Théorie des Langages

Recueil d'exercices

1 Langages et point fixe

Exercice 1 Donner des formules logiques pour les énoncés suivants. On pourra supposer qu'il existe des relations « posséder » et « avoir accès » entre un utilisateur et un fichier ou un dossier.

— La fonction f est une surjection.
— La fonction f est une injection.
— Un utilisateur a accès à tous les fichiers.
— Un dossier est possédé par tous les utilisateurs.
— Tous les utilisateurs possèdent un dossier.
— Il existe un unique entier n tel que $P(n)$.

Exercice 2 [A savoir faire] Soit L un langage. Montrer que les propositions suivantes sont équivalentes :

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

Exercice 3 Soit V un vocabulaire. Étant 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 $|x| \geqslant |y|$.

▷ **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 conjugaison est une relation d'équivalence.

Exercice 4 [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).

Exercice 5 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.

Exercice 6 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. $(L \cup M)^* = (L^*M^*)^*$

Exercice 7 On s'intéresse aux symboles qui commencent les mots d'une famille de langages $(L_i)_{1 \leqslant i \leqslant n}$ sur un alphabet V lorsque la famille (L_i) est définie comme le plus petit point fixe d'un système d'équations contenant des concaténations et des unions. On pose donc :

$$\mathrm{Prem}(L) = \{x \in V \mid \exists w \in V^*,\ x.w \in L\}$$

On souhaite calculer les $\mathrm{Prem}(L_i)$ par commutation de plus petit point fixe.

▷ **Question 1.** Sous quelles conditions peut-on appliquer le lemme de commutation ?

▷ **Question 2.** Donner les valeurs de $\mathrm{Prem}(\{\varepsilon\})$ et $\mathrm{Prem}(\{y\})$ pour $y \in V$. Exprimer $\mathrm{Prem}(L_1.L_2)$ et $\mathrm{Prem}(L_1 \cup L_2)$ en fonction de $\mathrm{Prem}(L_1)$ et $\mathrm{Prem}(L_2)$. Vous pourrez utiliser la fonction $\varepsilon(L) = L \cap \{\varepsilon\}$ vue en cours.
En déduire une méthode pour calculer $\mathrm{Prem}(L)$ à l'aide du lemme de commutation.

▷ **Question 3.** Appliquer la méthode de la question précédente au système suivant sur le vocabulaire $V = \{a, b, c, x\}$:

$$\begin{cases} L_1 = L_2.L_3 \cup \{\varepsilon\} \\ L_2 = \{a\}.L_1.\{b\} \cup \{x\} \\ L_3 = L_3.\{c\}.L_2 \cup \{\varepsilon\} \end{cases}.$$

2 Induction structurelle

Exercice 8 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 propositionnelles 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 $w \in E$. Montrer que $|w|_N = 2|w|_B + |w|_U + 1$. En déduire une expression de $|w|$ en fonction de $|w|_B$ et $|w|_U$.

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

Exercice 9 Soit V un vocabulaire quelconque. Définir la fonction $|.|_A : V^* \to \mathbb{N}$ de l'exercice 8 par induction structurelle sur V^*.

Exercice 10 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 $i \neq j$.

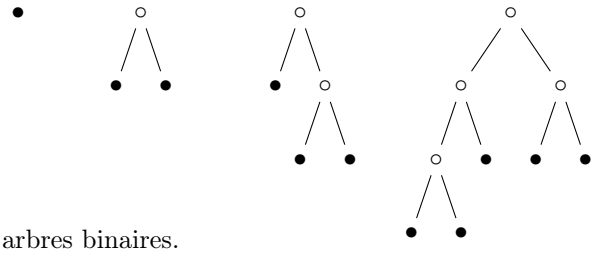
Exercice 11 Définitions inductives d'ensembles.

▷ **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.

Exercice 12
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 ∘ et le nombre n_f de feuilles (nœuds sans fils) • 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.

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

Exercice 13 [Avancé] On considère un langage E_{p} d'expressions préfixées *sur des chiffres*. Pour cela, on définit les ensembles suivants :

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

L'ensemble E_{p} est alors un langage sur le vocabulaire $V = \textsc{Num} \cup \textsc{Op}$, 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 $\times$ $+$ `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 $w =$ $\times$ $+$ `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 $|w|_{\textsc{Num}}$ le nombre d'éléments de Num présents dans w, et on note $|w|_{\textsc{Op}}$ 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 $|w|_{\textsc{Num}} = |w|_{\textsc{Op}} + 1$.

3 Expressions régulières

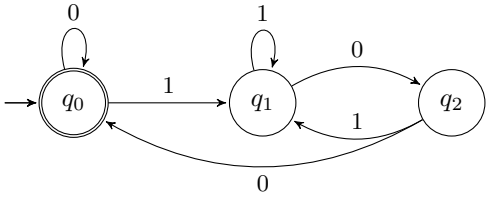
Exercice 14 [A savoir faire] Soit E une expression régulière. Simplifier les expressions suivantes :

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

Exercice 15 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.

Exercice 16 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.



Exercice 17 [A savoir faire] Caractériser (par une phrase en français) les langages représentés par les expressions régulières suivantes :

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

Exercice 18 [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.

Exercice 19 [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.

Exercice 20 [Avancé] En admettant le théorème de l'exercice 19, 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.

Exercice 21 [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).

Expliquer votre méthode sur les automates de :
1. l'exercice 38 (éliminiation des ε-transitions) ;
2. l'exercice 42 (déterminisation) ;
3. l'exercice 46 (les deux à la fois).

