors $w = \varepsilon$, qui est bien élément de L_2.
  - Sinon, w est de la forme bw', et par hypothèse, $w' \in L_2$; donc, bw' est également dans L_2.

  Prouvons maintenant par récurrence sur $|w|_a$ que $L_2' \subseteq L_2$.
  - Si $|w|_a = 0$, alors $w \in \{b\}^*$, et est bien dans L_2 d'après ce qui précède.
  - Sinon, w est de la forme $b^i a b^j a w'$, où $i, j \geq 0$ et $w' \in L_2'$. Par hypothèse de récurrence, $w' \in L_2$, et comme b^i et b^j sont également élements de L_2, le mot $b^i a b^j a w'$ est bien un élément de L_2.

▷ Question 2 Voici un automate reconnaissant L_2 :

Q	a	b	initial/final
q_0	q_1	q_0	I, F
q_1	q_0	q_1	



Exercice 17 [Avancé] On considère un vocabulaire V et une relation $R \subseteq V \times V$. On définit

$$H_R \stackrel{\text{def}}{=} \{w_1 \cdots w_k \,|\, k \geq 2, \forall 1 \leq i < k, (w_i, w_{i+1}) \in R\}.$$

Autrement dit, la relation R impose des contraintes sur les symboles qui peuvent se suivre au sein des mots de H_R. Le but de cet exercice est de montrer que H_R est régulier.

On pose $V_0 = \{a, b, c, d, e\}$ et $R_0 = \{(a,b), (b,e), (d,d), (a,c), (d,c), (e,d)\}$.

▷ Question 1 Enumérer les mots de H_{R_0} de longueur inférieure ou égale à 3.

▷ Question 2 Construire un automate qui reconnaît H_{R_0}.

▷ Question 3 En supposant V et R **quelconques**, démontrer que le langage H_R est reconnu par un automate fini.

Solution de l'Exercice 17.

▷ Question 1 Les mots de longueur inférieure ou égale à 3 dans H_{R_0} sont :

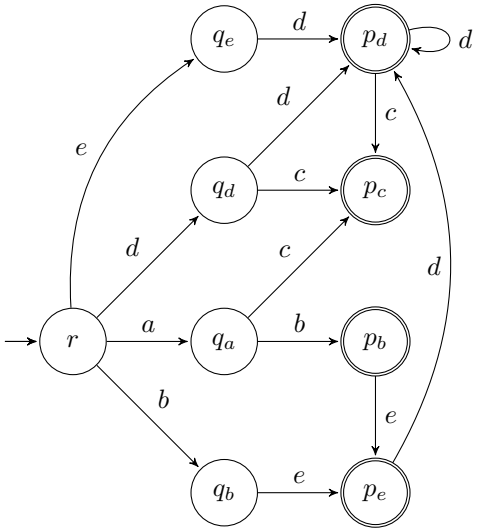
$$ab,\ be,\ dd,\ ac,\ dc,\ ed, \quad abe,\ bed,\ ddc,\ edc,\ edd,\ ddd.$$

▷ Question 2 L'automate reconnaissant H_{R_0} est défini de la façon suivante :
- L'état r est l'état initial, dans lequel aucune lettre n'a été lue.
- Pour $x \in V_0$, l'état q_x est l'état dans lequel on se trouve après avoir lu x comme première lettre.
- Pour $y \in V_0$, l'état p_y est l'état dans lequel on se trouve après avoir lu y, y étant au moins la deuxième lettre.

La relation de transition est la suivante (les états inatteignables ne sont pas montrés) :

Q	a	b	c	d	e	initial/final
r	q_a	q_b	$\times$	q_d	q_e	I
q_a	$\times$	p_b	p_c	$\times$	$\times$	
q_b	$\times$	$\times$	$\times$	$\times$	p_e	
q_d	$\times$	$\times$	p_c	p_d	$\times$	
q_e	$\times$	$\times$	$\times$	p_d	$\times$	
p_b	$\times$	$\times$	$\times$	$\times$	p_e	F
p_c	$\times$	$\times$	$\times$	$\times$	$\times$	F
p_d	$\times$	$\times$	p_c	p_d	$\times$	F
p_e	$\times$	$\times$	$\times$	p_d	$\times$	F

On remarque qu'il y a une transition de q_x vers p_y ou de p_x vers p_y *ssi* le couple (x, y) est dans R et q_x (ou p_x) est accessible (depuis r).



▷ QUESTION 3 Posons $A_R = \langle Q, V, \delta, I, F\rangle$, où :
- $Q = \{q_a \,|\, a \in V\} \uplus \{p_a \,|\, a \in V\} \uplus \{r\}$ (où $\uplus$ représente l'union disjointe),
- $I = \{r\}$,
- $F = \{p_a \,|\, a \in V\}$,
- La relation de transition est définie par

$$\begin{array}{rcll} \delta & \stackrel{\text{def}}{=} & \{(r, a, q_a) \,|\, a \in V\} & \cup \quad \{(q_a, b, p_b) \,|\, (a, b) \in R\} \\ & & & \cup \quad \{(p_a, b, p_b) \,|\, (a, b) \in R\} \end{array}$$

Montrons que $\mathcal{L}(A) = H_R$. On montre, par induction sur k, que pour tout mot w de longueur k, on a l'équivalence : $w \in \mathcal{L}(A) \Leftrightarrow w \in H_R$. On remarque que si $k \in \{0, 1\}$ la propriété est immédiate, puisqu'il est clair que $\mathcal{L}(A)$ et H_R ne contiennent aucun mot de longueur 0 ou 1 (les chemins de r à un état final sont de longueur 2 au minimum dans A).
- Pour $k = 2$:

$$\begin{array}{rcl} w_1w_2 \in H_R & \Leftrightarrow & w_1 \in V \text{ et } (w_1, w_2) \in R \\ & \Leftrightarrow & (r, w_1, q_{w_1}) \in \delta \text{ et } (q_{w_1}, w_2, p_{w_2}) \in \delta \\ & \Leftrightarrow & \exists \text{ un chemin de } r \text{ à } p_{w_2} \text{ dans } A_R \text{ de trace } w_1w_2 \\ & \Leftrightarrow & w_1w_2 \in \mathcal{L}(A_R) \end{array}$$

- Pour $k \geq 3$:

$$\begin{array}{rcl} w_1 \cdots w_k \in H_R & \Leftrightarrow & w_1 \cdots w_{k-1} \in H_R \text{ et } (w_{k-1}, w_k) \in R \\ & \Leftrightarrow & w_1 \cdots w_{k-1} \in \mathcal{L}(A_R) \text{ et } (w_{k-1}, w_k) \in R \text{ (hyp. ind.)} \\ & \Leftrightarrow & \exists \text{ un chemin de } r \text{ à } p_{w_{k-1}} \text{ de trace } w_1 \cdots w_{k-1} \\ & & \text{et } (p_{w_{k-1}}, w_k, p_{w_k}) \in \delta \\ & \Leftrightarrow & \exists \text{ un chemin de } r \text{ à } p_{w_k} \text{ de trace } w_1 \cdots w_k \\ & \Leftrightarrow & w_1 \cdots w_k \in \mathcal{L}(A_R) \end{array}$$

Remarque : Les équivalences sont à manipuler avec précaution. Ainsi, dans le cas où $k \geq 3$, pour justifier formellement les dernières équivalences, il faudrait démontrer (par induction) que tout chemin d'origine r et d'extrémité p_a (pour $a \in V$) a une trace de la forme wa et que si $w_1 \cdots w_k \in H_R$, alors $w_1 \cdots w_{k-1} \in H_R$ et $(w_{k-1}, w_k) \in R$.

Exercice 18 [Avancé] Donner en français un algorithme qui prend en entrée deux automates et qui teste si oui ou non ils reconnaissent le même langage.

Solution de l'Exercice 18. Plusieurs solutions :

1. On minimise les deux automates puis on vérifie s'ils sont isomorphes.
2. On calcule la différence symétrique des deux automates et on regarde si elle accepte un mot.

Exercice 19 [Avancé] Soit L un langage régulier sur un vocabulaire V. On définit le langage $\sqrt{L}$ de la façon suivante :

$$\sqrt{L} \stackrel{\text{def}}{=} \{x \in V^* \mid xx \in L\}.$$

Démontrer que $\sqrt{L}$ est régulier.

Solution de l'Exercice 19. Idées : Faire tourner deux copies d'un automate pour L, pour les deux copies de x. Comme il faut les exécuter en même temps, on doit faire l'automate produit. Enfin, il faut assurer que l'état à la fin de la première copie et celui au début de la seconde est bien le même. Pour cela, on ajoute une troisième composante à l'état, constante le long de l'exécution, qui contient cet état.

Soit $A = \langle Q, V, \delta, I, F\rangle$ un automate régulier qu'on suppose sans ε-transition, et qui reconnaît L. On considère l'automate $A' = \langle Q', V, \delta', I', F'\rangle$, où :

$$\begin{aligned} Q' &= Q \times Q \times Q \\ \delta' &= \{(\langle q_1, q_2, q_3\rangle,\ a,\ \langle q_1', q_2', q_3\rangle) \mid (q_1, a, q_1') \in \delta \wedge (q_2, a, q_2') \in \delta\} \\ I' &= \{\langle q_i, q, q\rangle \mid q_i \in I \wedge q \in Q\} \\ F' &= \{\langle q, q_f, q\rangle \mid q_f \in F \wedge q \in Q\} \end{aligned}$$

L'automate A' vérifie les propriétés suivantes :

Lemme 1 *S'il existe un chemin d'origine $\langle p_1, q_1, r_1\rangle$ et d'extrémité $\langle p_n, q_n, r_n\rangle$ dans A', alors $r_1 = r_n$.*

PREUVE. 1 Par définition de δ', la troisième composante de l'état ne change jamais. Pour le montrer formellement, il faut utiliser une induction sur les chemins. ∎

Lemme 2 *Il existe un chemin d'origine $\langle p_1, q_1, r_1\rangle$, d'extrémité $\langle p_n, q_n, r_n\rangle$ et de trace w dans A' si et seulement s'il existe un chemin de p_1 à p_n de trace w dans A **et** un chemin de q_1 à q_n de trace w dans A.*

PREUVE. 2 La définition de δ' fait que les première et deuxième composantes de l'état suivent les transitions de A, d'où le résultat. Pour le montrer formellement, on peut utiliser une induction sur w. ∎

Théorème 3 *On a $\mathcal{L}(A') = \sqrt{L}$.*

PREUVE. 3 Soit $w \in V^*$. Ce mot est reconnu par A' si et seulement s'il existe un chemin d'un état initial de A' à un état final de A', de trace w. Il existe donc $q_i \in I$, $q_f \in F$ et $q \in Q$ tels qu'il existe un chemin de trace w de $\langle q_i, q, q\rangle$ à $\langle q, q_f, q\rangle$ dans A'. D'après le lemme précédent, ceci est le cas si et seulement s'il existe un chemin de q_i à q de trace w dans A **et** un chemin de q à q_f de trace w dans A. En concaténant ces deux chemins, on en déduit qu'il existe dans A un chemin de q_i à q_f de trace ww, et que $ww \in L$. La réciproque se montre de la même manière. ∎

Exercice 20 Le produit de deux automates est défini de la façon suivante : soient $A_1 = (Q_1, V, \delta_1, I_1, F_1)$ et $A_2 = (Q_2, V, \delta_2, I_2, F_2)$ deux automates qu'on suppose complets et sans ε-transition. Notez qu'ils ne sont pas forcément déterministes. On considère l'automate

$$A_1 \times A_2 \stackrel{\text{def}}{=} (Q_1 \times Q_2,\ V,\ \delta,\ I_1 \times I_2,\ F_1 \times F_2),$$

où δ est défini par : pour tous $\langle q_1, q_2\rangle, \langle q_1', q_2'\rangle \in Q_1 \times Q_2$ et pour tout $a \in V$,

$$(\langle q_1, q_2\rangle, a, \langle q_1', q_2'\rangle) \in \delta \text{ si et seulement si } (q_1, a, q_1') \in \delta_1 \wedge (q_2, a, q_2') \in \delta_2.$$

▷ Question 1 Donner des automates complets et sans ε-transition qui reconnaissent :

1. les mots sur $\{0,1\}$ contenant un nombre pair de 0 ;
2. les mots sur $\{0,1\}$ contenant un nombre impair de 1.

▷ Question 2 Calculer l'automate produit des deux automates de la question précédente. Constater qu'il reconnaît les mots sur $\{0,1\}$ contenant un nombre pair de 0 **et** un nombre impair de 1.

On considère dans la suite deux automates A_1 et A_2 quelconques, complets et sans ε-transition. On souhaite prouver que $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

▷ Question 3 **[Avancé]** Montrer par induction sur w l'équivalence entre les deux propositions suivantes :

1. il existe un chemin de trace w de l'origine p_1 à l'extrémité q_1 dans A_1
   et il existe un chemin de trace w de l'origine p_2 à l'extrémité q_2 dans A_2 ;
2. il existe un chemin de trace w de l'origine $\langle p_1, p_2\rangle$ à l'extrémité $\langle q_1, q_2\rangle$ dans $A_1 \times A_2$.

▷ Question 4 En déduire que $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

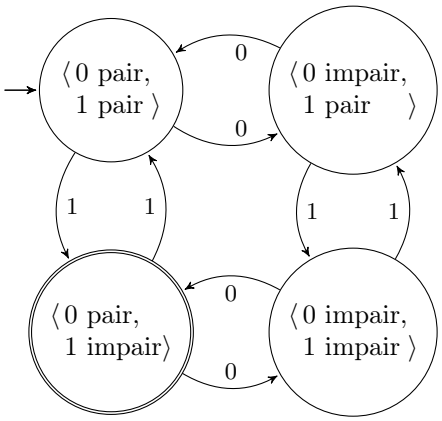
▷ Question 5 Quel langage aurait été reconnu si l'ensemble des états finaux de $A_1 \times A_2$ avait été $(F_1 \times Q_2) \cup (Q_1 \times F_2)$ et non pas $F_1 \times F_2$? Justifier.

▷ Question 6 Que se passe-t-il si les automates A_1 et A_2 ne sont pas complets ? Considérer les cas où l'ensemble des états acceptants est $F_1 \times F_2$ ou $(F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Solution de l'Exercice 20.

▷ Question 1 Pour déterminer si le nombre de 0 dans un mot est pair, l'information à maintenir est la parité du nombre de 0, à mettre à jour à chaque nouveau symbole lu. Il faut donc deux états, un état « nombre pair de 0 » et un état « nombre impair de 0 ». De même pour un nombre impair de 1. Ainsi, on a les deux automates suivants, qui reconnaissent respectivement les mots avec un nombre pair de 0 et les mots avec un nombre impair de 1. Remarquez que les états initiaux sont ceux avec un nombre pair, car le mot vide contient un nombre pair de 0 et de 1.

▷ Question 2 L'intuition à garder en tête pour comprendre l'automate produit est d'exécuter en parallèle deux automates. Les états de l'automate produit sont donc des couples, dont la première composante sert à exécuter le premier automate et la seconde composante le second automate. L'automate produit est donc :



▷ Question 3 Le résultat se démontre par induction sur w.

Cas $w = \varepsilon$ **:** Le chemin vide () est un chemin de p_1 à p_1 (donc $q_1 = p_1$ ici) de trace ε dans A_1 et de même, () est un chemin de p_2 à p_2 de trace ε dans A_2 et () est un chemin de $\langle p_1, p_2\rangle$ à $\langle p_1, p_2\rangle$ de trace ε dans $A_1 \times A_2$.

Cas $w = aw'$ **:** Comme A_1 ne contient pas d'ε-transitions, un chemin de trace aw' dans A_1 est de la forme $(p_1, a, r_1)\chi_1$, avec $(p_1, a, r_1) \in \delta_1$ et χ_1 un chemin de r_1 à q_1 de trace w' dans A_1. De même pour A_2 : un chemin de trace aw' est de la forme $(p_2, a, r_2)\chi_2$. L'hypothèse d'induction sur χ_1 et χ_2 donne un chemin χ dans $A_1 \times A_2$ de $\langle r_1, r_2\rangle$ à $\langle q_1, q_2\rangle$ de trace w'. De $(p_1, a, r_1) \in \delta_1$ et $(p_2, a, r_2) \in \delta_2$, la définition de δ nous donne que $(\langle p_1, p_2\rangle, a, \langle r_1, r_2\rangle) \in \delta$ donc $(\langle p_1, p_2\rangle, a, \langle r_1, r_2\rangle)\chi$ est un chemin de $\langle p_1, p_2\rangle$ à $\langle q_1, q_2\rangle$ de trace aw' dans $A_1 \times A_2$.

▷ QUESTION 4 On peut raisonner directement par équivalence ici :

$$
\begin{aligned}
w \in \mathcal{L}(A_1 \times A_2) &\overset{\text{def}}{\iff} \text{il existe un chemin dans } A_1 \times A_2 \text{ d'origine } \langle i_1, i_2\rangle \text{ et d'extrémité } \langle q_1, q_2\rangle \in F_1 \times F_2 \\
&\overset{\text{quest 3}}{\iff} \text{il existe un chemin de trace } w \text{ de l'origine } i_1 \text{ à l'extrémité } q_1 \in F_1 \text{ dans } A_1 \\
&\qquad \text{et il existe un chemin de trace } w \text{ de l'origine } i_2 \text{ à l'extrémité } q_2 \in F_2 \text{ dans } A_2 \\
&\overset{\text{def}}{\iff} w \in \mathcal{L}(A_1) \text{ et } w \in \mathcal{L}(A_2)
\end{aligned}
$$

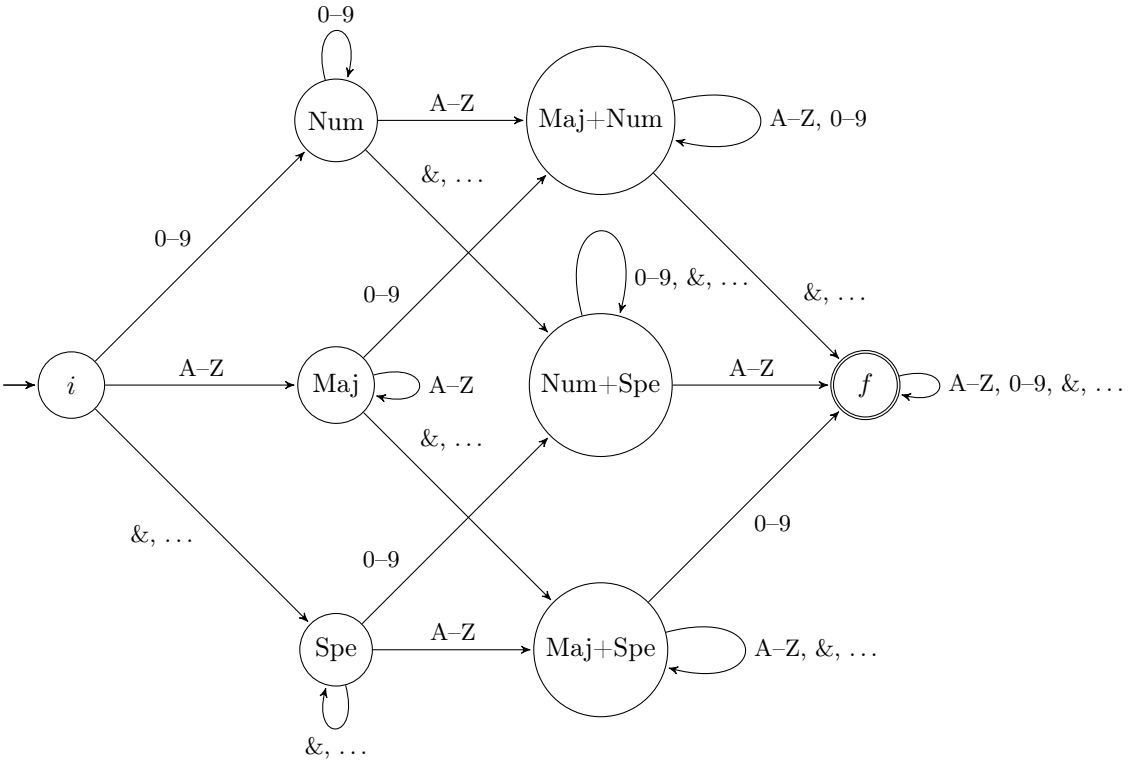
▷ QUESTION 5 Si l'ensemble des états finaux de $A_1 \times A_2$ avait été $(F_1 \times Q_2) \cup (Q_1 \times F_2)$, l'automate aurait reconnu $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. On peut adapter la preuve de la question précédente pour le montrer proprement. En effet, $(F_1 \times Q_2) \cup (Q_1 \times F_2)$ signifie : la première composante accepte (cas $F_1 \times Q_2$) ou la seconde composante accepte (cas $Q_1 \times F_2$).

