1
Les langages.

Dans tous les exercices, $\mathcal{A}$ désigne un alphabet quelconque.

La concaténation.

Exercice 1.

a) La *longueur* $|u|$ de $u \in \mathcal{A}^*$ peut se définir par récurrence de la façon suivante :

$$|\varepsilon| = 0 \qquad |ux| = |u| + 1 \text{ pour tout } u \in \mathcal{A}^* \text{ et tout } x \in \mathcal{A}.$$

Vérifier que $|u \cdot v| = |u| + |v|$.

b) Le *nombre d'occurrences* de $a \in \mathcal{A}$ dans $u \in \mathcal{A}^*$, noté $|u|_a$, peut se définir par récurrence de la façon suivante :

$$|\varepsilon|_a = 0 \qquad |ux|_a = \begin{cases} |u|_a + 1 & \text{si } x = a \\ |u|_a & \text{sinon} \end{cases} \qquad \text{pour tout } u \in \mathcal{A}^* \text{ et tout } x \in \mathcal{A}.$$

Vérifier que $|u \cdot v|_a = |u|_a + |v|_a$.

Exercice 2. Lemme de Lévy.

Montrer que si les mots u, v, α et $\beta \in \mathcal{A}^*$ vérifient : $uv = \alpha\beta$ et $|u| \geq |\alpha|$, alors il existe $\delta \in \mathcal{A}^*$ tel que $u = \alpha\delta$ et $\beta = \delta v$.

Exercice 3. Langages commutatifs.

On dit que $L \subseteq \mathcal{A}^*$ est *commutatif* ssi pour tout $u \in L$ et tout $v \in L$ on a $uv = vu$.

a) Vérifier que, pour tout $w \in \mathcal{A}^*$, tout $L \subseteq w^*$ est commutatif.

b) Réciproquement, montrer que pour tout $L \subseteq \mathcal{A}^*$ commutatif, il existe $w \in \mathcal{A}^*$ tel que $L \subseteq w^*$.

Indications. Considérer le langage $\overline{L} \subseteq \mathcal{A}^*$ défini par : $v \in \overline{L}$ ssi $uv = vu$ pour tout $u \in L$.
Montrer alors que tout $w \in \overline{L} - \varepsilon$ de la plus petite longueur possible répond à la question.

Exercice 4. Relation de conjugaison.

On considère la relation binaire C sur $\mathcal{A}^*$ définie par : $C(u,v)$ ssi il existe $\gamma \in \mathcal{A}^*$ tel que $u\gamma = \gamma v$.

On dira que u et v sont *conjugués* lorsqu'ils vérifient $C(u,v)$.

a) Montrer que : $C(u,v)$ ssi il existe $\gamma \in \mathcal{A}^*$ tel que $u\gamma = \gamma v$ et $|u| \geq |\gamma|$.

b) Montrer que : $C(u,v)$ ssi il existe $\gamma \in \mathcal{A}^*$ et $\delta \in \mathcal{A}^*$ tels que $u = \gamma\delta$ et $v = \delta\gamma$.

c) On considère l'application $Conj : \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ définie par : $v \in Conj(u)$ ssi $C(u,v)$, qui à tout $u \in \mathcal{A}^*$ associe l'ensemble de ses conjugués et son extension aux langages $Conj : \mathcal{P}(\mathcal{A}^*) \to \mathcal{P}(\mathcal{A}^*)$, définie pour chaque $L \subseteq \mathcal{A}^*$, par : $v \in Conj(L)$ ssi il existe $u \in L$ tel que $v \in Conj(u)$.

Calculer

1) $Conj(abab)$,
2) $Conj(x\mathcal{A}^*y)$ et $Conj(x\mathcal{A}^*y\mathcal{A}^*z)$ pour des éléments x, y et z quelconques de $\mathcal{A}$.

Opérations simples sur les mots et les langages.

Exercice 5. Facteurs gauches.