Indication : Montrer à l'aide du lemme d'Arden généralisé (cf. exercice 20) que la plus petite solution de $X = X + B(X)$ est aussi celle de $X = B(X)$.

Exercice 22 On considère les *expressions régulières étendues*, qui sont obtenues en ajoutant aux expressions régulières les constructions suivantes :
- si E est une E.R. étendue ou non, alors $\neg E$ est une E.R. étendue ;
- si E et E' sont des E.R. étendues ou non, alors $E \cap E'$ est une E.R. étendue ;
- si E est une E.R. étendue ou non, 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.

Exercice 23 [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.

Exercice 24 [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 :

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

▷ **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 $\mathrm{H}_K(L) = 0$.

4 Automates finis et modélisation

Exercice 25 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 séquences d'ADN codant une protéine : les mots sur le vocabulaire $\{A, T, C, G\}$ qui commencent par le codon d'initiation ATG et termine par l'un des trois codons-stop TAG, TGA, TAA.
4. Les mots sur $\{a, b\}$ contenant un nombre pair de a et un nombre pair de b.

Exercice 26 Montrer que les automates de l'exercice précédent sont corrects, c.-à-d. que les langages reconnus par les automates sont bien ceux de l'énoncé. Procéder par double inclusion.
Indication pour le dernier : Caractériser les langages reconnus depuis chaque état et procéder par récurrence sur la longueur du mot reconnu.

Exercice 27 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 :

$$\begin{aligned}
\texttt{nonzerodigit} &\stackrel{\text{def}}{=} \{\mathbf{1}, \ldots, \mathbf{9}\} \\
\texttt{digit} &\stackrel{\text{def}}{=} \{\mathbf{0}\} \cup \texttt{nonzerodigit} \\
\texttt{integer} &\stackrel{\text{def}}{=} \texttt{nonzerodigit}\,(\texttt{digit}^*) \cup \{\mathbf{0}\}^+ \\
\texttt{dot} &\stackrel{\text{def}}{=} \{.\} \\
\texttt{pointfloat} &\stackrel{\text{def}}{=} (\texttt{digit}^*)\,\texttt{dot}\,(\texttt{digit}^+) \cup (\texttt{digit}^+)\,\texttt{dot} \\
\\
\texttt{exponent} &\stackrel{\text{def}}{=} \{\mathbf{e}, \mathbf{E}\}\,\{\varepsilon, +, -\}\,\texttt{digit}^+ \\
\texttt{exponentfloat} &\stackrel{\text{def}}{=} \big(\texttt{digit}^+ \cup \texttt{pointfloat}\big)\,\texttt{exponent} \\
\texttt{number} &\stackrel{\text{def}}{=} \texttt{integer} \cup \texttt{pointfloat} \cup \texttt{exponentfloat}
\end{aligned}$$

▷ **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`.

Exercice 28 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 ?

Exercice 29 [A savoir faire] Donner des automates reconnaissant les langages suivants :

1. $L_1 = \{\omega \in \{a,b\}^* \mid \omega \text{ contient au moins un } a \text{ et un } b\}$.
2. $L_2 = \{\omega \in \{a,b\}^* \mid \omega \text{ ne contient pas deux } a \text{ consécutifs}\}$.
3. $L_3 = \{\omega \in \{a,b\}^* \mid \omega \text{ ne contient pas } \textit{plus de} \text{ deux } a \text{ consécutifs}\}$.
4. $L_4 = \{\omega \in \{a,b\}^* \mid \omega \text{ a } bab \text{ pour suffixe}\}$.
5. $L_5 = \{\omega \in \{a,b\}^* \mid \text{la cinquième lettre de } \omega \text{ est un } a\}$.

Exercice 30 [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.

Exercice 31 [Avancé] On considère un vocabulaire V et une relation $R \subseteq V \times V$. On définit

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

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.

Exercice 32 [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.

Exercice 33 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.

Exercice 34 Le produit de deux automates est une construction qui permet d'éxécuter deux automates en même temps sur un même mot. Il est formellement 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 :

a) les mots sur $\{0,1\}$ contenant un nombre pair de 0 ;

b) les mots sur $\{0,1\}$ contenant un nombre impair de 1.

▷ **Question 2.** Construire 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.

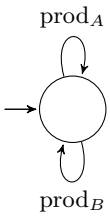
▷ **Question 6.** Que se passe-t-il si les automates A_1 et A_2 ne sont pas complets ? Considérer les cas où l'ensemble des états acceptants est $F_1 \times F_2$ ou $(F_1 \times Q_2) \cup (Q_1 \times F_2)$.

▷ **Question 7.** Comment faire pour adapter la construction de l'automate produit au cas où les automates A_1 et A_2 contiennent des ε-transitions ?

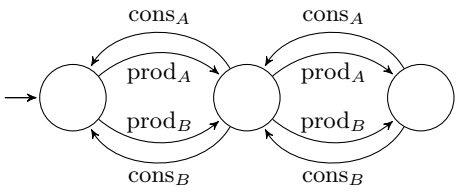
Exercice 35 Lorsqu'une personne souhaite donner des informations à une autre mais que les deux ne peuvent se rencontrer (se synchroniser), il est courant d'utiliser une boîte au lettre (électronique ou non), où la première personne dépose les informations que la seconde récupérera plus tard. En informatique, 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.