▷ QUESTION 6 Si l'un des automates n'est pas complet, chaque transition qui manque ne pourra être réalisée dans l'automate produit. Pour $F_1 \times F_2$, cela ne pose pas de problème car les mots acceptés sont reconnus par les deux automates donc une transition manquante aurait de toute façon donné un mot non accepté. En revanche, pour $(F_1 \times Q_2) \cup (Q_1 \times F_2)$, cela peut poser problème. Par exemple si A_1 ne contient aucune transition (donc $\mathcal{L}(A_1) = \emptyset$), on a $\mathcal{L}(A_1 \times A_2) = \emptyset \neq \mathcal{L}(A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

Exercice 21 Lors d'un changement de mot de passe, il est courant d'imposer des *règles de composition*, c'est à dire des règles syntaxiques que doit satisfaire le mot de passe afin d'être accepté. Par exemple, on peut demander à ce qu'il y ait au moins un chiffre, une majuscule et un caractère spécial. Donner un automate qui accepte les mots de passe de cet exemple.

Solution de l'Exercice 21. Une méthode naïve est de considérer les divers entrelacements possibles des caratères à observer, ici $2^3 = 8$ chemins possibles de trois états chacun. Au total, 24 états.

Une seconde méthode plus simple est de voir que le langage à reconnaître correspond à l'intersection de 3 langages vérifiant chacun une condition, et donc au produit de 3 automates de 2 états chacun. Une autre façon d'arriver au même automate est de construire directement les états pertinents : aucune règle validée, une règle validée (et laquelle), deux règles validées (donc encore quelle règle à valider), trois règles validées. Ce faisant, on a besoin de huit états seulement :

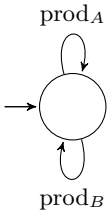


Exercice 22 Lorsqu'une personne souhaite donner des informations à une autre mais que les deux ne peuvent se rencontrer (se synchroniser), il est courant d'utiliser une boîte au lettre (électronique ou non), où la première personne dépose les informations que la seconde récupérera plus tard. En informatique, cela se produit dès que deux processus désynchronisés veulent communiquer et la boîte aux lettres s'appelle alors un *buffer*.

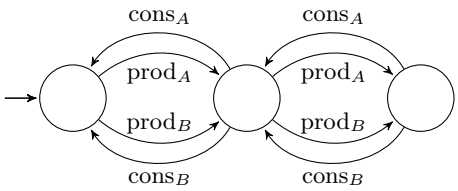
L'objectif de cet exercice est de modéliser *un buffer à deux places* (c.-à-d. pouvant stocker au plus deux données) entre un producteur et un consommateur pour deux types de ressources A et B, le tout à l'aide d'automates. Le

vocabulaire (également appelé *ensemble de synchronisation*) est : { prod_A, prod_B, cons_A, cons_B } où prod représente « produire » et cons « consommer », l'indice indiquant le type de ressource impliquée.

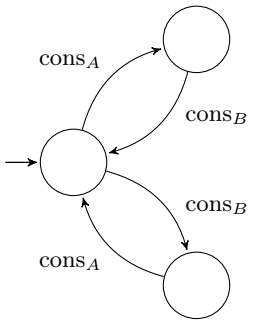
Supposons les trois automates suivants décrivant respectivement un producteur, un buffer et un consommateur. Notez qu'ils ne sont pas complets et qu'ils n'ont pas d'états acceptants car le système n'est pas sensé s'arrêter et évolue indéfiniment à chaque événement (chaque symbole prod_A, prod_B, cons_A, cons_B lu). Dans l'automate du consommateur, il y a implicitement des boucles prod_A et prod_B sur chaque état car le consommateur ignore ces événements. De même pour l'automate du producteur, il y a implicitement des boucles cons_A et cons_B.



Producteur



Buffer à deux places

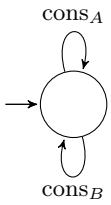


Consommateur

▷ Question 1 Calculer le produit des automates prod×buffer×cons.

▷ Question 2 Quel est le problème sur les types de ressources mis en avant dans ce produit ?

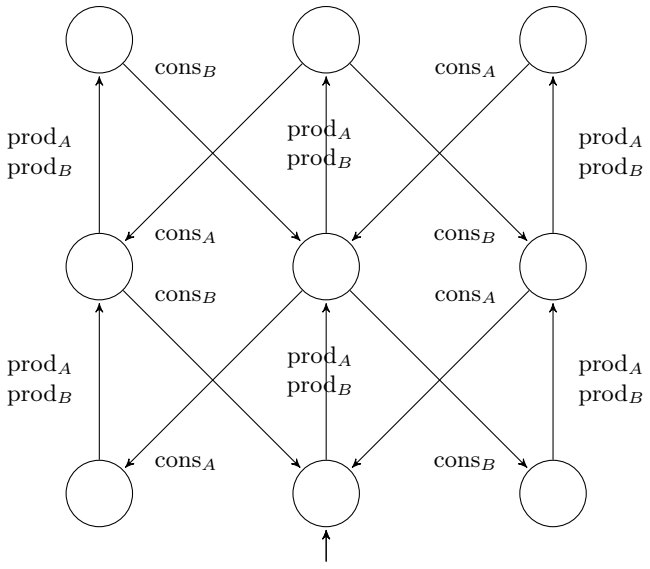
▷ Question 3 En supposant le consommateur plus classique ci-dessous, modéliser à nouveau l'automate attendu du buffer à deux places.



▷ Question 4 Quel automate faudrait-il donner pour le buffer afin de corriger le problème identifié ?

Solution de l'Exercice 22.

▷ Question 1

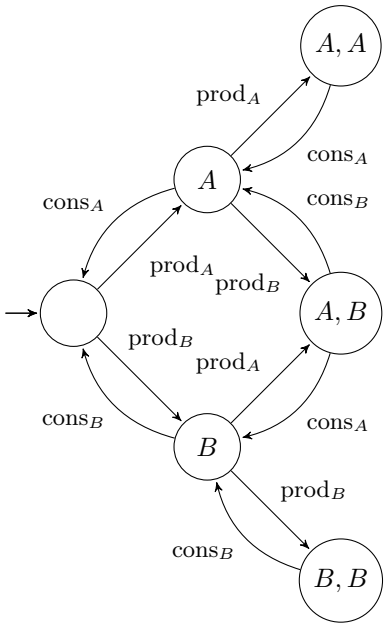


▷ Question 2 On ne fait pas la différence entre les ressources de type A et B : on peut produire un A et le consommer comme un B.

Remarque : Ces problèmes peuvent être détectés automatiquement par logique temporelle puis model-checking.

▷ Question 3 Comme le producteur et le consommateur n'ont qu'un état, le produit se réduit à l'automate du buffer. Notez que cela revient à ne plus faire du tout la distinction des types de ressources, de sorte qu'on pourrait utiliser simplement prod et cons comme événements.

▷ Question 4 On doit faire la distinction entre les resources de type A et B donc maintenir dans l'état quel est le type des ressources stockées.



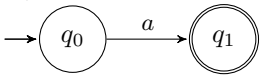
Exercice 23 Pour construire un automate reconnaissant le langage complémentaire d'un automate A donné, on propose d'inverser simplement les états acceptants et non acceptants de A. Ainsi, les chemins non-acceptants deviennent acceptants et réciproquement.

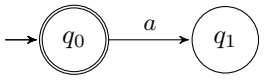
▷ Question 1 Montrer que cette transformation est incorrecte.

▷ Question 2 Donner une condition suffisante sur A pour que la transformation soit correcte. Justifier en montrant la correction de la transformation dans ce cas. Est-ce que la condition est nécessaire ?

Solution de l'Exercice 23.

▷ Question 1 Sur le vocabulaire $V = \{a\}$, si on prend $L = \{a\}$, l'automate le plus simple pour L est :



La transformation proposée donne l'automate suivant :  qui reconnaît le langage $\{\varepsilon\}$ qui n'est pas le complémentaire de L.

▷ Question 2 Le problème vient du fait que l'automate n'est pas complet donc il manque des chemins. Dans l'automate de départ, ce n'est pas un problème car ces chemins ne sont pas acceptants mais pour le complémentaire, ils devraient l'être. De même, si l'automate possède un chemin acceptant et un chemin non acceptant *pour un même mot*, alors le mot est accepté et le sera toujours après transformation.

Une condition suffisante pour éviter ces problèmes est que l'automate A soit déterministe complet. Dans ce cas, on sait que pour tout mot w, il y a toujours un unique chemin depuis l'état initial de trace w, donné par δ^*. Ainsi, $w \in \mathcal{L}(A') \iff \delta^*(i,w) \in F' \iff \delta^*(i,w) \notin F \iff w \notin \mathcal{L}(A)$.

La condition n'est pas strictement nécessaire : on peut modifier marginalement un automate déterministe complet pour le rendre non déterministe ou non complet sans créer de problème. Voici quelques exemples :

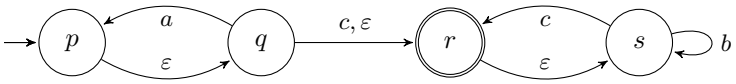
— couper un état en deux en rejoignant chaque moitié par une ε-transition,

— dupliquer l'automate et supprimer des parties de l'une des copies pour la rendre non déterimiste.

On peut noter que ces cas sont plutôt artificiels et qu'en pratique, partir d'un automate déterministe complet est une solution simple et efficace.

4 Élimination des ε-transitions

Exercice 24 [A savoir faire] Construire un automate sans ε-transition équivalent à celui ci-dessous :



Solution de l'Exercice 24. On calcule les ensembles d'états ε-accessibles, soit directement, soit par itération selon la méthode vue en cours :

$\mathrm{Acc}_\varepsilon(\_)$	p	q	r	s
$k=0$	p	q	r	s
$k=1$	p,q	q,r	r,s	s
$k=2$	p,q,r	q,r,s	r,s	
$k=3$	p,q,r,s	q,r,s		
$k=4$	p,q,r,s			

ce qui nous donne

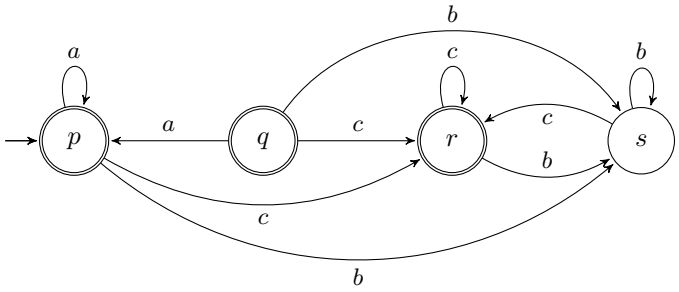
Q	$\mathrm{Acc}_\varepsilon(q)$
p	$\{p,q,r,s\}$
q	$\{q,r,s\}$
r	$\{r,s\}$
s	$\{s\}$

Notez que dans le tableau de calcul, on arrête chaque colonne lorsqu'elle a stabilisée.

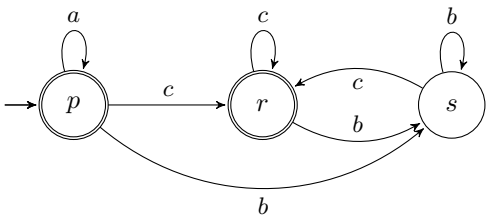
On en déduit la relation de transition de l'automate :

δ	a	b	c	initial/final
p	p	s	r	F
q	p	s	r	F
r	$\times$	s	r	F
s	$\times$	s	r	

L'automate obtenu est donc :



On remarque que l'état q n'est plus accessible donc peut être supprimé.

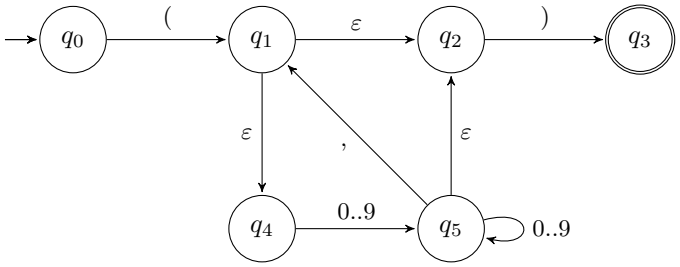


Exercice 25 [A savoir faire] On s'intéresse à l'ensemble des tuples d'entiers (simples) en Python. Dans ce langage, il est possible d'écrire :

- Le tuple vide : ()
- Un tuple à un élément : $(42,)$
- Un couple : $(42, 29)$ ou bien $(42, 29,)$
- Un triplet : $(12, 25, 37)$ ou bien $(12, 25, 37,)$
- etc...

En particulier, (123) ne représente **pas** un tuple : il s'agit de la valeur 123 entourée de parenthèses superflues. Les autres expressions telles que $(,)$ ou encore $(, 42, 29)$ sont interdites.

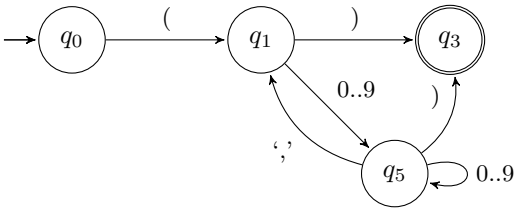
On propose l'automate suivant pour reconnaître les tuples d'entiers.



Construire un automate sans ε-transition équivalent à celui proposé. L'automate proposé répond-il bien aux spécifications ? Justifier.

Solution de l'Exercice 25.

q	$\mathrm{Acc}_\varepsilon(q)$
q_0	q_0
q_1	q_1, q_2, q_4
q_2	q_2
q_3	q_3
q_4	q_4
q_5	q_2, q_5



L'automate ne répond pas à la spécification : on reconnaît par exemple (1) en trop.

Exercice 26 Soit $A = (Q, V, \delta, I, F)$ un automate. On rappelle qu'un état $q \in Q$ est *accessible dans* A s'il existe un chemin dans A dont l'origine est dans I et l'extrémité est q. L'état q est *productif dans* A s'il existe un chemin dans A dont l'origine est q et l'extrémité est dans F.

On note $\mathrm{Acc}(A)$ l'ensemble des états accessibles dans A, et $\mathrm{Prod}(A)$ l'ensemble des états productifs dans A.

▷ Question 1 Soit $Q = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$. On considère l'automate $A = (Q, \{a, b\}, \delta, \{p_0, p_1\}, \{p_3, p_4\})$, où la relation de transition δ est définie par :

δ	a	b
p_0	p_1	p_2
p_1	-	p_3
p_2	p_1	p_3, p_4
p_3	-	-
p_4	-	p_3
p_5	p_6	p_4
p_6	p_4	p_4

Déterminer les ensembles $\mathrm{Acc}(A)$ et $\mathrm{Prod}(A)$.

On considère maintenant un automate A **quelconque, sans ε-transition**.

▷ Question 2 Définir un algorithme qui calcule l'ensemble des états accessibles dans A.

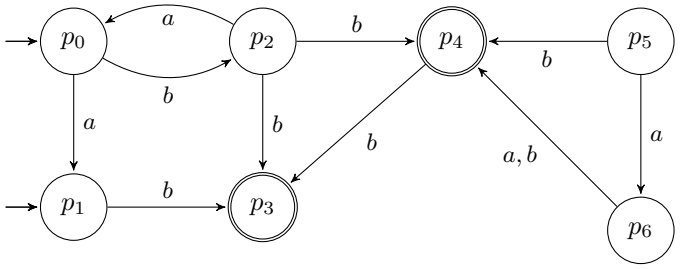
▷ Question 3 Même question pour l'ensemble des états productifs dans A.

▷ Question 4 Donner une condition nécessaire et suffisante sur $\mathrm{Acc}(A)$ et $\mathrm{Prod}(A)$ pour que $\mathcal{L}(A) \neq \emptyset$.

▷ Question 5 Donner une condition suffisante sur A permettant d'assurer $\mathcal{L}(A) = V^*$?

Solution de l'Exercice 26.

▷ Question 1



On a donc :

$$\begin{aligned} \mathrm{Acc}(A) &= \{p_0, p_1, p_2, p_3, p_4\} \\ \mathrm{Prod}(A) &= Q \end{aligned}$$

▷ Question 2 Définition inductive de $\mathrm{Acc}(A)$:

Base : $I \subseteq \mathrm{Acc}(A)$.

Induction : Si $p \in \mathrm{Acc}(A)$ et $(p, a, q) \in \delta$ (où $a \in V \cup \{\varepsilon\}$), alors $q \in \mathrm{Acc}(A)$.

Remarquez qu'il s'agit d'une définition très similaire à $\mathrm{Acc}_\varepsilon(\_)$: on modifie le cas de base et on peut quivre toutes les transitions, pas uniquement celles étiquetées par ε. On en déduit l'algorithme suivant :

$i := 0$
$A_0 := I$
répéter
  $i := i + 1$
  $A_i := A_{i-1} \cup \{q \in Q \mid \exists p \in A_{i-1}, \exists a \in V \cup \{\varepsilon\}, (p, a, q) \in \delta\}$
jusqu'à $A_i = A_{i-1}$
renvoyer A_i

▷ Question 3 Contrairement à Acc, on « remonte » les transitions plutôt que de les suivre. Définition inductive de $\mathrm{Prod}(A)$:

Base : $F \subseteq \mathrm{Prod}(A)$.

Induction : Si $q \in \mathrm{Prod}(A)$ et $(p, a, q) \in \delta$, alors $p \in \mathrm{Prod}(A)$.

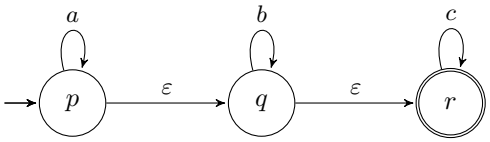
D'où l'algorithme suivant :

$i := 0$
$P_0 := F$
répéter
  $i := i + 1$
  $P_i := P_{i-1} \cup \{p \in Q \mid \exists q \in P_{i-1}, \exists a \in V \cup \{\varepsilon\}, (p, a, q) \in \delta\}$
jusqu'à $P_i = P_{i-1}$
renvoyer P_i

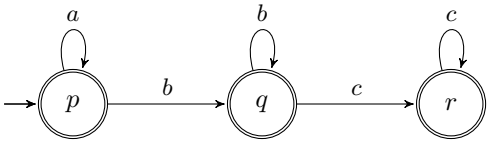
▷ Question 4 On a $\mathcal{L}(A) \neq \emptyset$ ssi $\mathrm{Acc}(A) \cap \mathrm{Prod}(A) \neq \emptyset$ ssi $I \cap \mathrm{Prod}(A) \neq \emptyset$ ssi $\mathrm{Acc}(A) \cap F \neq \emptyset$ (toutes ces conditions sont équivalentes).

▷ Question 5 On a $\mathcal{L}(A) = V^*$ ssi $\mathrm{Acc}(A) \subset F$ et tous les états de $\mathrm{Acc}(A)$ peuvent effectuer toutes les transitions (« l'automate $\mathrm{Acc}(A)$ est complet »).

Exercice 27 [A savoir faire] Construire un automate sans ε-transition équivalent à celui ci-après. Quel est le langage reconnu par cet automate ?

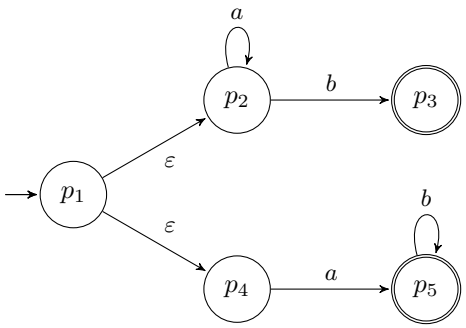


Solution de l'Exercice 27.



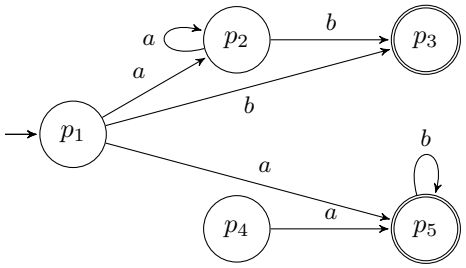
5 Déterminisation

Exercice 28 [A savoir faire] Déterminiser l'automate suivant :



Solution de l'Exercice 28. L'ensemble des états ε-accessibles depuis p_1 est $\{p_1, p_2, p_4\}$; les autres ensembles sont des singletons. On en déduit la relation de transition suivante pour l'automate obtenu après suppression des ε-transitions :

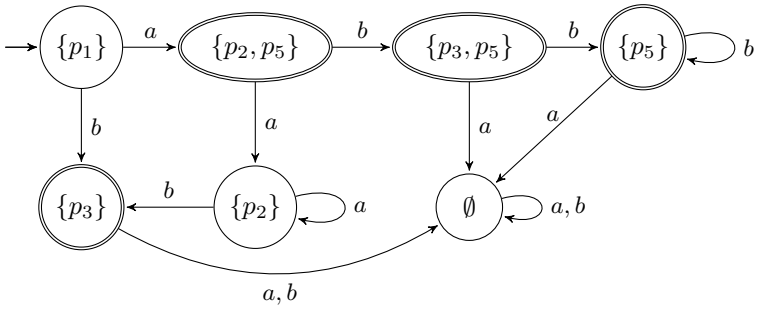
δ	a	b
p_1	p_2, p_5	p_3
p_2	p_2	p_3
p_3	-	-
p_4	p_5	-
p_5	-	p_5



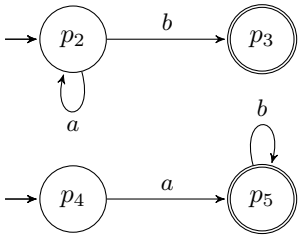
On remarque que p_4 n'est plus accessible et peut donc être supprimé, ce que la déterminisation va faire pour nous.