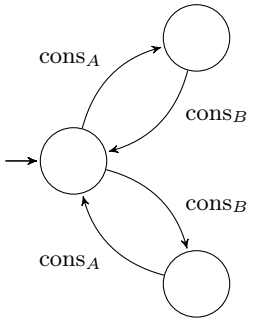
Supposons les trois automates suivants décrivant respectivement un producteur, un buffer et un consommateur. Notez qu'ils ne sont pas complets et qu'ils n'ont pas d'états acceptants car le système n'est pas sensé s'arrêter et évolue indéfiniment à chaque événement (chaque symbole prod_A, prod_B, cons_A, 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

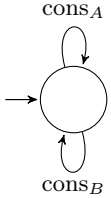


Consommateur

▷ **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 ?

Exercice 36 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 ?

Exercice 37

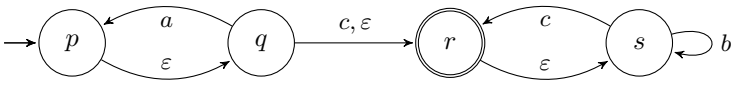
▷ **Question 1.** Montrer que la construction vue au CM3 de l'automate pour l'union est correcte :
$A_1 \cup A_2 \stackrel{\text{def}}{=} (Q_1 \uplus Q_2 \uplus \{i, f\}, V, \delta, \{i\}, \{f\})$ avec $\delta \stackrel{\text{def}}{=} \delta_1 \cup \delta_2 \cup \{(i, \varepsilon, q) \mid q \in I_1 \cup I_2\} \cup \{(q, \varepsilon, f) \mid q \in F_1 \cup F_2\}$.

▷ **Question 2. [Avancé]** Donner une définition formelle de la construction de l'automate pour l'itération vue au CM3 et montrer qu'elle est correcte.

5 Transformations d'automates

5.1 Élimination des ε-transitions

Exercice 38 [A savoir faire] Construire un automate sans ε-transition équivalent à celui ci-dessous :

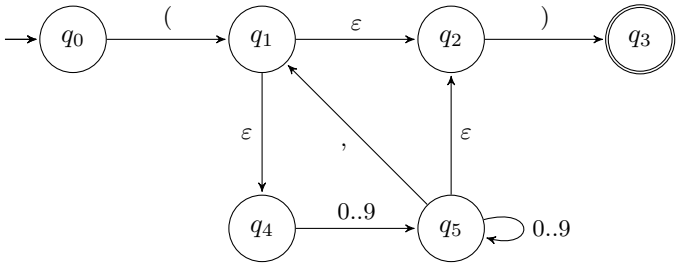


Exercice 39 [A savoir faire] On s'intéresse à l'ensemble des tuples d'entiers (simples) en Python. Dans ce langage, il est possible d'écrire :

- 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.

Exercice 40 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**.

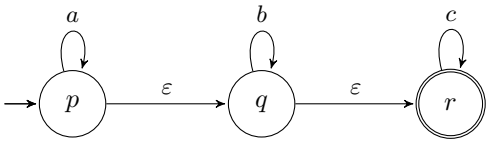
▷ **Question 2.** Définir un algorithme qui calcule l'ensemble des états accessibles dans A.

▷ **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 $\mathcal{L}(A) \neq \emptyset$.

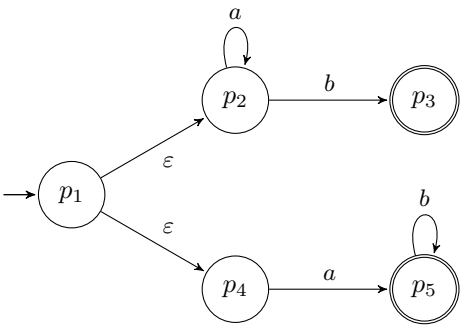
▷ **Question 5.** Donner une condition suffisante sur A permettant d'assurer $\mathcal{L}(A) = V^*$?

Exercice 41 [A savoir faire] Construire un automate sans ε-transition équivalent à celui ci-après. Quel est le langage reconnu par cet automate ?



5.2 Déterminisation

Exercice 42 [A savoir faire] Déterminiser l'automate suivant :



Exercice 43 [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 $\{a\}^* \{aba\}^*$.

Exercice 44 [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.

Exercice 45 [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.

Exercice 46 [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.

Exercice 47 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 ?

Exercice 48 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,

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

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

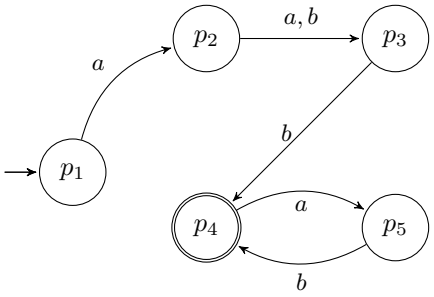
▷ **Question 2.** Construire un automate déterministe complet minimal reconnaissant L_2.

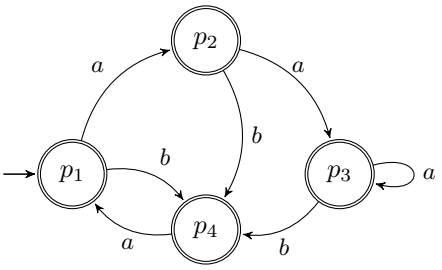
On cherche à borner la taille minimale d'un automate déterministe complet reconnaissant L_k. Soit $A = (Q, \{0, 1\}, \delta, \{q_0\}, F)$ un automate déterministe complet reconnaissant L_k. On définit $f : \{0, 1\}^k \to Q$, qui à tout mot u de longueur k associe $\delta^*(q_0, u)$. Autrement dit, $f(u)$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

5.3 Minimisation

Exercice 49 Minimiser les automates suivants :





Exercice 50 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	$-$	q_3, q_4
q_1	q_1	q_0	$-$
q_2	$-$	q_4	q_1
q_3	q_3	q_1	$-$
q_4	$-$	$-$	q_3

Exercice 51 [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 le facteur aa.
3. L'ensemble $\{a\}\{aa, bb\}^*\{a, b\}^*\{b\}$.

6 Propriétés de clôture

Exercice 52 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 $n \geqslant 0$. Montrer que le langage $M_n = \bigcup_{0\leqslant i\leqslant n} L_i$ est régulier.

▷ **Question 2.** Peut-on en déduire que $\bigcup_{i\in\mathbb{N}} L_i$ est régulier ? Justifier.

Exercice 53 On admet que $M = \{a^n b^n \mid n \geqslant 0\}$ n'est pas régulier. Montrer que les langages suivants ne sont pas réguliers non plus **sans se servir du lemme de l'étoile** :

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

Exercice 54 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. $L_2 = \left\{w \in \{a, b\}^* \;\middle|\; w \text{ est un palindrome}\right\}$
3. **[Avancé]** $L_3 = \left\{1^{i^2} \;\middle|\; i \geqslant 0\right\}$
4. **[Avancé]** $L_4 = \{1^p \mid p \text{ est premier}\}$

Exercice 55 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 34), comment faire cette construction plus directement ?

Exercice 56 [A savoir faire] Les langages suivants sont-ils réguliers ?

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

Exercice 57 [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.

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.

7 Grammaires et Hiérarchie de Chomsky

Exercice 58 Langages hors-contextes et langages réguliers

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

1. $\{a^n b^p \mid n \geqslant p \geqslant 0\}$
2. $\{a^n b^p \mid n \neq p\}$
3. $\{a^n b^p \mid 2p \geqslant n \geqslant p\}$
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

Exercice 59 Langages sous-contextes

▷ **Question 1.** Soit la grammaire $G = (\{a,b,c\}, \{S,B\}, S, R)$ 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 :

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

Exercice 60 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.

Exercice 61 [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 :

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

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

Exercice 62 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 $A \to w$ ou $A \to wB$ avec $w \in V_T^*$, $A \in V_N$ et $B \in V_N$.

Def2 : règles de la forme $A \to xB$ ou $A \to \varepsilon$ 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 $S \to abS \mid \varepsilon$. 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.

Exercice 63 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 $w_1, w_2 \in (V_T \cup V_N)^*$.

— une grammaire dont tous les symboles non terminaux sont accessibles et productifs est dite *réduite*.

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

$$S \to AB \qquad S \to \varepsilon \qquad A \to aA \qquad A \to D$$
$$B \to bB \qquad B \to aS \qquad C \to c \qquad C \to cC \qquad D \to dA$$

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 : $R' = R - \{A \to w_1 B w_2 \mid A \notin Pr \vee B \notin Pr\}$.
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 : $R'' = R' - \{A \to w_1 B w_2 \mid A \notin Ac \vee B \notin Ac\}$.

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

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

▷ **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 : $R' = R - \{A \rightarrow w_1 B w_2 \mid A \notin Ac \vee B \notin Ac \vee A \notin Pr \vee B \notin Pr\}$.

Exercice 64 Raisonnement sur les grammaires Une ε-règle est une règle de la forme $A \rightarrow \varepsilon$.

Une 1-règle est une règle de la forme $A \rightarrow B$ avec A et B éléments du vocabulaire non-terminal.
Soit G une grammaire hors-contexte ne contenant ni ε-règle ni 1-règle.

▷ **Question 1.** Pour tout mot $\omega \in \mathcal{L}(G)$ on peut borner la longueur de la dérivation $S \Longrightarrow^* \omega$ en fonction de $|\omega|$. Soit $S \Longrightarrow^d \omega$. Montrer que $d \leqslant 2 * |w| - 1$.

▷ **Question 2.** On suppose que G ne contient aucun symbole auto-imbriqué, c'est-à-dire de la forme $A \Longrightarrow^+ \alpha A \beta$. On veut montrer qu'il existe une borne supérieure de la taille des mots qui peuvent être produit par G. On propose la borne $M^{|V_N|}$ où M est la taille maximale d'une partie droite. En utilisant les arbres de dérivation, valider cette borne. Que peut-on dire de $\mathcal{L}(G)$?

▷ **Question 3.** Donner un algorithme permettant de déterminer si G contient un symbole auto-imbriqué.

8 Grammaires hors-contexte

Exercice 65 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 hors-contexte. 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.** 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 ?

▷ **Question 4. [Avancé]** Si on suppose que les grammaires G_1 et G_2 sont sous-contexte ou générales, est-ce que les résultats précédents se généralisent ?

Exercice 66 Grammaire ambiguë

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

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

▷ **Question 2.** Donner une grammaire non ambiguë pour le langage $\{a^n b^p \mid 2p \geqslant n \geqslant p\}$. Montrer que votre grammaire est non-ambiguë en vérifiant les deux conditions vues en cours qui ensemble suffisent à montrer la non-ambiguïté d'une grammaire.

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

Une 1-règle est une règle de la forme $A \rightarrow B$ 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 $d \leqslant 2 * |w| - 1$.

Exercice 67 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 $\forall u \in Prefixe(w), |u|_a \geqslant |u|_b$. 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 $L_1 \subseteq L_2$.

Exercice 68 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 quatre symboles suivants : 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.

9 Analyse LL(1)

Exercice 69 On considère la BNF suivante, sur le vocabulaire terminal $\{\mathbf{a}, \mathbf{b}, \mathbf{x}\}$, d'axiome S et avec deux autres non-terminaux U et O :

(1)	S	::=	U O	(3)	U	::=	**a S b**	(5)	O	::=	U O
(2)		|	ε	(4)		|	**x**	(6)		|	ε

▷ **Question 1.** En suivant les principes vus en cours, donnez les directeurs LL(1) de cette BNF. Est-elle LL(1) ?