Déterminisation :

δ	a	b	I/F
p_1	p_2, p_5	p_3	I
p_2, p_5	p_2	p_3, p_5	F
p_3	$\emptyset$	$\emptyset$	F
p_2	p_2	p_3	
p_3, p_5	$\emptyset$	p_5	F
p_5	$\emptyset$	p_5	F
$\emptyset$	$\emptyset$	$\emptyset$	



Un des intérêts de cet exercice : il est facile de décrire en français le langage reconnu sur l'automate d'origine (« autant de a qu'on veut puis un b, ou alors un a puis autant de b qu'on veut »), mais c'est plus difficile sur l'AFD... Sur l'automate d'origine on voit bien l'union (avec les ε) ; on la verrait aussi bien en partant de l'automate

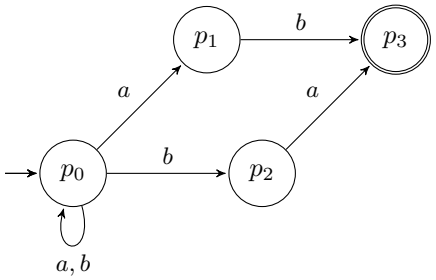


Exercice 29 [A savoir faire] Construire des automates déterministes reconnaissant les langages sur $\{a, b\}$ suivants :

1. L'ensemble des mots terminés par ab ou bien par ba.
2. Le langage $\{aab\}^*\{b\}$.
3. L'ensemble des mots contenant au moins deux fois la séquence ab.
4. Le langage $\{a\}^*\{aba\}^*$.

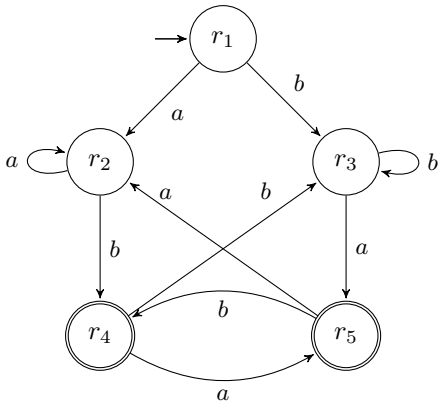
Solution de l'Exercice 29. Ici, selon leur provenance, certains ont tendance à chercher directement un automatate déterministe. Sur les automates 3 et 4, il est facile de se tromper...

1. Automate non-déterministe :

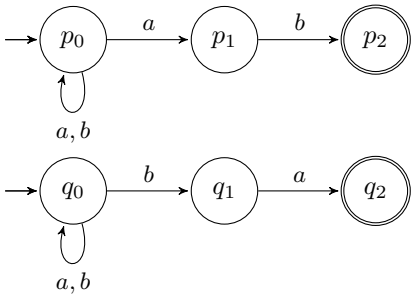


Déterminisation :

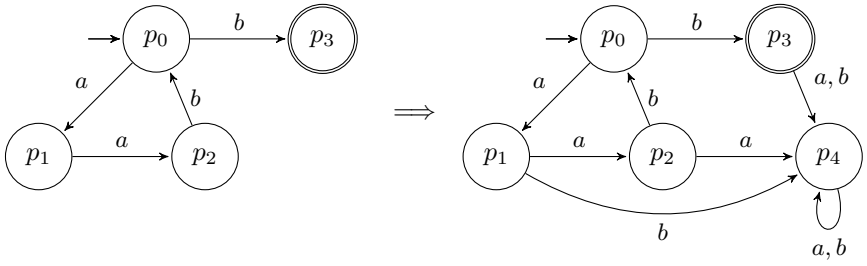
Nom	I/F	δ	a	b
r_1	I	p_0	p_0, p_1	p_0, p_2
r_2		p_0, p_1	p_0, p_1	p_0, p_2, p_3
r_3		p_0, p_2	p_0, p_1, p_3	p_0, p_2
r_4	F	p_0, p_2, p_3	p_0, p_1, p_3	p_0, p_2
r_5	F	p_0, p_1, p_3	p_0, p_1	p_0, p_2, p_3



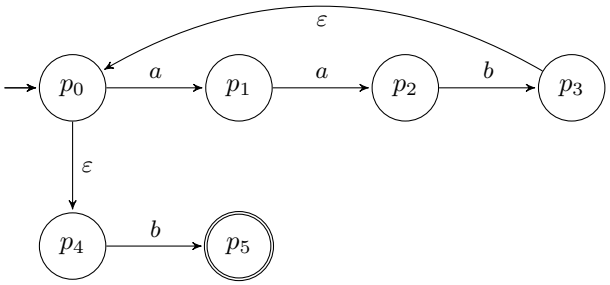
On peut aussi partir de



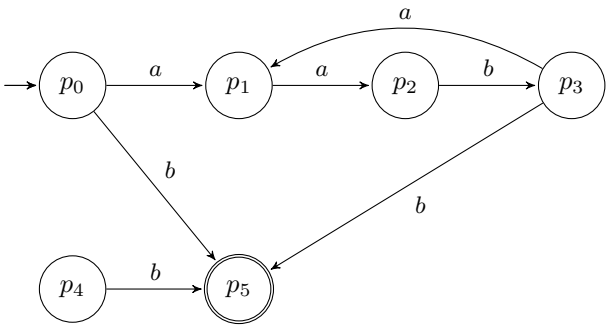
2. On peut directement construire un automate déterministe non complet, puis le compléter :



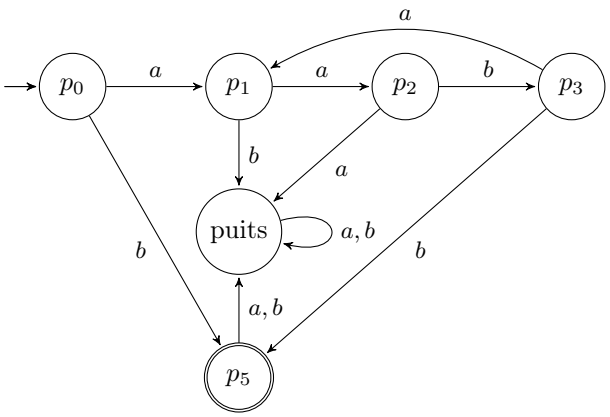
Ou bien passer par un automate non-déterministe, qu'on déterminise (et dont on supprime d'abord les ε-transitions au besoin) :



Après suppression des ε-transitions (non détaillée) :

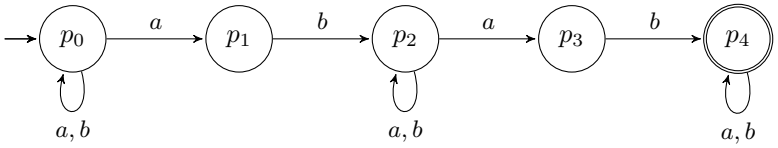


L'automate est déjà déterministe mais ni complet ni initialement connecté. On peut corriger cela en supprimant l'état p_4 pour devenir initialement connecté et en ajoutant un état puits pour devenir complet :



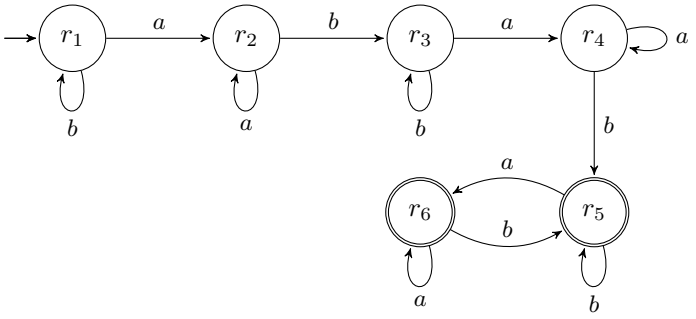
Remarquez que les automates déterministes obtenus par les deux méthodes n'ont pas le même nombre d'états mais sont équivalents.

3. Automate non-déterministe :



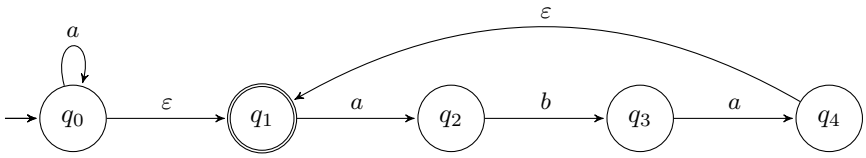
Déterminisation :

Nom	I/F	δ	a	b
r_1	I	p_0	p_0, p_1	p_0
r_2		p_0, p_1	p_0, p_1	p_0, p_2
r_3		p_0, p_2	p_0, p_1, p_2, p_3	p_0, p_2
r_4		p_0, p_1, p_2, p_3	p_0, p_1, p_2, p_3	p_0, p_2, p_4
r_5	F	p_0, p_2, p_4	p_0, p_1, p_2, p_3, p_4	p_0, p_2, p_4
r_6	F	p_0, p_1, p_2, p_3, p_4	p_0, p_1, p_2, p_3, p_4	p_0, p_2, p_4



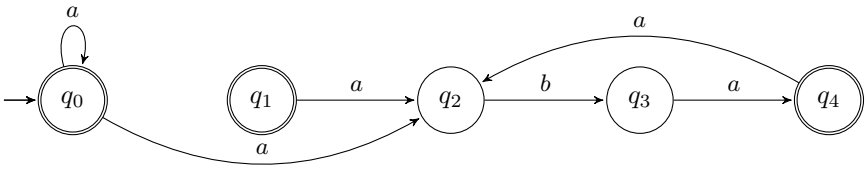
En question subsidiaire, on peut demander « exactement 2 fois la séquence ab ».

4. Automate non déterministe :

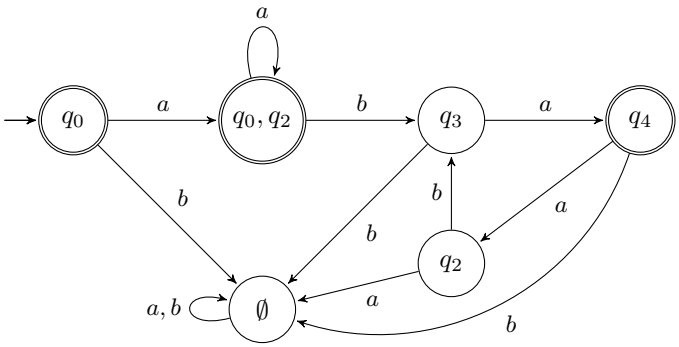


À comparer avec la question précédente : ici si on ne met pas d'ε-transition entre q_0 et q_1 ça ne marche pas... mais on peut faire directement aba en triangle (sans q_4 et son ε-transition vers q_1).

Automate non déterministe sans ε-transition :



Automate déterminisé :



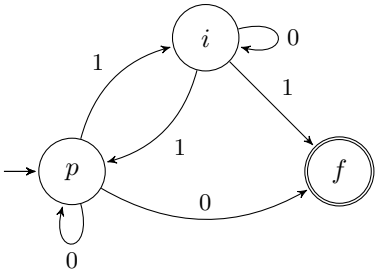
Exercice 30 [A savoir faire] On s'intéresse au langage L des mots binaires dont le dernier bit est un bit de parité. Plus précisément, un mot $wx \in \{0,1\}^+$ est dans L si le nombre de 1 dans w est impair et x vaut 1, ou si le nombre de 1 dans w est pair et x vaut 0. Autrement dit, on choisit x pour que le nombre de 1 dans wx soit pair.

Exemples : 0, 011011, 1010 et 001111 sont dans L.
Contre-exemples : 11010, 110001 et ε ne sont pas dans L.

Construire un automate déterministe complet reconnaissant L.

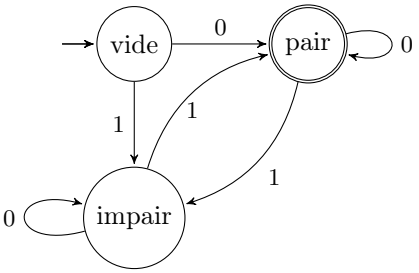
Solution de l'Exercice 30.

On commence par construire un automate qui compte la parité d'un mot w, puis on lui ajoute une transition vers un nouvel état pour le bit de parité. Ce dernier état est le seul état final.



Déterminisation :

δ	0	1	I/F
p	p, f	i	I
p, f	p, f	i	F
i	i	p, f	



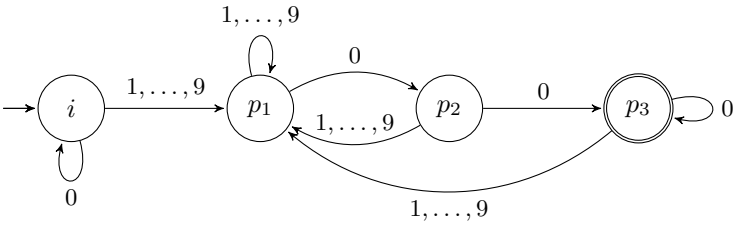
Remarque : On peut aussi produire directement le résultat en reconnaissant les mots non vides avec un nombre pair de 1 (d'où les noms des états a posteriori dans l'AFD.)

Exercice 31 [A savoir faire] Construire des automates déterministes complets reconnaissant :

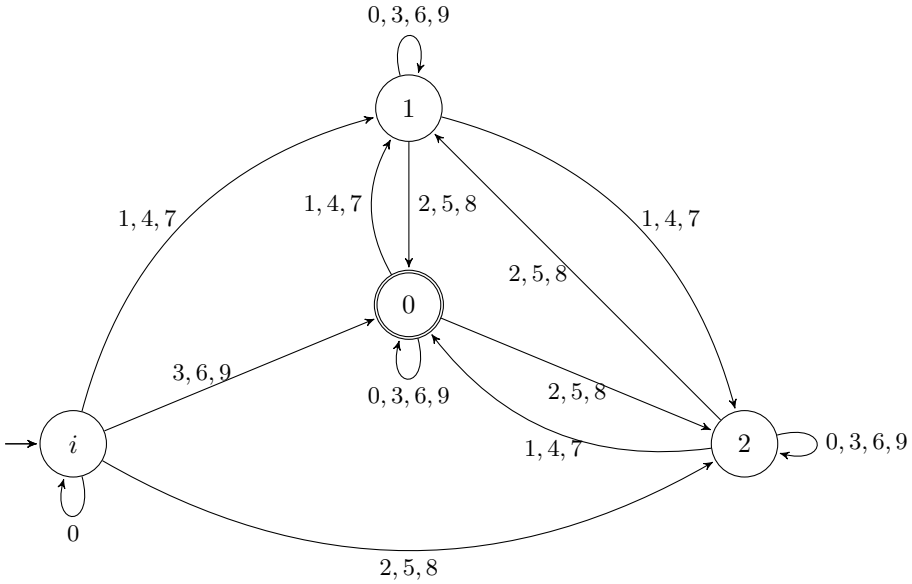
1. les nombres entiers > 0, en base 10, qui sont des multiples de 100 ;
2. les nombres entiers > 0, en base 10, qui sont des multiples de 3 ;
3. **[Avancé]** pour n et k entiers, les nombres entiers > 0, en base k, qui sont multiples de n.

Solution de l'Exercice 31.

1. Multiples de 100 :



2. Multiples de 3 :



3. Multiples de n en base k :
   Les information dont on doit se souvenir sont le reste de la division euclidienne du nombre lu jusqu'à présent. Ainsi, on a naturellement n états, plus un état initial i pour refuser le mot vide. Lorsqu'on a lu w, on se trouve dans l'état p qui correspond à la congruence modulo n du nombre $\overline{w}$ représenté par w en base k : $\overline{w} \equiv p$ [modulo n]. Lorsqu'on ajoute un nouveau chiffre x, la nouvelle classe de congruence devient :

$$\overline{wx} \equiv \overline{w} * k + x \equiv p * k + x \text{ [modulo } n]$$

   qui ne dépend que de l'état courant p et du symbole qu'on lit (x), k étant une constante, donc définit bien une relation de transition. L'ensemble des états acceptants est $\{0\}$, les entiers dont le reste modulo n est 0.

   Au total, l'automate est défini par

$$\mathcal{A} \stackrel{\text{def}}{=} (\{i\} \cup \{0, \dots n\}, \quad \{0, k-1\}, \quad \delta, \quad \{i\}, \quad \{0\})$$

où

$$\delta \stackrel{\text{def}}{=} \{(i, x, x \mod n) \mid 0 \leqslant x < k\} \\ \cup \{(p, x, (p * k + x) \mod n \mid 0 \leqslant p < n \text{ et } 0 \leqslant x < k\}$$

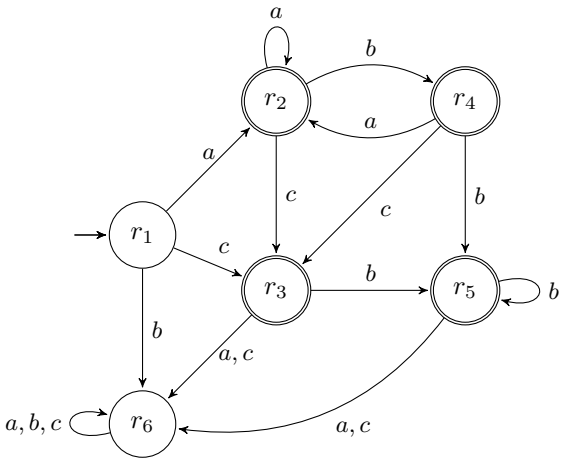
Exercice 32 [A savoir faire] Soit $Q = \{p_1, p_2, p_3, p_4\}$. On considère l'automate $A = (Q, \{a, b, c\}, \delta, \{p_1\}, \{p_3, p_4\})$, où la relation de transition δ est définie par :

δ	a	b	c	ε
p_1	p_1, p_2	-	p_3	-
p_2	-	p_1	-	p_3
p_3	-	p_3, p_4	-	-
p_4	-	p_4	-	-

Construire un automate déterministe complet équivalent à A.

Solution de l'Exercice 32. Après élimination des ε-transitions et déterminisation, on a la fonction de transition suivante :

I/F	nom	δ	a	b	c
I	r_1	p_1	p_1, p_2	$\emptyset$	p_3
F	r_2	p_1, p_2	p_1, p_2	p_1, p_3, p_4	p_3
F	r_3	p_3	$\emptyset$	p_3, p_4	$\emptyset$
F	r_4	p_1, p_3, p_4	p_1, p_2	p_3, p_4	p_3
F	r_5	p_3, p_4	$\emptyset$	p_3, p_4	$\emptyset$
	r_6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Exercice 33 Un barman aveugle portant des gants de boxe joue au jeu suivant avec l'un de ses clients réguliers : quatre verres sont disposés en carré sur un plateau circulaire pouvant pivoter autour de son centre. Au début du jeu, le client pose les verres dans le sens qu'il souhaite. Par la suite, le barman et le client jouent à tour de rôle. L'objectif du barman est de mettre tous les verres dans le même sens : soit tous à l'endroit, soit tous à l'envers. Pour cela, il peut retourner n'importe quel sous-ensemble des verres. Après chaque mouvement du barman, le client peut faire pivoter le plateau d'un ou plusieurs quarts de tour (y compris zéro ou des tours complets). Évidemment, comme le barman ne peut savoir s'il a gagné (il est aveugle et porte des gants de boxe), le client arrête le jeu lorsque le barman gagne.

La question est la suivante : le barman possède-t-il une stratégie gagnante ? Si oui, quelle est la stratégie optimale (i.e. celle avec un nombre minimum de coups) ?

▷ QUESTION 1 En utilisant les symétries du problème, déterminer quels sont les états du plateau pertinents pour le barman. À cet effet, on rappelle que le barman ne peut distinguer le sens des verres (à l'endroit ou à l'envers) ou l'orientation du plateau (que le client peut faire tourner). Faire de même pour les coups du barman.

▷ QUESTION 2 Donnez un automate A décrivant toutes les transitions possibles entre les états du plateau, en fonction des coups du barman. On ne s'occupe ici que de la fonction de transition et on fixera les états initiaux et finaux plus tard.

▷ QUESTION 3 Donnez un automate A_{client} (éventuellement non déterministe) qui donne toutes les séquences de coups du barman pour lesquels le client peut gagner (à condition de faire toujours les bons choix). Autrement dit, il faut

reprendre l'automate précédent et supprimer les états et transitions qui feraient perdre le client puis fixer les états intitiaux et finaux.