▷ **Question 2.** Construisez l'arbre d'analyse du mot « **xaxxabxbaxb** » avec la méthode d'analyse LL(1). Quel caractère a été utilisé pour déterminer la règle à utiliser pour le dernier O ?

▷ **Question 3.** Écrivez en pseudo-code les fonctions d'analyse des non-terminaux : `parse_S`, `parse_U` et `parse_O`. Vous supposerez disposer d'une fonction `parse_token(t)` qui vérifie que `t` est bien le caractère courant et le consomme.

Exercice 70 Soit la BNF suivante $\langle \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}, \{\mathsf{O}, \mathsf{P}, \mathsf{S}\}, S, R\rangle$ dont voici les règles :

(1)	S	::=	O **c**	(2)	O	::=	P	
				(3)		|	ε	
(4)	P	::=	**b**					
(5)		|	**a** P **c** P					

▷ **Question 1.** Calculer les directeur LL(1) de cette BNF.

On pose $u =$ `abcb`.

▷ **Question 2.** Expliquez pourquoi et à quel moment l'analyse LL(1) échoue pour l'exemple $\mathbf{a}.u.\mathbf{bc}$.

▷ **Question 3.** Traitez les exemples $\mathbf{a}.u.\mathbf{cbc}$ et $\mathbf{abc}.u.\mathbf{c}$.

Exercice 71 On considère la grammaire suivante d'axiome E, sur le vocabulaire terminal $\{@, =, +, \mathbf{x}, \mathbf{1}\}$.

(1)	E	::=	L A	(7)	F	::=	L	
(2)		|	**1** O	(8)		|	**1**	
(3)	A	::=	= E	(9)	L	::=	**x** R	
(4)		|	O	(10)	R	::=	E @ R	
(5)	O	::=	+ F O	(11)		|	ε	
(6)		|	ε					

▷ **Question.** Calculer les directeurs LL(1) de cette BNF. Est-elle LL(1) ?

Exercice 72 Donner une BNF LL(1) de chacun des deux langages suivants :

$$L_1 = \{a^n b^m \mid 0 \leqslant n \leqslant m\} \qquad \text{et} \qquad L_2 = \{a^n b^m \mid 0 \leqslant m \leqslant n \leqslant m+2\}.$$

Exercice 73 Calculer les directeurs LL(1) des BNFs suivantes. Sont-elles LL(1) ? ambiguës ?

1. S ::= **a**X X ::= S**b** | **b**S | ε
2. S ::= XX | ε X ::= **b**S
3. S ::= Y**a**X**c**Y X ::= **a** | ε Y ::= **b**S | ε

Exercice 74 On considère les terminaux **NAT**, **MINUS** qui correspondent aux langages de lexèmes des expressions arithmétiques, plus les terminaux **SHARP**, **OPAR** et **CPAR** qui correspondent respectivement aux singletons {#}, {(} et {)}. On considère aussi la BNF suivante avec les non-terminaux S et exp :

(1)	S	::=	exp
(2)	exp	::=	**NAT**
(3)		|	**OPAR** exp **CPAR**
(4)		|	exp **SHARP**
(5)		|	exp **MINUS** exp

▷ **Question 1.** Cette BNF étant ambiguë, donner deux arbres d'analyse du mot "`3 # - 3 #`".

▷ **Question 2.** Calculer les directeurs LL(1) de chacune des règles, y compris la règle (1). La BNF est-elle LL(1) ?

Exercice 75 On considère les trois équations indépendantes suivantes sur le vocabulaire terminal $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$.

X ::= **a** X | **b** X | **a b** | **b a**
Y ::= **a a a** Y **b** | ε | **a b** | **a a b b**
Z ::= **a** Z **c** | Z **b** | ε

Pour chacune des équations, donner une grammaire LL(1) qui reconnaît le même langage que celui reconnu par l'équation. Dans chaque cas : appliquer des transformations de grammaires (e.g. factorisation à gauche, élimination de la récursion à gauche) qui préservent le langage ; et, justifier que la grammaire obtenue est LL(1) en donnant les directeurs.

Exercice 76 Dans les questions ci-dessous, on étudie des BNFs reconnaissant des fragments du langage C. Pour chacune des BNFs, on aimerait trouver une (E)BNF LL(1) engendrant le même langage que la BNF initiale. On justifiera le caractère LL(1) des (E)BNFs proposées.

▷ **Question 1.** En C, les instructions peuvent commencer par des labels de `goto`. Exemple :

```
etat1: if (cc=='a') goto etat1;
```

Voici la BNF à transformer en (E)BNF LL(1) :

$$\begin{array}{lcl} \mathsf{inst} & ::= & \textbf{idf}\!:\!\mathsf{inst} \mid \mathsf{exp}; \\ \mathsf{exp} & ::= & \textbf{idf}\!=\!\mathsf{exp} \mid \textbf{num} \end{array}$$

▷ **Question 2.** La syntaxe du langage C définit la notion de *lvalue*, pour « valeur à gauche » d'une affectation. C'est une catégorie d'expressions dont la valeur a une adresse mémoire. Typiquement, un littéral entier « `1` » n'est pas une lvalue mais une variable « `x` » en est une. Voici la BNF à transformer :

$$\begin{array}{lcl} \mathsf{list} & ::= & \mathsf{inst} \mid \mathsf{inst\ list} \\ \mathsf{inst} & ::= & \mathsf{exp}\ ; \\ \mathsf{exp} & ::= & \textbf{num} \mid \mathsf{lvalue} \mid \mathsf{lvalue} = \mathsf{exp} \mid \mathsf{lvalue}{+}{+} \\ \mathsf{lvalue} & ::= & \mathsf{lvalue}\ .\ \textbf{idf} \mid \mathsf{lvalue}\ [\ \mathsf{exp}\ ] \mid \textbf{idf} \end{array}$$

▷ **Question 3. [Avancé]** En C, une branche d'un **if/else** peut ne pas être délimitée par des accolades lorsqu'elle ne contient qu'une seule instruction (comme dans l'exemple de la question 1). Mais cela peut introduire des contre-sens sur la signification du programme (notamment si l'indentation est incorrecte). Voici ci-dessous une BNF G_1 qui reconnaît un fragment du langage C avec **if/else**. Le symbole exp correspond à celui de la question 2. Les symboles L et I remplacent respectivement les list et inst précédents.

$$\begin{array}{llllllll} (1) & \mathsf{L} ::= & \mathsf{I\ L} & (3) & \mathsf{I} ::= & \mathsf{exp}\ ; & (5) & \mathsf{O} ::= \ \textbf{else}\ \mathsf{B} \\ (2) & \mid & \varepsilon & (4) & \mid & \textbf{if}\ (\ \mathsf{exp}\ )\ \mathsf{B\ O} & (6) & \mid \ \varepsilon \end{array} \qquad \begin{array}{lll} (7) & \mathsf{B} ::= & \{\ \mathsf{L}\ \} \\ (8) & \mid & \mathsf{I} \end{array}$$

1. Montrer que la BNF G_1 est ambiguë.
2. Comme G_1 est ambiguë, elle n'est pas LL(1). Indiquer une paire de règles dont les directeurs sont en conflit.
3. On considère la BNF G_2 obtenu en supprimant la règle (8) de G_1. Calculer les directeurs de G_2. Est-elle LL(1) ? Est-elle ambiguë ?
4. Dans la sémantique du langage C, les ambiguïtés des **if/else** sont éliminées en rattachant le **else** au **if** le plus proche (parmi les **if** ambiguës). Par exemple, la sémantique de $\textbf{if}(e_1)\textbf{if}(e_2)e_3;\textbf{else}\,e_4;$ est équivalente à $\textbf{if}(e_1)\{\textbf{if}(e_2)e_3;\textbf{else}\,e_4;\};$
   Expliquer comment écrire un analyseur récursif inspiré d'une analyse LL(1) qui reconnaît le langage de G_1 et dont l'arbre des appels récursifs applique la règle de désambiguation ci-dessus : on donnera en particulier du pseudo-code pour `parse_0` et `parse_B`.

▷ **Question 4. [Avancé]** Il n'existe en fait pas de BNF LL(1) équivalente à G_1. Par contre, il existe une BNF non ambiguë équivalente (et même LALR).

$$\begin{array}{lll} \mathsf{L} ::= & \mathsf{L\ I} & \mid \ \varepsilon \end{array} \qquad \begin{array}{lll} \mathsf{I} ::= & \mathsf{I}_0 & \mid \ \mathsf{I}_1 \end{array} \qquad \begin{array}{lll} \mathsf{I}_0 ::= & \mathsf{exp}\ ; & \mid \ \textbf{if}\ (\ \mathsf{exp}\ )\ \mathsf{B}_0\ \textbf{else}\ \mathsf{B}_0 \end{array} \qquad \begin{array}{lll} \mathsf{B}_0 ::= & \{\ \mathsf{L}\ \} & \mid \ \mathsf{I}_0 \end{array}$$

$$\mathsf{I}_1 \ ::= \ \textbf{if}\ (\ \mathsf{exp}\ )\ \mathsf{I} \mid \textbf{if}\ (\ \mathsf{exp}\ )\ \{\ \mathsf{L}\ \} \mid \textbf{if}\ (\ \mathsf{exp}\ )\ \mathsf{B}_0\ \textbf{else}\ \mathsf{I}_1$$

1. Attacher sur cette BNF un système d'attributs qui associe à tout programme reconnu par G_1, un programme de même sémantique mais sans ambiguïté. Autrement dit, ce système d'attributs "ajoute" des accolades, comme dans l'exemple précédent, pour qu'on puisse comprendre le code sans avoir à appliquer la règle de désambiguation. On supposera écrit le non-terminal de profil $\mathsf{exp}{\uparrow}w$ où w est une chaîne de caractères correspondant à l'expression reconnue. On essaiera de minimiser les accolades ajoutées.
2. Appliquer votre système d'attributs sur le mot w_1="$\textbf{if}(e_1)\textbf{if}(e_2)e_3;\textbf{else if}(e_4)e_5;$" et le mot w_1 suivi de "$\textbf{else}\ e_6;$".

Exercice 77 On considère la BNF attribuée suivante sur le vocabulaire terminal $\{a, b, c\}$, avec des non-terminaux de profil $\mathsf{S}{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$, $\mathsf{X}{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$ et $\mathsf{Y}{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$:

(1) $\mathsf{S}{\downarrow}h{\uparrow}\max(r_1, r_2) ::= \ \textbf{a}\ \mathsf{X}{\downarrow}h{\uparrow}r_1\ \mathsf{Y}{\downarrow}h{\uparrow}r_2$

$$\begin{array}{llll} (2) & \mathsf{X}{\downarrow}h{\uparrow}r ::= & \mathsf{S}{\downarrow}h{+}1{\uparrow}r\ \textbf{b} & \\ (3) & \mid & \varepsilon & r := 3 \times h \end{array} \qquad \begin{array}{llll} (4) & \mathsf{Y}{\downarrow}h{\uparrow}r ::= & \textbf{c}\ \mathsf{Y}{\downarrow}h{+}1{\uparrow}r\ \textbf{a} & \\ (5) & \mid & \varepsilon & r := 2^h \end{array}$$

▷ **Question 1.** La BNF est-elle LL(1) ? Est-elle ambiguë ? Justifier.