Pour trouver une stratégie gagnante pour le barman, il faut trouver une suite de coups après laquelle le client ne peut plus gagner, c'est à dire qui rendre nécessairement dans un état puits (ou un ensemble d'états puits).

▷ Question 4 Déterminiser A_{client} et en déduire la réponse à notre problème.

▷ Question 5 Que faut-il modifier si la condition de fin de partie est que tous les verres soient tous à l'endroit (et non tous à l'envers) ?

▷ Question 6 **[Avancé]** Le barman a-t-il une stratégie gagnante quel que soit le nombre de verres ?

Solution de l'Exercice 33.

▷ Question 1 Comme on s'intéresse au point de vue du barman, il faut tout d'abord représenter les différents états intéressants du jeu de son point de vue. On peut remarquer deux choses :

- **le sens exact des verres n'importe pas** : si le barman possède une stratégie gagnante, alors la même stratégie va fonctionner si les verres étaient tous retournés : toutes les configurations seront retournées et on finira donc dans une configuration où tous les verres sont dans le même sens, mais retournés.
- **les états sont invariants par rotation** : le barman n'obtient aucune information lorsque le client fait tourner le plateau et il ne peut donc différencier des états qui sont des rotations les uns des autres.

De ces deux remarques, on tire qu'il y a 4 états du jeu différenciables pour le barman. Le premier est celui où tous les verres sont dans le même sens, soit tous à l'endroit, soit tous à l'envers. Si ce n'est pas le cas, il y a au moins un verre à l'endroit et un à l'envers. Par rotation et retournement, on peut toujours supposer que le verre au nord est à l'endroit et que celui à l'ouest est à l'envers. Il reste donc 4 configurations (les sens des verres à l'est et au sud) dont deux sont équivalentes (lorsque ces deux verres sont dans le même sens).

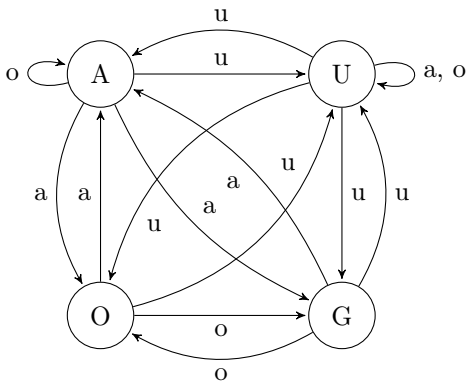
Les états du jeu sont donc :

- G (Gagné) : les verres sont tous dans le même sens,
- U (Unique) : un verre dans un sens, les trois autres dans l'autre,
- A (Adjacents) : deux verres dans chaque sens, l'un à coté de l'autre,
- O (Opposé) : deux verres dans chaque sens, à des emplacement opposés sur le plateau.

Le barman peut renverser un nombre arbitraire de verres mais seuls quatre types de mouvements peuvent faire passer d'un état à un autre ; les autres leur sont équivalents. Ce sont les mêmes que pour les états :

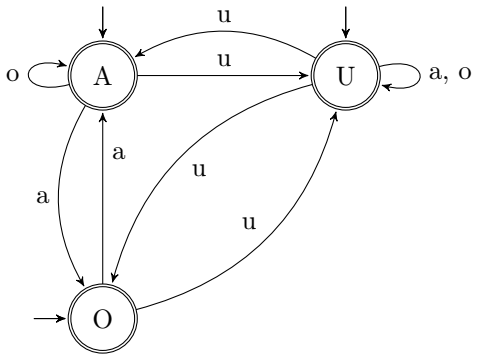
- r (rien) : on ne retourne rien (donc on reste dans le même état),
- u (unique) : on retourne un seul verre,
- a (adjacent) : on retourne deux verres adjacents,
- o : on retourne deux verres opposés.

▷ Question 2 Dans la suite, on retire les transitions r qui ne font pas progresser l'état du jeu (ce sont des boucles sur chacun des états). Alors, le système de transition [2] qui correspond est :

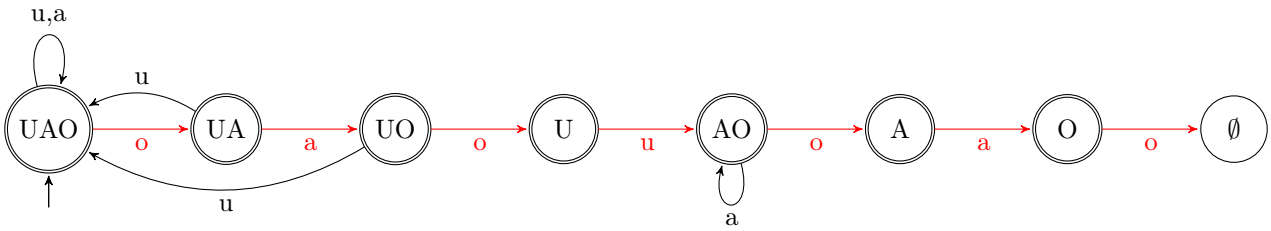


▷ Question 3 Du point de vue du client, l'état G est un état puits (il a déjà perdu !) donc on peut le retirer. De plus, comme le client choisit l'état initial du jeu, les états A, O et U sont tous initiaux. L'automate A_{client} est donc :

2. Ce n'est pas encore un automate car les états initiaux et acceptants ne sont pas définis.



▷ QUESTION 4 Si on déterminise cet automate A_{client}, on obtient :



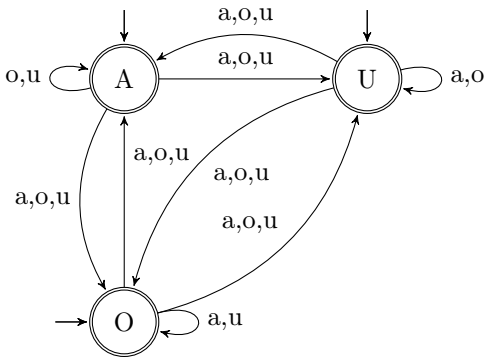
On voit en rouge une séquence de 7 transitions qui permet d'aller de l'état UAO à l'état puits ∅ : oaouoao. Il s'agit donc d'une stratégie gagnante pour le barman. On voit qu'elle est minimale directement sur l'automate car toutes les transitions font au plus progresser d'un pas vers la droite.

On peut aussi remarquer que la stratégie se décompose ainsi :
- o est une stratégie gagnante depuis O,
- a permet de transformer A en O et o laisse A invariant donc oao est une stratégie gagnante depuis O ou A (elle gagne en 1 coup si on est en O et en 3 si on est en A),
- u permet de transformer U en O ou A et a et o laissent U invariant donc oao u oao est une stratégie gagnante depuis A, U ou O.

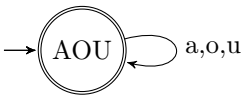
▷ QUESTION 5 Il n'est pas suffisant d'ajouter un retournement complet de tous les verres à la fin. En effet, la stratégie précédente suppose qu'on ne passe jamais pas un état où tous les verres sont dans le même sens alors qu'ici, les verres pourraient tous être à l'envers sans que le jeu ne s'arrête. Une façon simple d'adapter la stratégie précédente est de rajouter un retournement de tous les verres entre chaque coup du barman, ce qui permet d'éviter ce problème.

Il s'avère que cette stratégie est également optimale en nombre de coups (15). Pour le montrer, il faut refaire les étapes de cet exercice en retirant l'équivalence par retournement, ce qui donne 6 états et 5 coups possibles à la question 1.

▷ QUESTION 6 **[Avancé]** Ce n'est pas vrai pour tous les nombres de verres : les coups du barman peuvent laisser trop de liberté au client. Par exemple pour 5 verres, l'automate A_{client} est :



qui donne après déterminisation :



Le seul état est gagnant pour le client donc le barman n'a pas de stratégie gagnante.

Exercice 34 Pour $k > 0$, soit L_k le langage constitué des mots sur $\{0,1\}$ de longueur au moins k, et dont le $k^{\text{ième}}$ symbole **en partant de la fin** est un 1. Par exemple, 00101 et 100110111 sont dans L_3. Formellement,

$$L_k \stackrel{\text{def}}{=} \{a_1 \dots a_n \mid n \geq k \wedge a_{n-k+1} = 1\}.$$

▷ Question 1 Construire un automate (non-déterministe) à $k+1$ états qui reconnaît L_k.

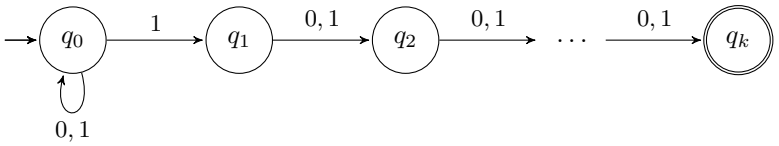
▷ Question 2 Construire un automate déterministe complet minimal reconnaissant L_2.

On cherche à borner la taille minimale d'un automate déterministe complet reconnaissant L_k. Soit $A = (Q, \{0,1\}, \delta, \{q_0\}, F)$ un automate déterministe complet reconnaissant L_k. On définit $f : \{0,1\}^k \to Q$, qui à tout mot u de longueur k associe $\delta^*(q_0, u)$. Autrement dit, $f(u)$ est l'état atteint par le chemin de trace u dans A, partant de q_0.

▷ Question 3 **[Avancé]** Montrer que f est injective. En déduire une borne inférieure de la taille de A.

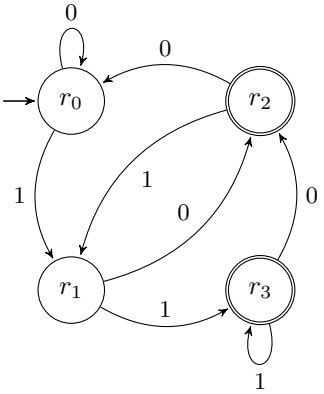
Solution de l'Exercice 34.

▷ Question 1 Automate non-déterministe :



▷ Question 2 Table de transition de l'automate déterministe :

I/F	nom	δ	0	1
I	r_0	q_0	q_0	q_0, q_1
	r_1	q_0, q_1	q_0, q_2	q_0, q_1, q_2
F	r_2	q_0, q_2	q_0	q_0, q_1
F	r_3	q_0, q_1, q_2	q_0, q_2	q_0, q_1, q_2



▷ Question 3 Supposons que f n'est pas injective. Il existe donc deux mots u et v de longueur k tels que $u \neq v$ et $f(u) = f(v)$. Notons en particulier que ceci signifie que pour tout $w \in \{0,1\}^*$, on a $\delta^*(q_0, uw) = \delta^*(\delta^*(q_0, u), w) = \delta^*(\delta^*(q_0, v), w) = \delta^*(q_0, vw)$.

Comme u et v sont distincts, leur plus grand préfixe commun w est un préfixe strict (éventuellement vide). Sans perte de généralité, on suppose que $u = w1u_1$ et $v = w0v_1$ (noter que $|u_1| = |v_1|$).

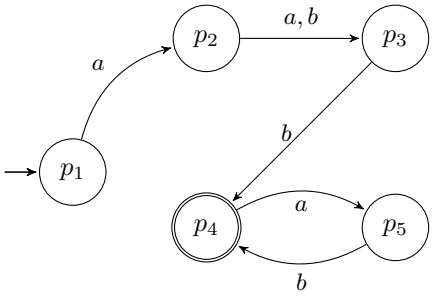
Soient $u' = uw$ et $v' = vw$. Alors $|1u_1w| = k$ (on a enlevé w au début et ajouté w à la fin par rapport à u), ce qui prouve que $u' = w1u_1w$ est élément de L_k ; donc $\delta^*(q_0, u') \in F$. Mais de la même façon, on vérifie que $v' \notin L_k$ et donc $\delta^*(q_0, v') \notin F$. Or, $\delta^*(q_0, u') = \delta^*(q_0, uw) = \delta^*(q_0, vw) = \delta^*(q_0, v')$; on a donc une contradiction. On en déduit que f est bien injective.

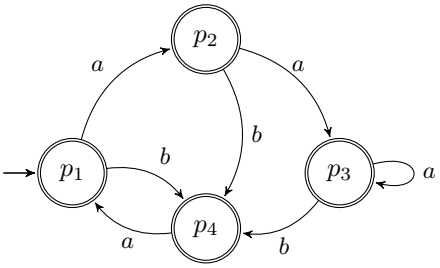
La fonction f étant injective, on en déduit que $Q \geq 2^k$ (car le cardinal de $\{0,1\}^k$ est 2^k).

Une façon intuitive de comprendre ce résultat est la suivante : pour accepter un mot de L_k, un automate déterministe doit savoir si le k^{e} symbole en partant de la fin est un 1 ou non. Du coup, il doit à tout instant se rappeler les k derniers symboles qu'il a lus, car à tout instant on peut, soit lui dire « c'est fini » et lui demander « est-ce qu'il y a k transitions tu as lu un 1 ? », soit lui dire « voici un nouveau symbole », auquel cas le $(k-1)^{\text{e}}$ en partant de la fin devient le k^{e}... La seule mémoire d'un AFD étant encodée dans ses états, vu que l'information à mémoriser consiste en k informations binaires, il faut 2^k états pour encoder cette information, et donc au moins 2^k états pour reconnaître L_k.

6 Minimisation

Exercice 35 Minimiser les automates suivants :



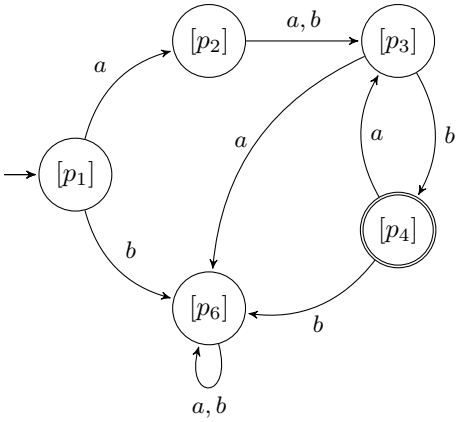


Solution de l'Exercice 35. Les automates sont déterministes mais non complets, il faut penser à ajouter un état puits.

Les premières classes (celles de $\equiv_0$) sont $Q \setminus F$ et F. Dans la suite, on détermine la classe $\equiv_{k+1}$ en fonction de la classe $\equiv_k$ et de la « signature » de chaque état, c.-à-d. des classes vers lesquelles on arrive après une transition par chacun des symboles du vocabulaire. Par exemple, la signature CD signifie qu'en lisant un a, on arrive dans la classe C et qu'en lisant un b, on arrive dans la classe D. On rappelle qu'il est inutile de traiter les classes singletons car elles ne peuvent pas diminuer d'avantage.

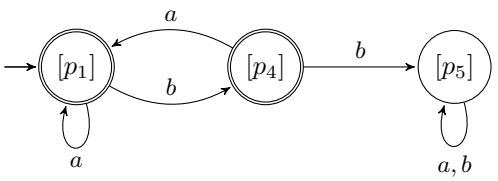
Premier automate : L'état p_6 est l'état puits qu'on a rajouté pour compléter l'automate.

$\equiv_0$	$\{p_1,$	$p_2,$	$p_3,$	p_5	$p_6\}$	$\{p_4\}$
noms des classes			A			B
« signatures »	AA	AA	AB	AB	AA	
$\equiv_1$	$\{p_1,$	$p_2,$	$p_6\}$	$\{p_3,$	$p_5\}$	$\{p_4\}$
noms des classes		C			D	B
« signatures »	CC	DD	CC	CB	CB	
$\equiv_2$	$\{p_1,$	$p_6\}$	$\{p_2\}$	$\{p_3,$	$p_5\}$	$\{p_4\}$
noms des classes		E	G		D	B
« signatures »	GE	EE		EB	EB	
$\equiv_3$	$\{p_1\}$	$\{p_6\}$	$\{p_2\}$	$\{p_3,$	$p_5\}$	$\{p_4\}$
noms des classes	H	I	G		D	B
« signatures »				IB	IB	
$\equiv_4$	$\equiv_3$					



Second automate : L'état p_5 est l'état puits, rajouté pour compléter l'automate.

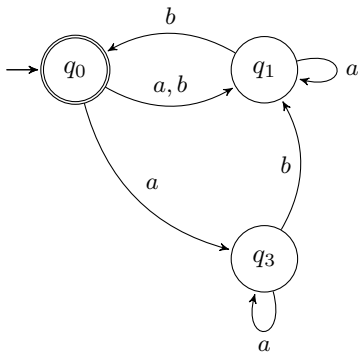
$\equiv_0$	$\{p_5\}$	$\{p_1,$	$p_2,$	$p_3,$	$p_4\}$
noms des classes	A		B		
« signatures »		BB	BB	BB	BA
$\equiv_1$	$\{p_5\}$	$\{p_1,$	$p_2,$	$p_3\}$	$\{p_4\}$
noms des classes	A		C		D
« signatures »		CD	CD	CD	
$\equiv_2$	$\equiv_1$				



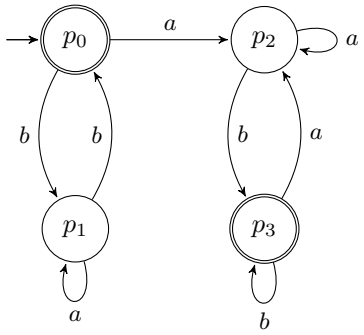
Exercice 36 Soit $Q = \{q_0, q_1, q_2, q_3, q_4\}$. Déterminiser et minimiser l'automate $A = (Q, \{a, b\}, \delta, \{q_0\}, \{q_4\})$, où la relation de transition δ est récapitulée ci-dessous :

δ	a	b	ε
q_0	q_1	$\times$	q_3, q_4
q_1	q_1	q_0	$\times$
q_2	$\times$	q_4	q_1
q_3	q_3	q_1	$\times$
q_4	$\times$	$\times$	q_3

Solution de l'Exercice 36. L'état q_2 n'est pas accessible. On le retire donc, et après élimination des ε-transitions, on a :



Après déterminisation, en posant $p_0 = \{q_0\}$, $p_1 = \{q_1\}$, $p_2 = \{q_1, q_3\}$ et $p_3 = \{q_0, q_1\}$, on obtient :



Minimisation :

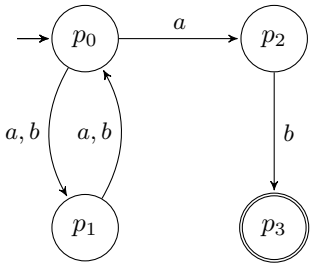
$$
\begin{aligned}
\equiv_0 &: \{p_0, p_3\}, \{p_1, p_2\} \\
\equiv_1 &: \{p_0\}, \{p_3\}, \{p_1, p_2\} \\
\equiv_2 &: \{p_0\}, \{p_3\}, \{p_1\}, \{p_2\} \\
\equiv_3 &: \equiv_2
\end{aligned}
$$

Exercice 37 [A savoir faire] Construire les automates minimaux reconnaissant les langages suivants sur $\{a, b\}$:

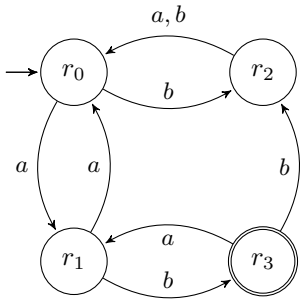
1. L'ensemble des mots de longueur paire et terminés par ab.
2. L'ensemble des mots contenant la sous-chaîne aa.
3. L'ensemble $\{a\}\{aa, bb\}^*\{a, b\}^*\{b\}$.

Solution de l'Exercice 37.

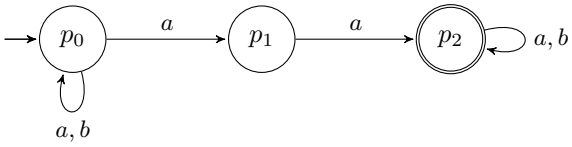
1. Automate non-déterministe :



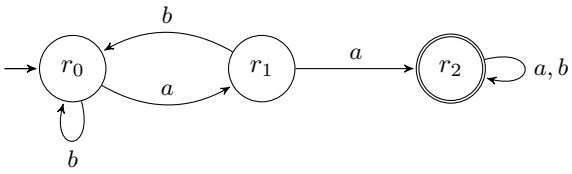
Automate minimal équivalent :



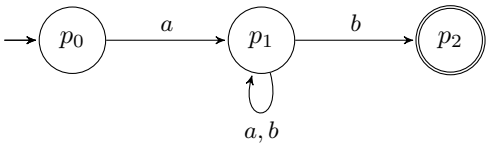
2. Automate non-déterministe :



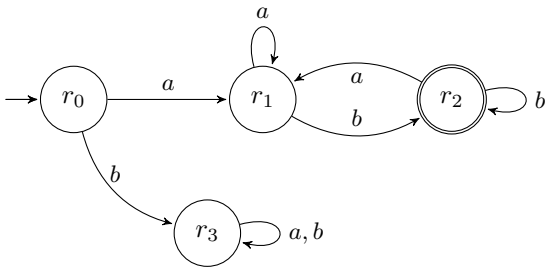
Automate minimal équivalent :



3. Pour aller plus vite, on peut remarquer que $\{a\}\{aa, bb\}^*\{a, b\}^*\{b\} = \{a\}\{a, b\}^*\{b\}$.
Automate non-déterministe :



Automate minimal équivalent :



7 Expressions régulières

Exercice 38 [A savoir faire] Soit E une expression régulière. Simplifier les expressions suivantes :

1. $E.E^* + \epsilon$
2. $\epsilon.E$
3. $\emptyset^*$
4. ϵ^*
5. $\emptyset.E$
6. $\emptyset + E$

Solution de l'Exercice 38.