▷ **Question 2.** Le mot `aaacabbca` est-il accepté par la BNF ? Si oui, en dessiner un arbre d'analyse avec la propagation d'attributs quand l'attribut hérité h à la racine vaut initialement 0.

▷ **Question 3.** On suppose définie ci-dessous la machine à états des analyseurs syntaxiques LL(1) vue en cours et en TD, qui modifie une variable globale `current` contenant le token de pré-vision.

```
Token = Enum('Token',['a','b','c','END']) # Token.END = token spécial de fin
def init_parser():   # initialise 'current' sur le premier token
def parse_token(t):  # vérifie 'current==t' et fait avancer 'current' sur le prochain token
```

Écrire le code Python d'une fonction "`parse()`" qui implémente l'analyseur spécifié par la BNF attribuée ci-dessus : elle retourne un entier r correspondant à celui calculé par le système d'attributs lorsque h est initialisé à 0. On fera bien attention à rejeter un mot comme "`acaa`" qui n'est pas reconnu par la BNF. On pourra introduire des fonctions auxiliaires.

Exercice 78 Sur le vocabulaire terminal $\{\mathbf{NAT}, -, **, (,)\}$ on considère la BNF suivante d'axiome $\mathsf{E}\uparrow\mathbb{R}$. Ici, $\mathbf{NAT}\uparrow\mathbb{N}$ représente les constantes entières positives.

$$\begin{array}{lcll} \mathsf{E}\uparrow r & ::= & \mathbf{NAT}\uparrow r & \\ & | & - \ \mathsf{E}\uparrow r_1 & r := -r_1 \\ & | & \mathsf{E}\uparrow r_1 \ ** \ \mathsf{E}\uparrow r_2 & r := r_1^{r_2} \\ & | & \mathsf{E}\uparrow r_1 \ - \ \mathsf{E}\uparrow r_2 & r := r_1 - r_2 \\ & | & (\mathsf{E}\uparrow r) & \end{array}$$

La sémantique de cette BNF ambiguë ne doit être calculée que sur des arbres d'analyse vérifiant les règles de priorités suivantes :

moins binaire	priorité 2 (MIN)	associatif à gauche
exposant ** et moins unaire	priorité 1	associatif à droite
le reste	priorité 0 (MAX)	

Ainsi, le mot "$\mathbf{NAT}\uparrow 1 - \mathbf{NAT}\uparrow 2 ** - \mathbf{NAT}\uparrow 3 ** \mathbf{NAT}\uparrow 4 - \mathbf{NAT}\uparrow 5$" correspond au calcul $(1 - 2^{-(3^4)}) - 5$. Donner une BNF attribuée **non ambiguë équivalente** pour la sémantique obtenue en appliquant les règles de priorités. Il faut donc encoder les règles de priorités dans la BNF. Vérifiez que votre BNF fonctionne comme attendu sur l'exemple.

Exercice 79 [Avancé] Le langage Java interdit l'utilisation de variables non-initialisées : cela participe par exemple à garantir que, dans ce langage, on ne peut pas "forger" de pointeurs.

```
1  x=1;
2  while (x <= 7) {
3      y=x;
4      x=x+1;
5  }
6  if (7 <= y) {
7      z=x;
8  } else {
9      z=z+1;
10 }
```

Le problème de savoir si un programme utilise ou pas des variables non-initialisées est indécidable. Le compilateur Java fait donc le choix d'interdire l'utilisation de toute variable dont il n'est pas *sûr* qu'elle soit initialisée. Par exemple, alors qu'à l'exécution le programme ci-contre initialise ses variables avec `y==7` et `z==8` (sans passer par la ligne 9), le compilateur Java rejette ce programme en déclarant que `y` et `z` sont utilisées aux lignes 6 et 9 alors qu'elles sont peut-être non-initialisées.

On considère ici la syntaxe abstraite ci-dessous. Autrement dit, cette BNF définit les arbres de syntaxe "simplifiés" sur lesquels on va décrire l'analyse faite par le compilateur Java.

$$\begin{array}{l} \mathsf{inst} ::= \varepsilon \ | \ \mathsf{exp} \ ; \ | \ \mathsf{inst} \ \mathsf{inst} \ | \ \mathbf{if} \ (\ \mathsf{exp}\ )\ \{\ \mathsf{inst}\ \}\ \mathbf{else}\ \{\ \mathsf{inst}\ \} \ | \ \mathbf{while}\ (\ \mathsf{exp}\ )\ \{\ \mathsf{inst}\ \} \\ \mathsf{exp} ::= \mathbf{idf} \ | \ \mathbf{idf} = \mathsf{exp} \ | \ \mathbf{num} \ | \ \mathsf{exp} + \mathsf{exp} \ | \ \mathsf{exp} <= \mathsf{exp} \end{array}$$

▷ **Question 1.** Ajouter des attributs permettant de calculer l'ensemble des variables *sûrement* initialisées. On supposera disposer du profil $\mathbf{idf}\uparrow x$ où x est un nom de variable.