1. $E.E^* + \epsilon = E^*$
2. $\epsilon.E = E$
3. $\emptyset^* = \epsilon$
4. $\epsilon^* = \epsilon$
5. $\emptyset.E = \emptyset$
6. $\emptyset + E = E$

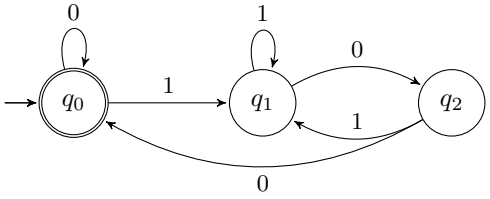
Exercice 39 Donner une expression régulière représentant chacun des langages suivants :

1. Les mots sur $\{0, 1\}$ contenant au moins un 0.
2. Les mots sur $\{0, 1\}$ de longueur paire.
3. Les mots sur $\{0, 1\}$ contenant deux 0 et/ou deux 1 consécutifs.
4. Les mots sur $\{0, 1\}$ où chaque 0 est suivi d'un 1.
5. Les mots sur $\{0, 1\}$ composés de 0 et de 1 alternés.

Solution de l'Exercice 39.

1. $(0 + 1)^*0(0 + 1)^*$, ou bien $1^*0(0 + 1)^*$.
2. $((0 + 1)(0 + 1))^*$.
3. $(0 + 1)^*(00 + 11)(0 + 1)^*$.
4. $(1 + 01)^*$.
5. $(0 + \epsilon)(10)^*(1 + \epsilon)$, ou bien $(01)^*(0 + \epsilon) + (10)^*(1 + \epsilon)$.

Exercice 40 Calculer l'expression régulière correspondant à l'automate ci-dessous, en résolvant le système d'équations obtenu dans deux ordres différents, puis en utilisant la méthode par suppression d'états dans les mêmes ordres. Constater que les systèmes associés aux automates avec certains états supprimés correspondent aux différentes étapes de résolution du système initial.



Solution de l'Exercice 40. Système d'équations associé :

$$\begin{cases} x_0 &= 0x_0 + 1x_1 + \varepsilon \\ x_1 &= 0x_2 + 1x_1 \\ x_2 &= 0x_0 + 1x_1 \end{cases}$$

— Élimination de x_2 puis x_1 :

$$\begin{cases} x_0 &= 0x_0 + 1x_1 + \varepsilon \\ x_1 &= 00x_0 + (1 + 01)x_1 \\ x_2 &= 0x_0 + 1x_1 \end{cases} \qquad \begin{cases} x_0 &= 0x_0 + 1(1 + 01)^*00x_0 + \varepsilon \\ x_1 &= (1 + 01)^*00x_0 \\ x_2 &= 0x_0 + 1x_1 \end{cases}$$

D'où

$$[0 + 1(1 + 01)^*00]^* .$$

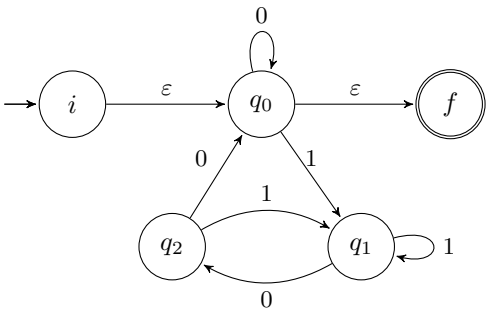
— Élimination de x_1 puis x_2 :

$$\begin{cases} x_0 &= 0x_0 + 1^+0x_2 + \varepsilon \\ x_1 &= 1^*0x_2 \\ x_2 &= 0x_0 + 1^+0x_2 \end{cases} \qquad \begin{cases} x_0 &= 0x_0 + (1^+0)^+0x_0 + \varepsilon \\ x_1 &= 1^*0x_2 \\ x_2 &= (1^+0)^*0x_0 \end{cases}$$

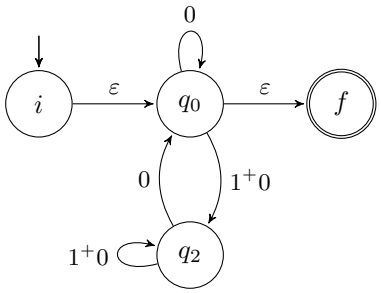
D'où

$$\left[0 + (1^+0)^+0\right]^* = \left[(1^+0)^*0\right]^* .$$

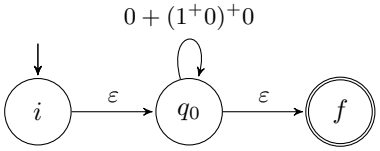
— Méthode graphique par élimination d'états, dans l'ordre q_1, puis q_2, puis q_0. On commence par modifier l'automate pour se ramener à un unique état initial sans transition entrante et un unique état acceptant sans transition sortante.



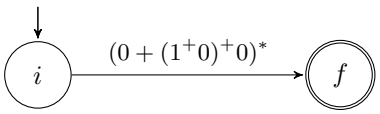
Supprimons q_1 :



Puis, supprimons q_2 :



Enfin, supprimons q_0 :

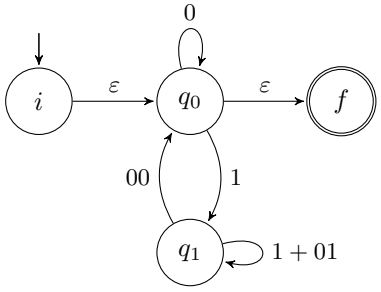


Les systèmes associés sont :

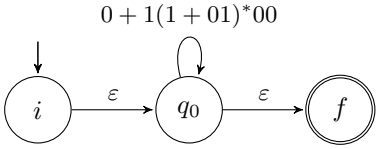
$$\begin{cases} x_i &= x_0 \\ x_0 &= 0x_0 + 1^+0x_2 + x_f \\ x_2 &= 0x_0 + 1^+0x_2 \\ x_f &= \varepsilon \end{cases} \quad \text{et} \quad \begin{cases} x_i &= x_0 \\ x_0 &= (0 + (1^+0)^+0x_2 + x_f \\ x_f &= \varepsilon \end{cases}$$

— Méthode graphique par élimination d'états, dans l'ordre q_2, puis q_1, puis q_0. Comme précéemment, on part de l'automate avec les états i et f ajoutés.

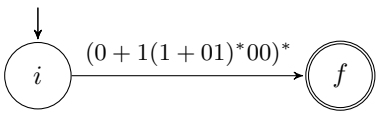
Supprimons q_2 :



Puis, supprimons q_1 :



Enfin, supprimons q_0 :



Les systèmes associés sont :

$$\begin{cases} x_i &= x_0 \\ x_0 &= 0x_0 + 1x_1 + x_f \\ x_1 &= 00x_0 + (1 + 01)x_1 \\ x_f &= \varepsilon \end{cases} \quad \text{et} \quad \begin{cases} x_i &= x_0 \\ x_0 &= (0 + 1(1 + 01)^*00)x_0 + x_f \\ x_f &= \varepsilon \end{cases}$$

On remarque que les équations de x_0, x_1 et x_2 dans les automates après suppression de certains états correspondent bien aux différentes étapes de résolution.

Exercice 41 [A savoir faire] Caractériser (par une phrase en français) les langages représentés par les expressions régulières suivantes :

1. $0^*(10^*10^*10^*)^*$
2. $(1+01+001)^*(\epsilon+0+00)$
3. $1^*(0+\epsilon)1^*$

Solution de l'Exercice 41.

1. Les mots $w \in \{0,1\}^*$ contenant un nombre de 1 multiple de 3 (c.-a-d., tels que $|w|_1 = 3k$, où $k \in \mathbb{N}$).
2. Les mots $w \in \{0,1\}^*$ qui ont au plus deux 0 consécutifs (jamais trois).
3. Les mots $w \in \{0,1\}^*$ contenant au plus un 0.

Exercice 42 [A savoir faire] Nous considérons une représentation des messages codés en Morse à l'aide du formalisme suivant. Les signaux qui peuvent être émis sont :

— le signal de début de phrase : D ;
— les signaux pour constituer des mots : L (signal long) et C (signal court)
— le signal de fin de phrase : F.

Un mot en Morse est une succession de trois signaux, longs ou courts. Une phrase en Morse est une séquence non-vide de mots, précédée du signal de début de phrase, et terminée par le signal de fin de phrase. Un message en Morse est une séquence éventuellement vide de phrases.

Donner une expression régulière décrivant l'ensemble des messages valides en Morse.

Solution de l'Exercice 42. Messages valides :

$$\Big(D\left((L+C)(L+C)(L+C)\right)^+ F\Big)^*$$

Exercice 43 [Avancé] Soit E un ensemble. Une fonction $f : \mathcal{P}(E) \to \mathcal{P}(E)$ est dite croissante, ssi pour toute partie X et Y de E, $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$. Montrer le théorème de **Knaster-Tarski** :

Soit $f : \mathcal{P}(E) \to \mathcal{P}(E)$ croissante. Il existe $S \in \mathcal{P}(E)$ tel que

1. pour tout $X \in \mathcal{P}(E)$, si $f(X) \subseteq X$ alors $S \subseteq X$;
2. $S = f(S)$.

NB : ce S est alors la *plus petite solution* de l'équation "$X = f(X)$" (S est aussi appelé *plus petit point-fixe* de f).

Indication : Soit $P = \{X \in \mathcal{P}(E) \mid f(X) \subseteq X\}$ et $S = \{e \in E \mid \forall X \in P, e \in X\}$ (on note aussi $S = \bigcap P$).
Montrer que le S ainsi défini a les propriétés attendues.

Solution de l'Exercice 43. Reprenons les définitions de l'indication.
Soit $X \in P$. Pour tout $e \in S$, on a $e \in X$. Donc $S \subseteq X$. Ceci montre 1.

Pour tout $X \in P$, comme $S \subseteq X$ et f croissante, on a : $f(S) \subseteq f(X) \subseteq X$.
Autrement dit, pour tout $e \in f(S)$, on a $e \in S$ (car pour tout $X \in P$, $e \in X$). Donc $f(S) \subseteq S$.
Comme de plus, f est monotone, on en déduit : $f(f(S)) \subseteq f(S)$, et donc $f(S) \in P$.
D'après 1, on en déduit aussi $S \subseteq f(S)$, et donc $S = f(S)$. Ceci montre 2.

Exercice 44 [Avancé] En admettant le théorème de l'exercice 43, montrer la généralisation suivante du lemme d'Arden.
Soit V un vocabulaire, A et B deux fonctions croissantes de $\mathcal{P}(V^*) \to \mathcal{P}(V^*)$.
Les équations "$X = A(X).X + B(X)$" et "$X = A(X)^*.B(X)$" admettent la même plus petite solution.

Solution de l'Exercice 44. Soient $f_1 : X \mapsto A(X).X + B(X)$ et $f_2 : X \mapsto A(X)^*.B(X)$.
Ces deux fonctions sont croissantes et donc admettent bien chacune un plus petit point-fixe d'après le théroème de Knaster-Tarski (exercice 43). Appelons respectivement S_1 et S_2 ces plus petits points-fixes.
Posons $A_1 = A(S_1)$, $B_1 = B(S_1)$, $A_2 = A(S_2)$ et $B_2 = B(S_2)$.

Comme $S_1 = A_1.S_1 + B_1$, d'après le lemme d'Arden, on a $f_2(S_1) = A_1^*.B_1 \subseteq S_1$. D'après le théorème de Knaster-Tarski pour f_2, de $f_2(S_1) \subseteq S_1$ on déduit $S_2 \subseteq S_1$.

Par ailleurs, $S_2 = A_2^*.B_2$ vérifie l'équation $S_2 = A_2.S_2 + B_2$ d'après le lemme d'Arden. Ceci peut se réécrire $S_2 = f_1(S_2)$, qui implique en particulier $f_1(S_2) \subseteq S_2$. Ainsi, d'après le théorème de Knaster-Tarski pour f_1, on a $S_1 \subseteq S_2$. D'où finalement, $S_1 = S_2$.

Exercice 45 [Avancé] Définir une méthode pour éliminer les ε-transitions et déterminiser un automate directement sur son système d'équations. Cette méthode n'utilisera que des opérations élémentaires sur les systèmes d'équations : remplacer une sous-expression (e.g. une variable) par une autre sous-expression qui lui est égale ; factoriser/distribuer des concaténations sur l'addition ; introduire une nouvelle variable satisfaisant une nouvelle équation pour nommer une certaine sous-expression ; simplifier une équation avec le lemme d'Arden (e.g. quand $A = \{\varepsilon\}$ ou $B = \emptyset$) ; etc.

Les avantages de cette technique sont les suivants : elle est simple, elle permet de faire les deux tâches en même temps, elle fusionne au passage certains états équivalents, et surtout, elle peut se généraliser au-delà des langages réguliers (e.g. aux langages LL, voir TL2).

Expliquer votre méthode sur les automates de :

1. l'exercice 24 (éliminiation des ε-transitions) ;
2. l'exercice 28 (déterminisation) ;
3. l'exercice 32 (les deux à la fois).

Indication : Montrer à l'aide du lemme d'Arden généralisé (cf. exercice 44) que la plus petite solution de $X = X + B(X)$ est aussi celle de $X = B(X)$.

Solution de l'Exercice 45. De façon générale, on va appliquer des transformations sur le système d'équations, en commençant par celle de l'état initial. On élimine au passage les équations qui ne sont plus accessibles depuis celle de l'état initial. Par convention, l'équation de l'état initial est toujours la première du système d'équation (comme pour les grammaires).

1. Pour éliminer une ε-transition, après simplification des termes « $\epsilon.X$ » en « X », on substitue dans une somme chaque terme réduit à une seule variable « X » par le membre droit de son équation, sauf si ce terme est en fait dans le membre droit de sa propre équation. Dans ce cas, d'après l'indication, on peut simplement éliminer ce terme « X » de la somme (sans faire la substitution). On itère cette transformation sur chacune des équations accessibles jusqu'à stabilité.

   Exemple sur l'exercice 24. Le système d'équation de départ est :

   $$\begin{cases} X_p &= \epsilon.X_q \\ X_q &= a.X_p + c.X_r + \epsilon.X_r \\ X_r &= \epsilon.X_s + \epsilon \\ X_s &= b.X_s + c.X_r \end{cases}$$

   Après simplification des « $\epsilon.X$ » et un premier niveau de substitution de X_q (resp. X_s) dans l'équation de X_p (resp. X_r), on obtient :

   $$\begin{cases} X_p &= a.X_p + c.X_r + X_r \\ X_r &= b.X_s + c.X_r + \epsilon \\ X_s &= b.X_s + c.X_r \end{cases}$$

   Puis finalement, en substituant X_r dans X_p (et factorisant le terme $c.X_r$) :

   $$\begin{cases} X_p &= a.X_p + b.X_s + c.X_r + \epsilon \\ X_r &= b.X_s + c.X_r + \epsilon \\ X_s &= b.X_s + c.X_r \end{cases}$$

   C'est bien le système de l'automate trouvé dans la solution de l'exercice 24.

   NB : Pour justifier que cette méthode converge, on met chaque itération sur le système d'équations en correspondance avec une itération du calcul de $\mathrm{Acc}_\varepsilon(.)$ comme plus petit point fixe. Ainsi, le nombre d'itérations avant stabilisation est borné par le nombre d'équations, c-à-d. le nombre de variables.

2. Dans chaque membre droit à traiter, après substitutions des variables « isolées » (i.e. élimination des ε-transitions), on commence par factoriser à gauche par chaque symbole du vocabulaire de façon à se rapprocher d'un membre droit « déterministe ». Ceci peut éventuellement introduire des sous-expressions sous forme de somme de variables : on introduit alors une nouvelle variable dans le système avec une nouvelle équation donnant un nom à cette somme de variables. On itère le procédé sur les nouvelles équations, jusqu'à n'avoir que des membres droits « déterministes ».

   Exemple sur l'exercice 28. Initialement, le système est :

   $$\begin{cases} P_1 &= P_2 + P_4 \\ P_2 &= a.P_2 + b.P_3 \\ P_3 &= \epsilon \\ P_4 &= a.P_5 \\ P_5 &= b.P_5 + \epsilon \end{cases}$$

Après substitions de P_2 et P_4 dans P_1, on a :

$$P_1 = a.P_2 + b.P_3 + a.P_5 = a(P_2 + P_5) + b.P_3$$

On introduit alors $R_2 = P_2 + P_5$, puis on réapplique le procédé à cette nouvelle équation, on a :

$$R_2 = a.P_2 + b.(P_3 + P_5) + \epsilon$$

On introduit $R_3 = P_3 + P_5$, puis on réapplique le procédé à cette nouvelle équation, on a :

$$R_3 = b.P_5 + \epsilon$$

On remarque ici que $R_3 = P_5$ (même membre droit). On peut donc remplacer R_3 par P_5 dans R_2. On obtient finalement le système déterministe :

$$\begin{cases} P_1 &= a.R_2 + b.P_3 \\ R_2 &= a.P_2 + b.P_5 + \epsilon \\ P_2 &= a.P_2 + b.P_3 \\ P_3 &= \epsilon \\ P_5 &= b.P_5 + \epsilon \end{cases}$$

C'est presque le système de l'automate déterministe trouvé à l'exercice 28, sauf qu'on a au passage fusionné deux états équivalents, $\{p_3, p_5\}$ et $\{p_5\}$ et que l'état puits $\emptyset$ n'a pas été introduit.

NB : Pour justifier la convergence de la méthode, il faut bien considérer chaque nouvelle variable introduite comme le nom d'une somme de variables d'origine, et ne pas introduire de variable qui correspond à une somme déjà nommée. L'argument de terminaison est alors identique à celui de l'algorithme de déterminisation de l'automate.

3. Idem sur l'exercice 32. Initialement :

$$\begin{cases} P_1 &= a.(P_1 + P_2) + c.P_3 \\ P_2 &= b.P_1 + P_3 \\ P_3 &= b.(P_3 + P_4) + \epsilon \\ P_4 &= b.P_4 + \epsilon \end{cases}$$

On déterminise P_1 avec $P_1 = a.R_2 + c.P_3$ en posant $R_2 = P_1 + P_2$, puis on transforme R_2 :

$$R_2 = a.R_2 + b.P_1 + c.P_3 + b.(P_3 + P_4) + \epsilon = a.R_2 + b.(P_1 + P_3 + P_4) + c.P_3 + \epsilon$$

On pose $R_3 = P_1 + P_3 + P_4$ qu'on transforme en :

$$R_3 = a.R_2 + b.(P_3 + P_4) + c.P_3 + \epsilon$$

On pose $R_4 = P_3 + P_4$ qu'on transforme en :

$$R_4 = b.R_4 + \epsilon$$

Ici, on remarque que $P_3 = b.(P_3 + P_4) + \epsilon = b.R_4 + \epsilon = R_4$. On peut donc remplacer R_4 par P_3 dans les équations précédentes. On obtient finalement le système déterministe :

$$\begin{cases} P_1 &= a.R_2 + c.P_3 \\ R_2 &= a.R_2 + b.R_3 + c.P_3 + \epsilon \\ R_3 &= a.R_2 + b.P_3 + c.P_3 + \epsilon \\ P_3 &= b.P_3 + \epsilon \end{cases}$$

C'est presque le système de l'automate déterministe trouvé à l'exercice 32, sauf qu'on a au passage fusionné les deux états équivalents r_3 et r_5 et qu'on n'introduit pas l'état puits r_6.

Exercice 46 On considère les *expressions régulières étendues*, qui sont obtenues en ajoutant aux expressions régulières les constructions suivantes :

— si E est une E.R., alors $\neg E$ est une E.R. étendue ;
— si E et E' sont des E.R., alors $E \cap E'$ est une E.R. étendue ;
— si E est une E.R., alors E^+ est une E.R. étendue.

La sémantique de ces opérateurs est la suivante :

— $\mathcal{L}(\neg E) = V^* \setminus \mathcal{L}(E)$;
— $\mathcal{L}(E \cap E') = \mathcal{L}(E) \cap \mathcal{L}(E')$;
— $\mathcal{L}(E^+) = \mathcal{L}(E).\mathcal{L}(E)^*$.

Démontrer qu'à toute E.R. étendue est associé un langage régulier.

Solution de l'Exercice 46. La démonstration se fait par induction structurelle sur les E.R. étendues. La propriété démontrée est « pour toute E.R. étendue E, $\mathcal{L}(E)$ est régulier ».

Cas E.R non étendues On peut reprendre la démonstration vue en cours.

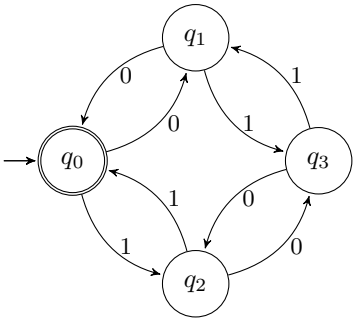
Cas E^+ Par hypothèse d'induction, $\mathcal{L}(E)$ est régulier. Comme $E^+ = E.E^*$ et que les langages réguliers sont clos par concaténation et concaténation itérée, le résultat est immédiat.

Cas $\neg E$ Par hypothèse d'induction, $\mathcal{L}(E)$ est régulier. On a vu au CM7 que les langages réguliers sont clos par complémentation, donc $V^* \setminus \mathcal{L}(E) = \mathcal{L}(\neg E)$ est un langage régulier.

Cas $E_1 \cap E_2$ Par hypothèses d'induction, $\mathcal{L}(E_1)$ et $\mathcal{L}(E_2)$ sont réguliers. On a vu au CM7 que les langages réguliers sont clos par intersection, donc $\mathcal{L}(E_1) \cap \mathcal{L}(E_2) = \mathcal{L}(E_1 \cap E_2)$ est un langage régulier.

Exercice 47 [A savoir faire] Donner une expression régulière représentant l'ensemble des mots avec un nombre pair de 0 et un nombre impair de 1, en définissant un automate reconnaissant ce langage et en résolvant les équations associées.

Solution de l'Exercice 47. On peut obtenir cet automate comme l'automate produit d'un automate qui reconnait les mots avec un nombre pair de 1 et celui qui reconnait les mots avec un nombre pairs de 0 (voir exercice 20).



Système d'équations associé :

$$\begin{cases} x_0 &= 0x_1 + 1x_2 + \varepsilon \\ x_1 &= 0x_0 + 1x_3 \\ x_2 &= 0x_3 + 1x_0 \\ x_3 &= 0x_2 + 1x_1 \end{cases}$$

On se sert de la régularité de l'automate pour ne pas avoir une expression régulière trop complexe. Élimination de x_1, x_2 puis x_3 :

$$\begin{cases} x_0 &= 00x_0 + 01x_3 + 1x_2 + \varepsilon \\ x_1 &= 0x_0 + 1x_3 \\ x_2 &= 0x_3 + 1x_0 \\ x_3 &= 0x_2 + 10x_0 + 11x_3 \end{cases} \quad \begin{cases} x_0 &= (00+11)x_0 + (01+10)x_3 + \varepsilon \\ x_1 &= 0x_0 + 1x_3 \\ x_2 &= 0x_3 + 1x_0 \\ x_3 &= (00+11)x_3 + (01+10)x_0 \end{cases}$$

Et au final, on obtient :

$$\begin{aligned} x_3 &= (00+11)^*(01+10)x_0 \\ x_0 &= [00 + 11 + (01+10)(00+11)^*(01+10)]^* \end{aligned}$$

Exercice 48 [Avancé] Etant donnée une expression régulière E, on définit la *hauteur d'étoile de* E comme le nombre d'étoiles de Kleene imbriquées dans E. Par exemple, $\mathrm{H}_K(a) = \mathrm{H}_K((a+b).(c+d)) = 0$, $\mathrm{H}_K(a^*) = \mathrm{H}_K(ab^*(a+c)^*) = 1$ et $\mathrm{H}_K((ab^*c)^*) = 2$.

▷ QUESTION 1 Définir formellement la fonction H_K sur l'ensemble des expressions régulières par induction structurelle.

La notion de hauteur d'étoile est étendue aux langages réguliers : si L est un langage régulier, alors $\mathrm{H}_K(L)$ est défini par :

$$\mathrm{H}_K(L) \;=\; \min\{\mathrm{H}_K(E) \,|\, \mathcal{L}(E) = L\}.$$

▷ QUESTION 2 Soit $E = a(a^*b^*)^*bb$. Quelle est la valeur de $\mathrm{H}_K(E)$? Quelle est la valeur de $\mathrm{H}_K(\mathcal{L}(E))$?

▷ QUESTION 3 Soit L un langage régulier. Démontrer que L est fini si et seulement si $\mathrm{H}_K(L) = 0$.

Solution de l'Exercice 48.

▷ QUESTION 1

$$\begin{aligned} \mathrm{H}_K(\emptyset) &= 0 \\ \mathrm{H}_K(\epsilon) &= 0 \\ \mathrm{H}_K(a) &= 0, \text{ où } a \in V \\ \mathrm{H}_K(E.E') &= \max(\mathrm{H}_K(E), \mathrm{H}_K(E')) \\ \mathrm{H}_K(E + E') &= \max(\mathrm{H}_K(E), \mathrm{H}_K(E')) \\ \mathrm{H}_K(E^*) &= 1 + \mathrm{H}_K(E) \end{aligned}$$

▷ QUESTION 2 On a $\mathrm{H}_K(E) = 2$. Comme E est équivalent [3] à $a(a+b)^*bb$, on a $\mathrm{H}_K(\mathcal{L}(E)) = 1$.

▷ QUESTION 3 Supposons que L est fini. Ce langage peut alors clairement être représenté par l'expression régulière $E \stackrel{\text{def}}{=} \sum_{w \in L} w$ (ou $E \stackrel{\text{def}}{=} \emptyset$ si $L = \emptyset$), donc, $\mathrm{H}_K(L) = 0$. Réciproquement, supposons que $\mathrm{H}_K(L) = 0$, et soit E une expression régulière telle que $\mathcal{L}(E) = L$. On a alors $\mathrm{H}_K(E) = 0$ donc E ne peut pas contenir d'occurrence de l'opérateur *, et une induction structurelle prouve que toute expression régulière sans occurrence de cet opérateur représente un langage fini.

8 Propriétés de clôture

Exercice 49 Etant donné un vocabulaire V, on considère une famille $(L_i)_{i \in \mathbb{N}}$ de langages sur V^* telle que pour tout $i \in \mathbb{N}$, L_i est un langage régulier.

▷ QUESTION 1 Soit $n \geq 0$. Montrer que le langage $M_n = \bigcup_{0 \leq i \leq n} L_i$ est régulier.

▷ QUESTION 2 Peut-on en déduire que $\bigcup_{0 \leq i} L_i$ est régulier ? Justifier.

Solution de l'Exercice 49.

▷ QUESTION 1 Le résultat se prouve par récurrence sur n. Pour $n = 0$ on a $M_0 = L_0$ qui est régulier d'après l'énoncé. Pour $n > 0$ on a $M_n = \bigcup_{0 \leq i \leq n} L_i = \left(\bigcup_{0 \leq i \leq n-1} L_i\right) \cup L_n$. Par HR, $\bigcup_{0 \leq i \leq n-1} L_i$ est régulier, et L_n est régulier d'après l'énoncé, or l'union de langages réguliers est un langage régulier (vu en cours) donc M_n est un langage régulier.

▷ QUESTION 2 Prenons la famille $(L_i)_{i \in \mathbb{N}}$ où pour tout $i \in \mathbb{N}$, $L_i \stackrel{\text{def}}{=} \{a^i b^i\}$. Chacun de ces langages est un singleton et est donc régulier, mais l'union de tous ces langages est $\{a^n b^n \mid n > 0\}$, non régulier. La réponse est donc non.

Exercice 50 On admet que $M = \{a^n b^n \mid n \geq 0\}$ n'est pas régulier. Montrer que les langages suivants ne sont pas réguliers non plus **sans se servir du lemme de l'étoile** :

1. $L_1 = \{w \in \{a,b\}^* \mid w \text{ a autant de } a \text{ que de } b\}$
2. $L_2 = \{a^i b^j c^k \mid i + j = k \geq 0\}$
3. $L_3 = \{(ab)^{2n}(cd)^{2n} \mid n \geq 0\}$
4. **[Avancé]** $L_4 = \{uv \in \{a,b\}^* \mid vu \in \{a^n b^n \mid n \geq 0\}\}$

Solution de l'Exercice 50.

1. On a $M = L_1 \cap a^*b^*$, donc L_1 n'est pas régulier.
2. Soit h la fonction définie par :

$$h : \begin{cases} a & \mapsto & a \\ b & \mapsto & a \\ c & \mapsto & b \end{cases}$$

C'est un homomorphisme, et on a $h(L_2) = M$ donc L_2 n'est pas régulier.

3. **Exercice :** prouver que $(a^*b^*)^* = (a+b)^*$.

3. **Remarque :** On ne voit plus la fermeture par homomorphisme inverse en cours (théorème 5.2.12 du polycopoié sur Chamilo, page 66).

   Soit h la fonction définie par :

$$h : \begin{cases} a & \mapsto & a \\ b & \mapsto & \varepsilon \\ c & \mapsto & b \\ d & \mapsto & \varepsilon \end{cases}$$

   On a $h(L_3) = \{a^{2n}b^{2n} \mid n \geq 0\}$; ce langage est régulier si L_3 est régulier. Donc, si L_3 est régulier, alors $h(L_3) \cup \{a\}.h(L_3).\{b\} = M$ est également régulier, une contradiction.
4. Soit $L'_4 = L_4 \cap b^*a^*$, et posons

$$h : \begin{cases} a & \mapsto & b \\ b & \mapsto & a \end{cases}$$

   Alors $h(L'_4) = M$; le langage L_4 ne peut pas être régulier.

   Autre méthode : Posons $L''_4 = L_4 \cap a^*b^*$. Montrons que $L''_4 = M$. Soit $w \in L''_4$, donc $w \in L_4$, c.-à-d. $w = uv$ avec $vu \in M$. Supposons $u \neq \varepsilon$ et $v \neq \varepsilon$. Alors, comme $vu \in M$, v commence par un a et u termine par un b. De ce fait, uv contient le sous-terme ba (à la frontière entre u et v) donc ne peut être dans L''_4, une contradiction. Ainsi, on a $u = \varepsilon$ ou $v = \varepsilon$ et on en déduit $w \in M$. Au final, $L''_4 = M$ d'où on tire que L_4 n'est pas régulier.

Exercice 51 En utilisant le lemme de l'étoile, montrer que les langages suivants ne sont pas réguliers :

1. $L_1 = \{wb^n \mid n \in \mathbb{N}, w \in \{a,b\}^n\}$
2. $L_2 = \{w \in \{a,b\}^* \mid w \text{ est un palindrome}\}$
3. **[Avancé]** $L_3 = \{1^{i^2} \mid i \geq 0\}$
4. **[Avancé]** $L_4 = \{1^p \mid p \text{ est premier}\}$

Solution de l'Exercice 51. D'après le lemme de l'étoile, si L est un langage régulier, alors il existe un entier n tel que si $z \in L$ est de longueur au moins n, alors z est de la forme uvw, où $|uv| \leq n$, $|v| \geq 1$ et pour tout $i \geq 0$, $uv^iw \in L$.

1. Soit n l'entier donné par le lemme de l'étoile appliqué à L_1. Prenons le mot $z = a^nb^n$; alors z est de la forme uvw, et comme $1 \leq |uv| \leq n$, on a $uv \in a^+$. Donc $v \in a^+$, et on devrait avoir $uv^2w = a^{n+|v|}b^n \in L_1$, ce qui est impossible.
2. Soit n l'entier donné par le lemme de l'étoile appliqué à L_2. Soit $z = a^nba^n$. Ce mot est un palindrome de longueur au moins n, il est donc de la forme uvw, et nécessairement, $v \in a^+$. Mais on a $uv^0w = a^{n-|v|}ba^n$ qui n'est pas un palindrome, une contradiction.
3. Soit $z = 1^{n^2}$; z est de la forme uvw. On pose $z' = uv^2w$. Comme $uv \leq n$, on en déduit que $|z'| \leq n^2+n < (n+1)^2$. Comme $v \neq \varepsilon$, on a aussi $n^2 < |z'|$, donc z' ne peut pas être élément de L_3.
4. Soit $z = 1^p$, où p est un nombre premier tel que $p \geq n+2$ (un tel nombre premier existe nécessairement puisqu'il y en a une infinité). z est de la forme uvw. Comme $|uv| \leq n$, on a $|w| \geq 2$, et donc $|uw| \geq 2$. Comme $|v| \geq 1$, on a également $1 + |v| \geq 2$.

   Posons $z' = uv^{|uw|}w$. On a $|z'| = |u| + |uw|\,|v| + |w| = |uw| + |uw|\,|v| = |uw|(1 + |v|)$. Les deux facteurs du produit sont ≥ 2, donc $|z'|$ n'est pas premier ; on a une contradiction.

Exercice 52 Equivalence entre automates

▷ QUESTION 1 Donner une méthode algorithmique pour déterminer si deux automates sont équivalents.

▷ QUESTION 2 Si deux automates ne sont pas équivalents, comment faire pour exhiber un contre-exemple, c.-à-d. un mot accepté par l'un mais pas par l'autre ?

▷ QUESTION 3 En se basant sur la construction de l'automate produit (exercice 20), comment faire cette construction plus directement ?

Solution de l'Exercice 52.

▷ QUESTION 1 Pour tester l'équivalence entre automates, on peut les déterminiser (au besoin) puis les minimiser. Par unicité de l'automate minimal (au renommage des états près), on peut alors facilement déterminer s'ils sont équivalents.

▷ Question 2 Notons A_1 et A_2 les deux automates pour lesquels on souhaite trouver un contre-exemple à leur équivalence. Pour trouver un mot accepté par l'un et non par l'autre, on peut s'intéresser aux langages $\mathcal{L}(A_1)\setminus\mathcal{L}(A_2) = \mathcal{L}(A_1)\cap\overline{\mathcal{L}(A_2)}$ et $\mathcal{L}(A_2)\setminus\mathcal{L}(A_1)$. Comme les langages réguliers sont clos par complémentaire, intersection et union, on peut déterminer un automate qui reconnaît $\mathcal{L}(A_1)\cap\overline{\mathcal{L}(A_2)}$ et de même pour $\mathcal{L}(A_2)\setminus\mathcal{L}(A_1)$. Pour trouver un mot reconnu par cet automate, il suffit alors de faire un parcours de graphe.

▷ Question 3 Il suffit de choisir $F = F_1\times(Q_2\setminus F_2)\cup F_2\times(Q_1\setminus F_1)$ pour reconnaître la différence symétrique. Attention, les automates doivent être déterministes pour cela (même problème que pour la négation) !

Remarque : On pourrait penser que cette méthode permet de décider l'équivalence entre automates sans avoir besoin de déterminiser mais il n'en est rien.

Exercice 53 [A savoir faire] Les langages suivants sont-ils réguliers ?

1. $L_1 = \{a^n b^m \mid n \neq m\}$.
2. $L_2 = \{w \in \{a,b\}^* \mid |w|_a \neq |w|_b\}$.

Solution de l'Exercice 53.

1. Si L_1 est régulier, alors $\overline{L_1}\cap a^*b^* = \{a^nb^n \mid n\geq 0\}$ l'est également, ce qui est impossible.
2. Si L_2 est régulier, alors $L_2\cap a^*b^* = L_1$ l'est également, ce qui est impossible.

Exercice 54 [A savoir faire] Description des commentaires en C et OCaml
Un commentaire du langage C commence par `/*` et se termine au premier `*/` rencontré. Ainsi, il ne peut y avoir de `*/` au milieu d'un commentaire.
Exemples : les mots `/**/`, `/***/`, `/****/`, `/*/*/` sont des commentaires mais `/*/` et `/***/*/` n'en sont pas.

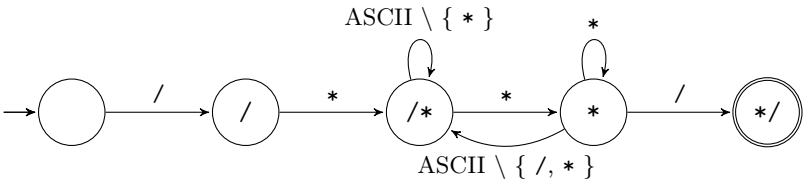
▷ Question 1 Le langage des commentaires en C est-il régulier ? Si oui, donner un automate qui le reconnaît. Si non, le démontrer et donner une grammaire hors-contexte qui le génère.

Un commentaire du langage OCaml commence par `(*` et se termine par `*)`. Contrairement à C, il est possible d'imbriquer des commentaires. Exemples : les mots `(**)`, `(***)`, `(*(**)*)`, `(*)*)` sont des commentaires mais `(*)`, `(***(**)` et `(***)*)` n'en sont pas.

▷ Question 2 Mêmes questions que pour le langage C.

Solution de l'Exercice 54.

▷ Question 1 Le langage des commentaires C est régulier car une fois dans un commentaire, on recherche simplement le motif `*/` pour en sortir. Fixons le vocabulaire à l'ensemble des caractères ASCII. Voici un automate qui reconnaît les commentaires C :



▷ Question 2 Pour correctement refermer des commentaires imbriqués, il faut connaître le nombre de commentaires ouverts donc savoir compter. Le langage de commentaires OCaml n'est donc intuitivement pas régulier. Montrons-le.

On ajoute deux symboles terminaux au vocabulaire : $V' = \text{ASCII}\cup\{\mathbf{OC},\mathbf{FC}\}$ (**OC** pour « Ouverture Commentaire » et **FC** pour « Fermeture Commentaire »). Soit L le langage sur V' des commentaires OCaml où les ouvertures et fermetures de commentaires ont été remplacées par **OC** et **FC**. Ce langage n'est pas régulier car avec la subtitution régulière $\sigma : V' \to \{a,b\}$

$$\sigma \stackrel{\text{def}}{=} \begin{cases} \mathbf{OC} & \mapsto a \\ \mathbf{FC} & \mapsto b \\ \text{ASCII} & \mapsto \varepsilon \end{cases}$$

on a $\sigma(L)\cap a^*b^* = \{a^nb^n \mid n\in\mathbb{N}\}$ qu'on sait être non-régulier.
De plus, l'image de L par l'homomorphisme $h : V' \to \text{ASCII}$ qui remplace **OC** par `(*` et **FC** par `*)` est exactement le langage des commentaires OCaml. Comme l'image inverse par un homorphisme préserve la régularité, le langage des commentaires OCaml n'est pas régulier.

Voici une grammaire hors-contexte qui l'engendre :

$$\begin{aligned} S &\to (*X*) \\ X &\to TSX \mid T \mid \varepsilon \\ T &\to (\text{ASCII} \setminus \{(,*\})T \mid (T' \mid *T'' \mid \varepsilon \\ T' &\to (\text{ASCII} \setminus \{(,*\})T \mid (T' \mid \varepsilon \\ T'' &\to (\text{ASCII} \setminus \{(,*,)\})T \mid (T' \mid *T'' \mid \varepsilon \end{aligned}$$

Ici, T représente le langage des textes arbitraires sans ouverture ni fermeture de commentaire.

9 Grammaires et Hiérarchie de Chomsky

Exercice 55 Langages hors-contextes et langages réguliers

▷ Question 1 Donner des grammaires hors-contextes engendrant les langages suivants :

1. $\{a^n b^p \mid n \geq p \geq 0\}$
2. $\{a^n b^p \mid n \neq p\}$
3. $\{a^n b^p \mid 2p \geq n \geq p\}$
4. $\{a^n b^p c^q \mid n + p = q\}$

▷ Question 2 Donner des grammaires régulières engendrant les langages suivants :

1. les mots sur $\{a, b\}$ ayant un nombre pair de a et impair de b
2. **[A savoir faire]** l'ensemble des constantes entières sans 0 inutiles en tête

Solution de l'Exercice 55.

▷ Question 1

1. Plusieurs grammaires sont possibles et correctes :

$$S \to aS \mid aSb \mid \varepsilon$$

$$\begin{aligned} S &\to aS \mid A \\ A &\to aAb \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S &\to aSb \mid A \\ A &\to aA \mid \varepsilon \end{aligned}$$

   Les deuxième et troisième fixent l'ordre entre les règles, alors que le première autorise les entrelacements. De ce fait, la première grammaire est ambiguë au contraire des deux autres.
2. On décompose $\{a^n b^p \mid n \neq p\}$ en $\{a^n b^p \mid n > p \geq 0\} \cup \{a^n b^p \mid p > n \geq 0\}$ pour se ramener à une variante de la question précédente :

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to aS_1 \mid aS_1b \mid a \\ S_2 &\to S_2b \mid aS_2b \mid b \end{aligned}$$

3. La définition signifie que le nombre de a est compris entre le nombre de b et deux fois le nombre de b. Autrement dit, pour chaque b, il y a un ou deux a, ce qui donne naturellement la grammaire suivante :

$$S \to aSb \mid aaSb \mid \varepsilon$$

   Si on veut une grammaire non ambiguë, on peut prendre à la place :

$$\begin{aligned} S &\to aaSb \mid A \\ A &\to aAb \mid \varepsilon \end{aligned}$$

4. La définition signifie qu'à chaque fois qu'on ajoute un a ou un b, il faut également ajouter un c. La difficulté est d'ordonner les règles : une fois qu'on a commencé à ajouter des b, on ne doit plus ajouter de a. On peut imposer cet ordre avec deux non-terminaux différents : le premier pour ajouter les a, puis on passe au second pour ajouter les b et s'arrêter.

$$\begin{aligned} S &\to aSc \mid A \\ A &\to bAc \mid \varepsilon \end{aligned}$$

▷ QUESTION 2 Il peut être plus facile de commencer par créer un automate avant de le transformer en grammaire.