▷ **Question 2.** Étendre le système précédent pour détecter les variables utilisées potentiellement non-initialisées. On utilisera le fait qu'en Java les expressions sont évaluées de gauche à droite.

Exercice 80 [Avancé] Soit $X \in \mathcal{P}(\mathbb{N})$. On définit $\min(X) = \{n \in X \mid \forall m \in X, n \leqslant m\}$.
Soit L un langage hors-contexte. On veut calculer $M(L) = \min(\{|w| \mid w \in L\})$. Ainsi, si L est non vide, $M(L)$ est un singleton entier qui correspond à la longueur minimum des mots de L.

1. Le calcul de $M(L)$ permettrait-il de décider si $L = \emptyset$? Et, si $\varepsilon \in L$?

2. On note $\mathcal{P}_1(\mathbb{N})$ l'ensemble des parties de $\mathbb{N}$ ayant au plus un élément. Soient $X, Y \in \mathcal{P}_1(\mathbb{N})$. On définit la relation $\sqsubseteq$ par $X \sqsubseteq Y \Leftrightarrow \forall n \in X, \exists m \in Y, n \geqslant m$. Montrer que $\sqsubseteq$ est un ordre total sur $\mathcal{P}_1(\mathbb{N})$. L'ordre $\sqsubseteq$ admet-il un minimum sur $\mathcal{P}_1(\mathbb{N})$? Un maximum ? Donner une formule simple du maximum de deux éléments X et Y en fonction de min ?
3. On admet que les résultats du cours sur les points fixes peuvent aussi s'appliquer lorsqu'on remplace $(\mathcal{P}(E), \subseteq)$ par $(\mathcal{P}_1(\mathbb{N}), \sqsubseteq)$ et avec la définition $\lim_{i \to +\infty} X_i = \min(\bigcup_{i \in \mathbb{N}} X_i)$.
Montrer comment utiliser le lemme de commutation sur la BNF de L, pour calculer un système d'équations dont le plus petit point fixe [2] donne $M(L)$.
Indication : on pourra introduire l'opération $X \dot{+} Y = \{n + m \mid n \in X \text{ et } m \in Y\}$.
4. Appliquer votre calcul sur la BNF suivante et calculer le plus petit point fixe.
   S ::= AB | D B ::= AaA | aD D ::= EC | aD
   A ::= **aaaa** | AA | **aa**C C ::= ε | S E ::= AD | DB
5. Le plus petit point fixe de votre système d'équations est-il atteint en un nombre fini d'itérations quelque soit L ?

Exercice 81 Analyse CYK

Voici une méthode pour transformer une grammaire HC *qui ne contient pas* ε en forme normale de Chomsky.

1. Suppression des ε-règles ($\mathsf{X} \to \varepsilon$) : on simule ces règles en les fusionnant avec celles qui précèdent
   (a) pour tout non-terminal X qui peut donner ε (qu'on peut calculer par $\varepsilon(\mathsf{X})$),
   (b) pour toute règle avec X en partie droite ($\mathsf{Y} \to \alpha\mathsf{X}\beta$),
   (c) on ajoute la règle $\mathsf{Y} \to \alpha\beta$.
2. Suppression des 1-règles ($\mathsf{X} \to \mathsf{Y}$) : comme pour l'élimination des ε-transitions
   (a) pour tout non-terminal X, on calcule les non-terminaux 1-accessibles (c.-à-d., uniquement par des 1-règles),
   (b) pour Y ($\neq \mathsf{X}$) 1-accessible depuis X et $\mathsf{Y} \to \alpha$,
   (c) on ajoute la règle $\mathsf{X} \to \alpha$.
3. Remplacement des terminaux :
   (a) Pour chaque terminal **a**, ajouter un nouveau non-terminal A et la règle A → **a**.
   (b) Dans toutes les règles avec plusieurs symboles en partie droite, remplacer **a** par A
4. Limitation de la taille en partie droite : on fait une chaîne de transitions
   $\mathsf{X} \to w_1 \dots w_n$ devient $\mathsf{X} \to w_1\mathsf{X}_1$, $\mathsf{X}_1 \to w_2\mathsf{X}_2$, …, $\mathsf{X}_{n-3} \to w_{n-2}\mathsf{X}_{n-2}$, $\mathsf{X}_{n-2} \to w_{n-1}w_n$.

▷ **Question 1.** Utiliser cette méthode pour transformer la grammaire S → **aSb** | **ab**.

▷ **Question 2.** En utilisant l'algorithme CYK sur cette nouvelle grammaire, analyser le mot *aaabbb*.

▷ **Question 3.** Mêmes questions pour la grammaire suivante et le mot *aacba*.
S → XaS | ε
X → S | SYb | ε
Y → SY | **c**

Exercice 82 Analyse LL et LR

On considère les grammaires suivantes :

G_1 : S → **a** S **b** | **a b**
G_2 : E → E op (E) | E op **nb** | **nb** op → + | − | ∗ | /

▷ **Question 1.** Dans une analyse LL de G_1, combien faut-il de terminaux pour choisir quelle règle utiliser ? Donner les suites de terminaux qui correspondent à chaque règle G_1.

▷ **Question 2.** Donner l'analyse du mot *aaabbb* par la grammaire G_1 en analyse LL et en analyse LR. Vous préciserez à chaque étape l'état de la pile et le ou les arbres déjà construits.

▷ **Question 3.** Donner les étapes de l'analyse LR du mot **nb**∗(**nb**+**nb**)−**nb** par la grammaire G_2.

▷ **Question 4.** Quel problème a-t-on pour une analyse LL du même mot ?

2. pour l'ordre point-à-point sur les n-uplets engendré par $\sqsubseteq$