1. Quatre non-terminaux qui correspondent à la parité du nombre de a et de b des mots engendrés par ces non-terminaux : PP, PI, IP, II avec axiome PI.

$$\begin{aligned} PI &\to aII \mid bPP \\ PP &\to aIP \mid bPI \mid \varepsilon \\ II &\to aPI \mid bIP \\ IP &\to aPP \mid bII \end{aligned}$$

   Si on commence par écrire un automate, le sens des états n'est pas le même : ce serait la parité des mots *déja générés*, l'axiome serait PP et l'état acceptant (donc celui produisant ε) serait PI.

2. Pour éviter les 0 inutiles en tête, il faut séparer l'entier 0.

$$\begin{aligned} S &\to 0 \mid CA \\ C &\to 1 \mid \ldots \mid 9 \\ A &\to 0A \mid CA \mid \varepsilon \end{aligned}$$

   Cette grammaire n'est pas régulière à cause des règles qui donnent CA. Pour éviter cela, il faut substituer C par toutes ses possibilités :

$$\begin{aligned} S &\to 0 \mid 1A \mid \ldots \mid 9A \\ A &\to 0A \mid 1A \mid \ldots \mid 9A \mid \varepsilon \end{aligned}$$

Exercice 56 Opérations sur les langages

Soient $G_1 = (V_T, V_{N_1}, S_1, R_1)$ et $G_2 = (V_T, V_{N_2}, S_2, R_2)$ deux grammaires. On supposera sans perte de généralité que $V_{N_1} \cap V_{N_2} = \emptyset$.

▷ QUESTION 1 Donner une grammaire G telle que $\mathcal{L}(G) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. Comment prouver que cette grammaire est correcte ?

▷ QUESTION 2 Même question pour les langages $\mathcal{L}(G_1)\,.\,\mathcal{L}(G_2)$ et $\mathcal{L}(G_1)^*$.

▷ QUESTION 3 Appliquer vos transformations sur les grammaires G_1 et G_2 suivantes :

$$\begin{aligned} G1 &: S1 \to aS_1b \mid c \\ G2 &: S2 \to d \end{aligned}$$

▷ QUESTION 4 En supposant que G_1 et G_2 soient de type T (régulière, hors-contexte) que peut-on dire des grammaires proposées aux deux premières questions ?

Solution de l'Exercice 56.

▷ QUESTION 1 Ajouter un nouvel axiome S et les règles $S \to S_1 \mid S_2$.
Formellement : $G_1 \cup G_2 = (V_T, V_{N_1} \cup V_{N_2} \cup \{S\}, S, R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\})$.

Pour la correction, on cherche à montrer $\mathcal{L}(G) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.

$$\begin{aligned} \mathcal{L}(G) &= \{w \in V_T^* \mid S \Longrightarrow^* w\} \\ &= \{w \in V_T^* \mid S_1 \Longrightarrow^* w \text{ ou } S_2 \Longrightarrow^* w\} \\ &= \{w \in V_T^* \mid S_1 \Longrightarrow^* w\} \cup \{w \in V_T^* \mid S_2 \Longrightarrow^* w\} \\ &= \mathcal{L}(G_1) \cup \mathcal{L}(G_2) \end{aligned}$$

▷ QUESTION 2 Pour $\mathcal{L}(G_1).\mathcal{L}(G_2)$, ajouter un nouvel axiome S et la règle $S \to S_1S_2$.
Pour $\mathcal{L}(G_1)^*$, ajouter un nouvel axiome S et les règles $S \to S_1S \mid \varepsilon$.

Les preuves de correction sont vues en cours et nécessitent le théorème de décomposition des dérivations.

▷ QUESTION 3 Facile à appliquer. Cet exemple permet de vérifier que les transformations précédentes sont correctes. En particulier, il est nécessaire d'introduire un nouveau symbole non-terminal pour l'axiome afin d'éviter des interactions non désirées entre G_1 et G_2 (cf. constructions des automates pour $\mathcal{L}(A_1) \cup \mathcal{L}(A_2)$ et $\mathcal{L}(A_1).\mathcal{L}(A_2)$) et au sein de $\mathcal{L}(G_1)$ (cf. construction de l'automate pour $\mathcal{L}(A_1)^*$).

▷ QUESTION 4 Les règles ajoutées sont hors-contexte donc les grammaires hors-contexte sont préservées mais pas les grammaires régulières. Néanmoins, comme les langages réguliers sont clos par union, concaténation et concaténation itérée, il est possible de trouver des grammaires régulières. Une méthode est de transformer les grammaires en automates, d'utiliser les transformations d'automates puis de revenir aux grammaires.

Exercice 57 Langages sous-contextes

▷ Question 1 Soit la grammaire $G = (\{a, b, c\}, \{S, B, C\}, S, R)$ avec R l'ensemble des règles suivantes :

$$\begin{array}{llll} (1) & S & \to & abc \\ (3) & cB & \to & Bc \end{array} \qquad \begin{array}{llll} (2) & S & \to & aSBc \\ (4) & bB & \to & bb \end{array}$$

a) Justifier le type de cette grammaire.
b) Construire une dérivation du mot $aabbcc$.
c) Soit un mot quelconque de la forme $a^n b^n c^n$ avec $n > 0$. Donner une méthode générale permettant de produire ce mot à partir de la grammaire précédente.

▷ Question 2 **[Avancé]** Donner une grammaire sous-contexte engendrant les mots de la forme wcw avec $w \in \{a, b\}^*$. On pourra partir de la grammaire suivante, qui engendre les mots de la forme $wc\widetilde{w}$ avec $\widetilde{w}$ l'image miroir de w :

$$S \to aSa \mid bSb \mid c$$

Solution de l'Exercice 57.

▷ Question 1 a) La grammaire est de type sous-contexte car la partie droite de chaque règle a une taille au moins égale à sa partie gauche. Elle n'est pas hors-contexte à cause des règles (3) et (4) dont la partie gauche contient deux symboles.

b) $\underline{S} \overset{(2)}{\Longrightarrow} a\underline{S}Bc \overset{(1)}{\Longrightarrow} aabc\underline{B}c \overset{(3)}{\Longrightarrow} aab\underline{B}cc \overset{(4)}{\Longrightarrow} aabbcc$

c) $S \overset{(2)}{\Longrightarrow}{}^{n-1} a^{n-1}S(Bc)^{n-1} \overset{(1)}{\Longrightarrow} a^{n-1}abc(Bc)^{n-1} = a^n b(cB)^{n-1}c$

$\overset{(3)}{\Longrightarrow}{}^{\frac{n(n-1)}{2} = \sum_{i=1}^{n-1} i} a^n bB^{n-1}c^{n-1}c \overset{(4)}{\Longrightarrow}{}^{n-1} a^n bb^{n-1}c^{n-1}c = a^n b^n c^n$

▷ Question 2 Idée : on adapte la grammaire pour $wc\widetilde{w}$ en $wc\widetilde{W}$ où W est le mot w en majuscule :

$$S \to aSA \mid bSB \mid c$$

Ensuite, il faut retourner le mot $\widetilde{W}$ en w. Pour éviter des échanges anarchiques ou sans savoir qui doit aller où, on ordonne les échanges et on marque les symboles qui se déplacent. On marque (par la minuscule) les symboles à déplacer vers la droite en partant de c :

$$cA \to ca \qquad cB \to cb$$

puis on déplace ces symboles vers la droite en traversant les symboles en capitale :

$$aA \to Aa \qquad aB \to Ba \qquad bA \to Ab \qquad bB \to Bb$$

Exercice 58 Langage des carrés

Soit $V_T = \{0, 1\}$. Le langage W des mots de la forme ww n'est pas hors-contexte mais on peut montrer que son complémentaire C l'est.

▷ Question 1 Soit Y le langage des mots sur V_T de longueur impaire dont le milieu est 0. Soit Z le langage des mots sur V_T de longueur impaire dont le milieu est 1. Donner une grammaire hors-contexte pour chacun de ces languages.

▷ Question 2 Montrer que tout mot de $YZ \cup ZY$ n'est pas de la forme ww.

▷ Question 3 Montrer que tout mot de longueur paire qui n'est pas de la forme ww appartient à $YZ \cup ZY$. En déduire une grammaire pour C.

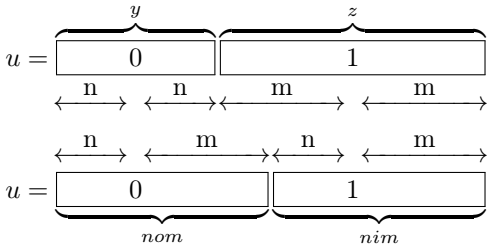
Solution de l'Exercice 58.

▷ Question 1 Deux solutions équivalentes, suivant qu'on factorise « 0 ou 1 » ou non.

$$\begin{aligned} Y &\to 0 \mid 0Y0 \mid 0Y1 \mid 1Y0 \mid 1Y1 \\ Z &\to 1 \mid 0Z0 \mid 0Z1 \mid 1Z0 \mid 1Z1 \end{aligned}$$

$$\begin{aligned} Y &\to 0 \mid BYB \\ Z &\to 1 \mid BZB \\ B &\to 0 \mid 1 \end{aligned}$$

▷ QUESTION 2 Soit u dans YZ. On a donc $u = yz$ avec $y \in Y$ et $z \in Z$.

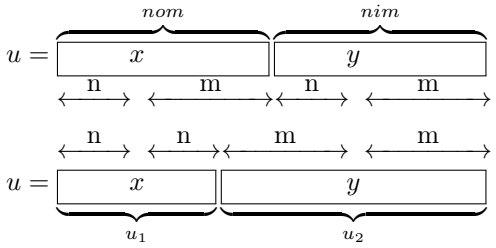


Il est clair que $|nom| = |nim| = n + m + 1$ et que $nom \neq nim$ (ils diffèrent en position $n + 1$). Donc $u = nom.nim$ n'est pas de la forme ww.

Idem pour $u \in ZY$.

▷ QUESTION 3 Il suffit de faire le raisonnement précédent à l'envers !

Soit u de longueur paire avec $u \neq ww$. Alors u peut s'écrire $nom.nim$ avec $|nom| = |nim|$ et $nom \neq nim$. Comme $|nom| = |nim|$ et $nom \neq nim$, il existe un indice $k \leqslant |nom|$ tel que $nom[k] \neq nim[k]$, donc (en posant $n \stackrel{\text{def}}{=} k - 1$) il existe un indice $n < |nom|$ tel que $nom[n + 1] \neq nim[n + 1]$.



avec $x \neq y$

Suivant la valeur de x (0 ou 1), on a alors $u_1 \in Y$ et $u_2 \in Z$ ou l'inverse.
Ainsi, tout mot de C de longueur paire appartient à $YZ \cup ZY$.

Comme C contient aussi tous les mots de longueur impaire, on obtient la grammaire suivante :

$$\begin{aligned} S &\to YZ \mid ZY \mid Y \mid Z \\ Y &\to 0 \mid BYB \\ Z &\to 1 \mid BZB \\ B &\to 0 \mid 1 \end{aligned}$$

Exercice 59 Forme réduite d'une grammaire

Soit $G = (V_T, V_N, S, R)$ une grammaire hors-contexte. On donne les définitions suivantes :

— un symbole A de V_N est dit *productif* si et seulement si il existe une dérivation $A \Longrightarrow^* w$ avec $w \in V_T^*$;
— un symbole A de V_N est dit *accessible* si et seulement si il existe une dérivation $S \Longrightarrow^* w_1 A w_2$ avec $w_1, w_2 \in (V_T \cup V_N)^*$.

▷ QUESTION 1 Soit la grammaire $G = (\{a, b, c, d\}, \{S, A, B, C, D\}, S, R)$ avec R défini par :

$$\begin{aligned} S &\to AB & S &\to \varepsilon & A &\to aA & A &\to D \\ B &\to bB & B &\to aS & C &\to c & C &\to cC & D &\to dA \end{aligned}$$

Donner l'ensemble Pr des symboles productifs de G ainsi que l'ensemble Ac des symboles accessibles.

▷ QUESTION 2 Soit $G = (V_T, V_N, S, R)$ une grammaire hors-contexte quelconque.

1. Donner une définition inductive de l'ensemble Ac des non-terminaux accessibles dans G. En déduire une méthode permettant de calculer Ac à partir des règles de G.
2. Même question pour l'ensemble Pr des non-terminaux productifs.

3. En utilisant les résultats précédents, comment peut-on décider simplement si $\mathcal{L}(G) \neq \emptyset$?

▷ QUESTION 3 L'algorithme de « nettoyage » des grammaires est le suivant :

1. On calcule Pr, l'ensemble des symboles productifs de la grammaire G, puis on construit $G' = (V_T, Pr, S, R')$ avec R' le sous-ensemble des règles de R ne contenant aucune occurrence d'un symbole non productif, ni en partie gauche, ni en partie droite : $R' = R - \{A \rightarrow w_1 B w_2 \mid A \notin Pr \vee B \notin Pr\}$.
2. On calcule Ac, l'ensemble des symboles accessibles de G', puis on construit $G'' = (V_T, Ac, S, R'')$ avec R'' le sous-ensemble des règles de R' ne contenant aucune occurrence d'un symbole non accessible, ni en partie gauche, ni en partie droite : $R'' = R' - \{A \rightarrow w_1 B w_2 \mid A \notin Ac \vee B \notin Ac\}$.

Appliquer cet algorithme à la grammaire $G = (\{a, b\}, \{S, A, C, D, E\}, S, R)$ pour R défini par :

$$S \rightarrow A \qquad S \rightarrow a \qquad A \rightarrow CD \qquad C \rightarrow b \qquad D \rightarrow A \qquad E \rightarrow C$$

▷ QUESTION 4 En utilisant l'exemple précédent, montrer que l'algorithme naïf qui consisterait à construire la grammaire G' ci-après est faux (i.e. ne produit pas une grammaire réduite).

$G' = (V_T, Ac \cap Pr, S, R')$ avec R' le sous-ensemble des règles de R ne contenant aucune occurrence d'un symbole non accessible ou productif, ni en partie gauche, ni en partie droite : $R' = R - \{A \rightarrow w_1 B w_2 \mid A \notin Ac \vee B \notin Ac \vee A \notin Pr \vee B \notin Pr\}$.

Solution de l'Exercice 59.

▷ QUESTION 1 $Pr = \{S, B, C\}$ et $Ac = \{S, A, B, D\}$

▷ QUESTION 2

1. La définition inductive des non-terminaux accessibles est :

$$A_0 = \{S\}$$
$$A_{i+1} = A_i \cup \{X \mid (Y \rightarrow \alpha X \beta) \in R \text{ avec } Y \in A_i \text{ et } \alpha, \beta \in (V_T \cup V_N)^*\}$$

Notez que cette définition s'apparente au calcul de l'ensemble des états accessibles par ε-transitions dans un automate. Ici, on initialise avec l'axiome et on itère la procédure suivante jusqu'à stabilisation : pour chaque règle, si le non-terminal en partie gauche est accessible, alors tout non-terminal dans la partie droite l'est également.

2. La définition inductive des non-terminaux productifs est :

$$P_0 = \{X \mid (X \rightarrow w) \in R \text{ avec } w \in V_T^*\}$$
$$P_{i+1} = P_i \cup \{X \mid (X \rightarrow \alpha_1 \dots \alpha_n) \in R \text{ avec } \forall k, \alpha_k \in V_T \vee \alpha_k \in P_i)\}$$

Pour les non-terminaux productifs, on commence par ajouter les non-terminaux possédant une règle qui ne produit que des terminaux. Ensuite, on *remonte* les règles qui produisent des non-terminaux : si tous les non-terminaux de la partie droite d'une règle sont productifs, le non-terminal de la partie gauche l'est également.

3. Il suffit que l'axiome soit productif. Attention : dire qu'il existe un non-terminal X qui est à la fois accessible et productif n'est pas suffisant : dans la suite de règle qui mène de l'axiome à X, il pourrait y avoir des non-terminaux non productifs (cf. question 4) et X ne serait pas accessible dans la grammaire réduite.

▷ QUESTION 3 $Pr(G) = \{S, C, E\}$ donc $G' = (\{a, b\}, \{S, C, E\}, S, \{S \rightarrow a;\ E \rightarrow C;\ C \rightarrow b\})$.
$Ac(G') = \{S\}$ donc $G'' = (\{a, b\}, \{S\}, S, \{S \rightarrow a\})$.

▷ QUESTION 4 L'algorithme naïf sur l'exemple précédent donne $(\{a, b\}, \{S, C\}, S, \{S \rightarrow a;\ C \rightarrow b\})$. On constate que C reste présent car il est à la fois accessible est productif, même s'il n'est pas accessible dans la grammaire réduite. Le problème est que le chemin qui rend C accessible passe par un non-terminal qui n'est pas productif donc est supprimé. En commençant par retirer les symboles non-productifs avant de s'intéresser aux symboles accessibles, on évite ce problème car un état non-productif ne peut plus être utilisé pour rendre un état accessible.

Exercice 60 Grammaires régulières

On donne ci-après plusieurs manières de décrire les grammaires régulières. Soit $G = (V_T, V_N, S, R)$ une grammaire avec les restrictions suivantes :

Def1 : règles de la forme $A \rightarrow w$ ou $A \rightarrow wB$ avec $w \in V_T^*$, $A \in V_N$ et $B \in V_N$.
Def2 : règles de la forme $A \rightarrow xB$ ou $A \rightarrow \varepsilon$ avec $x \in V_T$ et $A \in V_N$ et $B \in V_N$.

▷ QUESTION 1 Soit L le langage défini sur le vocabulaire $\{a, b\}$ par $(ab)^*$. Une grammaire possible pour ce langage conforme à **Def1** est $S \rightarrow abS \mid \varepsilon$. Donner une grammaire conforme à **Def2** pour ce langage.

▷ QUESTION 2 Montrer que la classe des langages définie par ces deux formes de grammaire est la même.

Solution de l'Exercice 60.

▷ Question 1 $S \to aS' \mid \varepsilon \qquad S' \to bS$

▷ Question 2 **Def2** $\Longrightarrow$ **Def1** : Les règles de la **Def2** sont des cas particuliers de celles de la **Def1**. Ainsi, on a directement l'implication réciproque.

Def1 $\Longrightarrow$ **Def2** : Soit G une grammaire suivant **Def1**. Il faut construire une grammaire équivalente G' suivant **Def2**. Pour cela, on va traduire chaque règle de type **Def1** en un ensemble de règle de type **Def2**.

- $A \to w$: On effectue la construction par induction structurelle sur w (ou de façon équivalente, par récurrence sur la longueur de w).
  - Si $w = \varepsilon$, $A \to \varepsilon$ est déjà de type **Def2**
  - Si $w = xw'$ avec $x \in V_T$ et $w' \in V_T^*$, on introduit un nouveau non-terminal A' et les règles $A \to xA'$ et $A' \to w'$. Ensuite, on décompose récursivement la seconde règle.
- $A \to wB$: On effectue la construction par récurrence sur la longueur de w.
  - Si $w = \varepsilon$, on remplace la règle $A \to B$ par les règles $A \to \alpha$ pour $(B \to \alpha) \in R$. Autrement dit, on fusionne les étapes $A \Longrightarrow B \Longrightarrow \alpha$ en une seule règle. Au besoin, on recommence récursivement : si $(B \to C) \in R$ pour un non-terminal C, on obtient alors $A \to C$ et il faut recommencer. Si on tombe sur une règle $X \to X$, on la supprime simplement.
  - Si $w = x \in V_T$, la règle $A \to xB$ est déjà de type **Def2**.
  - Si $w = xw'$ avec $x \in V_T$ et $w' \in V_T^+$, on introduit un nouveau non-terminal A' et les règles $A \to xA'$ et $A' \to w'B$. Ensuite, on décompose récursivement la seconde règle.

10 Grammaires hors-contexte

Exercice 61 Grammaire ambiguë

▷ Question 1 Montrer que les grammaires suivantes sont ambiguës et proposer des grammaires équivalentes non ambiguës :

1. $S \to aSaS \qquad S \to \varepsilon$
2. $S \to aSb \qquad S \to aS \qquad S \to \varepsilon$

▷ Question 2 Donner une grammaire non ambiguë pour le langage $\{a^n b^p \mid 2p \geq n \geq p\}$. Montrer que votre grammaire est non-ambiguë en appliquant les conditions suffisantes vues en cours.

▷ Question 3 **[Avancé]** Une ε-règle est une règle de la forme $A \to \varepsilon$.

Une 1-règle est une règle de la forme $A \to B$ avec A et B éléments du vocabulaire non-terminal.

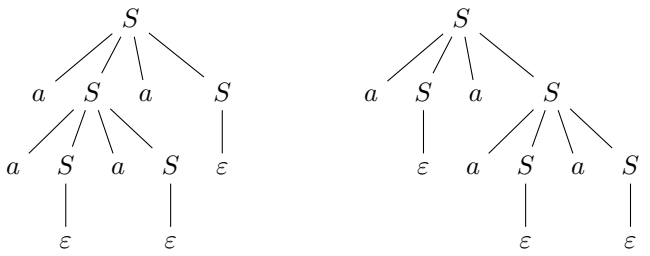
Soit G une grammaire hors-contexte ne contenant ni ε-règle ni 1-règle.

Pour tout mot $\omega \in \mathcal{L}(A)$, avec $A \in V_N$, on peut borner la longueur de la dérivation $A \Longrightarrow^* \omega$ en fonction de $|\omega|$. Soit $A \Longrightarrow^d \omega$. Montrer que $d \leq 2 * |w| - 1$.

Solution de l'Exercice 61.

▷ Question 1 1. Pour $S \to aSaS \mid \varepsilon$:

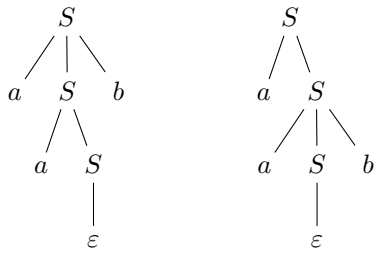
Le mot $aaaa$ peut être généré par deux arbres de dérivation différents :



Le langage engendré par cette grammaire est $(aa)^*$ donc on peut proposer $S \to aaS \mid \varepsilon$ ou $S \to aSa \mid \varepsilon$. Si on veut une grammaire régulière : $S \to aX \mid \varepsilon \qquad X \to aS$.

2. Pour $S \to aSb \qquad S \to aS \qquad s \to \varepsilon$:

Le mot aab possède deux arbres de dérivation :



Pour obtenir un grammaire équivalente non-ambiguë, il faut ordonner les règles. Deux solutions :
— $S \to aS \mid A \qquad A \to aAb \mid \varepsilon$
— $S \to aSb \mid A \qquad A \to aA \mid \varepsilon$

▷ QUESTION 2 On commence par une grammaire ambiguë (cf. exercice 55) : $S \to aaSb \mid aSb \mid \varepsilon$. Pour lever l'ambiguïté, il faut ordonner les règles qui posent problème, donc choisir si on commence par deux a pour un b ou un seul. Cela revient à décomposer le langage comme suit : $\{a^{2p}a^qb^qb^p \mid p,q \geq 0\}$ ou $\{a^pa^{2q}b^qb^p \mid p,q \geq 0\}$. Voici les grammaires non-ambiguës qui correspondent :

$$S \to aaSb \mid X \qquad X \to aXb \mid \varepsilon$$

$$S \to aSb \mid X \qquad X \to aaXb \mid \varepsilon$$

Pour montrer que ces grammaires sont non-ambiguës, on utilise les deux conditons suffisantes vues en cours :
— pour tout couple de règles $(A \to \alpha, A \to \beta)$ de G tel que $\alpha \neq \beta$, $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) = \emptyset$;
Autrement dit, à un point donné d'une dérivation, on ne peut remplacer une règle par une autre et obtenir le même mot (cf. cas 2 de la question 1 de l'exercice 61).
— pour toute règle de la forme $A \to X_1X_2...X_n$, où $X_i \in V_T \cup V_N$, $\forall w \in V_T^*$ tel que $X_1X_2...X_n \Longrightarrow^* w$, $\exists!(w_1, w_2, ...w_n)$ tel que $w_i \in V_T^*$, $w = w_1w_2...w_n$ et $\forall i$, $X_i \Longrightarrow^* w_i$.
Autrement dit, lorsqu'on a appliqué une règle, les mots qui doivent être générés par chacun des non-terminaux sont uniques (cf. cas 1 de la question 1 de l'exercice 61).

Utilisons ces conditions sur la première grammaire.

Première condition : Pour les règles $X \to AXb$ et $X \to \varepsilon$, la longueur permet de les distinguer : $\mathcal{L}(\varepsilon) = \varepsilon$ qui est de longueur 0 alors tout $w \in \mathcal{L}(aSb)$ est de longueur ≥ 2.
Pour les règles $S \to aaSb$ et $S \to X$, il faut remarquer que tout mot issu de X contient autant de a que de b (on a même $\mathcal{L}(X) = \{a^nb^n \mid n \in \mathbb{N}\}$) alors que ce n'est pas le cas d'un mot issu de $aaSb$ car un mot $w \in \mathcal{L}(S)$ vérifie $|w|_a \geq |w|_b$.

Seconde condition : Comme les parties droites des règles ne contiennent qu'un unique symbole non-terminal, les preuves sont triviales. En effet, tout symbole terminal ne peut dériver qu'en lui-même donc le reste doit l'être par le symbole non-terminal.

▷ QUESTION 3 Comme la grammaire G ne contient ni ε-règle ni 1-règle, les parties droites de règles ou bien contiennent au moins deux symboles ou bien font exactement un symbole terminal.

Démonstration 1 : Pour une dérivation $A \Longrightarrow^d w$, on va montrer $d \leq 2|w| - 1$ par récurrence forte sur d. Considérons la première règle utilisée dans la dérivation $A \Longrightarrow^d w$.
— $A \to x$ avec $x \in V_T$: Alors $d = |w| = 1$ et on a bien $d \leq 2|w| - 1$.
— $A \to \alpha$ avec $\alpha \in (V_N \cup V_T)^*$ et $|\alpha| \geq 2$: Le théorème de décomposition des dérivations donne pour tout i entre 1 et $|\alpha|$, $\alpha_i \Longrightarrow^{d_i} w_i$ avec $w = w_1w_2 \dots w_{|\alpha|}$ et $d = 1 + d_1 + \cdots + d_{|\alpha|}$. On rappelle que par convention tout mot dérive vers lui-même en 0 étape donc cette écriture est également valable lorsque α_i est terminal en prenant $d_i = 0$. Ainsi, en utilisant l'hypothèse d'induction sur les dérivations $\alpha_i \Longrightarrow^{d_i} w_i$, on obtient :

$$d = d_1 + \cdots + d_{|\alpha|} + 1 \leq (2|w_1| - 1) + \cdots + (2|w_{|\alpha|}| - 1) + 1 = 2(|w_1| + \cdots + |w_{|\alpha|}|) + 1 - |\alpha| = 2|w| + 1 - |\alpha|$$

Or $|\alpha| \geq 2$ donc $1 - |\alpha| \leq -1$ et on obtient le résultat.

Démonstration 2 : On considère l'arbre de dérivation de la dérivation $A \Longrightarrow^d w$. Les règles de la forme $X \to x$ avec $x \in V_T$ ne peuvent se produire que pour les feuilles de l'arbre final donc elles sont utilisées au plus $|w|$ fois. Les autres règles possèdent au moins deux fils (mais potentiellement plus). L'utilisation de l'une de ces règles remplace une feuille (le père) par ses fils (au moins deux) donc fait augmenter le nombre de feuilles de l'arbre en construction. Chacune de ces feuilles donne au moins une lettre dans le mot final donc on peut appliquer ces règles au plus $|w| - 1$ fois (on commence avec une feuille : l'axiome). Au total, on a bien au plus $|w| + (|w| - 1)$ étapes dans la dérivation $A \Longrightarrow^d w$.

Exercice 62 Preuves sur les grammaires

On définit deux langages L_1 et L_2 sur le vocabulaire $V = \{a, b\}$.

Soit $P_1(w)$ la propriété $|w|_a = |w|_b$ et $P_2(w)$ la propriété $\forall u \in Prefixe(w), |u|_a \geq |u|_b$. Rappelons que la notation $|w|_x$ représente le nombre d'occurrences du symbole x dans w et u est un préfixe de w si et seulement il existe un mot $v \in V^*$ tel que $uv = w$.

— **Définition de L_1 par compréhension :** $L_1 = \{w \mid P_1(w) \wedge P_2(w)\}$
— **Définition de L_2 par grammaire :** $S \to \varepsilon$, $S \to SS$ et $S \to aSb$ avec $L_2 = \mathcal{L}(S)$.

▷ Question 1 Justifier en quoi la chaîne $aabbab$ appartient à la fois à L_1 et L_2.

▷ Question 2 On veut montrer que $L_2 \subseteq L_1$. On rappelle que ceci revient à montrer que tout mot dérivable à partir de S vérifie les propriétés $P1$ et $P2$.

1. ajouter une règle de grammaire qui violerait la propriété $P2$.
2. en considérant la grammaire initiale prouvez $L_2 \subseteq L_1$.

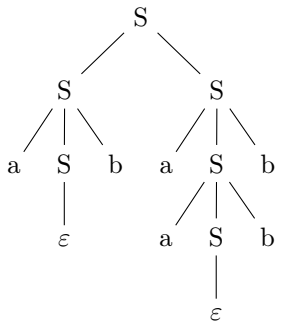
▷ Question 3 On veut maintenant montrer que $L_1 \subseteq L_2$. Pour cela on doit montrer que tout mot vérifiant les propriétés $P1$ et $P2$ peut être dérivé de l'axiome.

1. On vous propose de faire l'analyse par cas suivante : (1) $w = \varepsilon$, (2) $w = abu$, (3) $w = uab$ (4) $w = aub$, avec $u \in L_1$. Justifier en quoi cette décomposition n'est pas complète, i.e. qu'il existe des mots de L_1 qui ne peuvent être produits comme une combinaison de ces différents cas.
2. Prouvez $L_1 \subseteq L_2$.

Solution de l'Exercice 62.

▷ Question 1 Pour L_1 :
— (P_1) $|abaabb|_a = 3 = |abaabb|_b$.
— (P_2) Ses sept préfixes ont au moins autant de a que de b. Donc $abaabb \in L_1$.
Pour L_2, deux options : écrire une dérivation ou un arbre de dérivation

$$
\begin{aligned}
S &\xrightarrow{S \to SS} SS \\
&\xrightarrow{S \to aSb} aSbS \\
&\xrightarrow{S \to \varepsilon} abS \\
&\xrightarrow{S \to aSb} abaSb \\
&\xrightarrow{S \to aSb} abaaSbb \\
&\xrightarrow{S \to \varepsilon} abaabb
\end{aligned}
$$



▷ Question 2
1. Plusieurs possibilités :
   — $S \to b$ permet de produire b qui viole à la fois P_1 et P_2 ;
   — $S \to ba$ permet de produire ba qui satisfait P_1 mais pas P_2 ;
   — $S \to a$ permet de produire a qui satisfait P_2 mais pas P_1.
2. Il faut montrer que si $S \Longrightarrow^* w$ alors w satisfait P_1 et P_2.
   De $w \in V_T^*$, on déduit $w \neq S$ donc qu'il existe $n > 0$ tel que $S \Longrightarrow^n w$, d'où $S \Longrightarrow \alpha \Longrightarrow^{n-1} w$ avec $S \to \alpha \in R$.
   On procède donc par récurrence sur n, la longueur de la dérivation.

   $n = 1$: Une possibilité : $S \Longrightarrow \varepsilon\ (= w)$
   ε satisfait P_1 et P_2 (il est son seul préfixe) donc $\varepsilon \in L_1$

   $n > 1$: Deux possibilités :
   — $S \Longrightarrow aSb \Longrightarrow^{n-1} aw_1b\ (= w)$ avec $S \Longrightarrow^{n-1} w_1$
   — $S \Longrightarrow SS \Longrightarrow^{n-1} w_1w_2\ (= w)$ avec $S \Longrightarrow^p w_1$ et $S \Longrightarrow^q w_2$ et $p + q = n - 1$
   L'hypothèse d'induction (HI) à utiliser est donc : $\forall k < n,\ S \Longrightarrow^k w \Rightarrow w \in L_1$.

   — $S \Longrightarrow aSb \Longrightarrow^{n-1} aw_1b\ (= w)$ avec $S \Longrightarrow^{n-1} w_1$
   Par HI, $w_1 \in L_1$ et donc, pour $w = aw_1b$:
   (a) $|w|_a = |w_1|_a + 1 = |w_1|_b + 1 = |w|_b$
   (b) u préfixe de w est, soit ε, soit aw_1b, soit au_1 avec u_1 préfixe de w_1. On conclut facilement en analysant ces trois cas (car $|u_1|_a \geq |u_1|_b$).

— $S \Longrightarrow SS \Longrightarrow^{n-1} w_1w_2$ $(= w)$ avec $S \Longrightarrow^{p} w_1$, $S \Longrightarrow^{q} w_2$, $p+q = n-1$ donc $p < n$, $q < n$
Par HI, w_1 et w_2 sont dans L_1 et donc, pour $w = w_1w_2$:

(a) $|w|_a = |w_1|_a + |w_2|_a = |w_1|_b + |w_2|_b = |w|_b$

(b) pour u préfixe de w :
— si u préfixe de w_1 : OK
— si $u = w_1u_2$ avec u_2 préfixe de w_2 : OK (car $|w_1|_a = |w_1|_b$)

▷ QUESTION 3 1. Voici un exemple de mot de L_1 qui n'est pas couvert par la décomposition proposée : *aabbaabb*.

2. Soit $w \in L_1$. Notons $n = |w|$ et supposons que pour tout mot $x \in L_1$ de longueur $< n$, on a $x \in L_2$. Montrons que $w \in L_2$ (i.e. $S \Longrightarrow^* w$).
Pour cela distinguons trois cas :

(a) $w = \varepsilon$ (qui $\in L_1$) : La règle $S \to \varepsilon$ assure que $w \in L_2$.

(b) Il existe $(x_1, x_2) \in {L_1}^2$ tel que $w = x_1x_2$, $x_1 \neq \varepsilon$ et $x_2 \neq \varepsilon$. On a $|x_1| < n$ et $|x_2| < n$ donc par HI $S \Longrightarrow^* x_1$ et $S \Longrightarrow^* x_2$. La règle $S \to SS$ finit de démontrer que $w \in L_2$: $S \Longrightarrow SS \Longrightarrow^* x_1S \Longrightarrow^* x_1x_2 = w$.

(c) Sinon : on n'est dans aucun des deux cas précédents, on a donc forcément $\forall x_1 \in Prefixe(w)\setminus\{\varepsilon, w\}, |x_1|_a > |x_1|_b$, sans quoi un tel x_1 (avec donc $|x_1|_a = |x_1|_b$) donnerait la décomposition $w = x_1x_2$ du second cas.

Du coup, on a forcément $w = axb$, avec $|x|_a = |x|_b$ (car $|w|_a = |w|_b$) et par ailleurs $\forall u \in Prefixe(x), au \in Prefixe(w)\setminus\{\varepsilon, w\}$, donc $|au|_a > |au|_b$ et donc $|u|_a \geq |u|_b$. Par conséquent $x \in L_1$, et donc par HI, $S \Longrightarrow^* x$. La règle $S \to aSb$ finit de démontrer que $w \in L_2$: $S \Longrightarrow aSb \Longrightarrow^* axb = w$.

On a ainsi prouvé que $L_1 \subseteq L_2$, ce qui donne au final $L_1 = L_2$.

Exercice 63 Décrire les langages de programmation

▷ QUESTION 1 Dans un langage de programmation, un *identificateur* est un nom choisi par le programmeur qui peut être utilisé pour une variable ou une fonction, par exemples « x », « toto », « fibo ». Écrire une grammaire décrivant les identificateurs Python V2 (non-terminal *Idf*) sur le vocabulaire $V_1 = \{\mathbf{a}, \ldots, \mathbf{z}, \mathbf{A}, \ldots, \mathbf{Z}, \mathbf{0}, \ldots, \mathbf{9}, \_\}$ (à partir de la version 3 d'autres caractères sont permis). On rappelle qu'un identificateur ne peut jamais commencer par un chiffre.

En Python l'instruction d'affectation est de la forme : $Inst \to Cible = Exp$. En partie gauche d'une affectation, on peut trouver (entre autre) les éléments suivants :
— un identificateur (ex : `x = 1`),
— l'accès à l'attribut d'un objet (ex : `o.x = 0`),
— l'accès à un élément d'une liste (ex : `l[i+1] = 0`)
Exemples : $x[y.z]$ $x.y[2]$ $x[3].y$

▷ QUESTION 2 Toute cible est une expression. Il existe par contre des expressions qui ne sont pas des cibles (ne peuvent apparaître en partie gauche d'affectation). Donner des exemples d'expression qui ne sont pas des cibles.

▷ QUESTION 3 Soit V_2 le vocabulaire obtenu à partir de V_1 en ajoutant les symboles point (**.**), crochet ouvrant (**[**), crochet fermant (**]**) et virgule (**,**) (pour la question suivante).

Écrire une grammaire décrivant la catégorie syntaxique *Cible*. On utilisera le non-terminal *Idf* défini à la question 1 ainsi que le non-terminal *Exp* (une expression quelconque), à ne pas définir mais qui contient *Cible*.

Donner l'arbre de dérivation associé à la cible $x.y[2]$.

La grammaire proposée est-elle ambigüe ? Si oui, en donner une non-ambigüe.

▷ QUESTION 4 En fait une cible peut aussi être une liste de cibles. Cette liste doit être non vide, commence par un crochet ouvrant et finit par un crochet fermant. Les cibles sont séparées par une virgule et la dernière occurrence de cible peut, ou non, être suivie d'une virgule. Compléter la grammaire pour prendre en compte cette définition de cible.

Solution de l'Exercice 63.

▷ QUESTION 1

Idf	→	début suite-idf
début	→	**a** | ... | **z** | **A** | ... | **Z** | **_**
suite-idf	→	ε | symb suite-idf
symb	→	début | **0** | ... | **9**

Note : C'est un langage régulier (cf. doc Python 2).

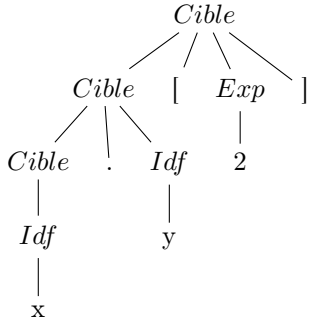
▷ QUESTION 2 Voici quelques exemples : x+1, x**3, 42, ...
Voir la documentation de Python 2 sur l'instruction d'affectation.

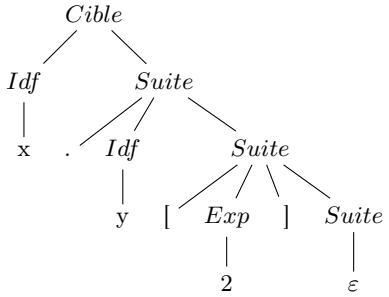
▷ Question 3 $Cible \rightarrow Idf \mid Cible\ .\ Idf \mid Cible\ [\ Exp\ ]$
La grammaire devient ambiguë si on prend $Cible\ .\ Cible$ plutôt que $Cible\ .\ Idf$.

ou bien

$Cible \rightarrow$ Idf $Suite$
$Suite \rightarrow .\ Idf\ Suite \mid [\ Exp\ ]\ Suite \mid \varepsilon$

Exemple : x.y[2]





On préfère en général la solution 1 mais la solution 2 a aussi ses avantages (*cf.* TL2).

▷ Question 4 $Cible \rightarrow \ldots \mid [\ Liste\text{-}Cible\ ]$
$Liste\text{-}Cible \rightarrow Cible\ ,\ Liste\text{-}Cible \mid Cible \mid Cible\ ,$

Exercice 64 [A savoir faire] Conversion entre grammaire régulière et automate

Soit la grammaire $G = (\{a, b, c\}, \{S, A, B, C\}, S, R)$ où R contient les règles suivantes :

$$S \rightarrow aS \mid aA \mid cB \qquad B \rightarrow bB \mid bC \mid \varepsilon$$
$$A \rightarrow B \mid bS \qquad C \rightarrow bC \mid \varepsilon$$

Construire un automate déterministe reconnaissant $\mathcal{L}(G)$.

Solution de l'Exercice 64. Lors de la transformation d'une grammaire régulière en automate, chaque symbole non-terminal devient un état et chaque règle devient une transition. Cela s'apparente à l'opposé de la construction du système d'équation correspondant à un automate.

Ici, on obtient les automates non-déterministe puis déterministe suivants :