L'ensemble $fg(u)$ des *facteurs gauches* d'un mot $u \in \mathcal{A}^*$, se définit par la récurrence suivante :

$$fg(\varepsilon) = \varepsilon \qquad fg(ux) = fg(u) + ux \text{ pour tout } u \in \mathcal{A}^* \text{ et tout } x \in \mathcal{A}.$$

a) Vérifier que cette définition est bien la bonne, c'est–à–dire que

$$v \in fg(u) \text{ ssi il existe } w \in \mathcal{A}^* \text{ tel que } u = vw.$$

b) Vérifier que $fg(uv) = fg(u) + ufg(v)$ pour tout $u \in \mathcal{A}^*$ et tout $v \in \mathcal{A}^*$.

c) L'application $fg : \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$, qui à tout mot associe l'ensemble de ses facteurs gauches, s'étend aux langages en une application préservant les sommes $fg : \mathcal{P}(\mathcal{A}^*) \to \mathcal{P}(\mathcal{A}^*)$, définie pour chaque $L \subseteq \mathcal{A}^*$, par

$$v \in fg(L) \text{ ssi il existe } u \in L \text{ tel que } v \in fg(u).$$

Vérifier que pour tout $L \subseteq \mathcal{A}^*$ et tout $M \subseteq \mathcal{A}^*$ on a

$$\text{si } M \neq \emptyset : fg(LM) = fg(L) + Lfg(M) \qquad \text{et} \qquad fg(L^*) = \varepsilon + L^* fg(L).$$

Exercice 6. Facteurs droits.

En mettant la définition de l'ensemble $fg(u)$ des facteurs gauches du mot $u \in \mathcal{A}^*$ devant un miroir on peut obtenir facilement une définition de l'ensemble $fd(u)$ de ses *facteurs droits*, par une récurrence basée sur l'ajout des lettres à gauche :

$$fd(\varepsilon) = \varepsilon \qquad fd(xu) = xu + fd(u) \text{ pour tout } x \in \mathcal{A} \text{ et tout } u \in \mathcal{A}^*.$$

Si l'on s'obstine à vouloir ajouter les lettres à droite, on est conduit à poser la définition par récurrence suivante :

$$fd(\varepsilon) = \varepsilon \qquad fd(ux) = \varepsilon + fd(u)x \text{ pour tout } x \in \mathcal{A} \text{ et tout } u \in \mathcal{A}^*.$$

a) Vérifier que cette définition est bien la bonne, c'est–à–dire que

$$v \in fd(u) \text{ ssi il existe } w \in \mathcal{A}^* \text{ tel que } u = wv.$$

b) Vérifier que $fd(uv) = fd(v) + fd(u)v$ pour tout $u \in \mathcal{A}^*$ et tout $v \in \mathcal{A}^*$.

c) Exprimer $fd(LM)$ et $fd(L^*)$ pour $L, M \subseteq \mathcal{A}^*$ quelconques.

Exercice 7. Facteurs.

L'ensemble $fact(u)$ des *facteurs* du mot $u \in \mathcal{A}^*$ peut se définir par récurrence de la façon suivante, si l'on ajoute des lettres "à droite et à gauche" :

$$fact(\varepsilon) = \varepsilon \qquad fact(x) = \varepsilon + x \text{ pout tout } x \in \mathcal{A},$$
$$fact(xuy) = xuy + fact(xu) + fact(uy) \text{ pour tout } x \text{ et } y \in \mathcal{A} \text{ et tout } u \in \mathcal{A}^*.$$

Si l'on s'obstine à vouloir ajouter les lettres à droite, on est conduit à poser la définition par récurrence suivante :

$$fact(\varepsilon) = \varepsilon \qquad fact(ux) = fact(u) + fd(u)x \text{ pout tout } x \in \mathcal{A} \text{ et tout } u \in \mathcal{A}^*.$$

a) Vérifier que cette définition est bien la bonne, c'est–à–dire que

$$v \in fact(u) \text{ ssi il existe } w \in \mathcal{A}^* \text{ et } w' \in \mathcal{A}^* \text{ tels que } u = wvw'.$$

b) Vérifier que $fact(uv) = fact(u) + fd(u)fg(v) + fact(v)$ pour tout $u \in \mathcal{A}^*$ et tout $v \in \mathcal{A}^*$.

c) Exprimer $fact(LM)$ et $fact(L^*)$ pour $L, M \subseteq \mathcal{A}^*$ quelconques.

Exercice 8. Facteurs stricts.

Désignons par f l'une des applications fg, fd et $fact$ précédentes.

a) Montrer que dans chaque cas on a $u \in f(u)$ pour tout $u \in \mathcal{A}^*$.

b) On définit la version stricte fs de chaque f par : $v \in fs(u)$ ssi $v \in f(u)$ et $v \neq u$.

Donner une définition par récurrence de chacune des applications fs.

c) Pour chaque f, exprimer $fs(LM)$ et $fs(L^*)$ pour $L, M \subseteq \mathcal{A}^*$ quelconques.

Exercice 9.

Calculer $fg(L_i), \ldots, facts(L_i)$ dans chacun des cas suivants :

$L_1 = a^*b^* = \{a^m b^n \mid 0 \leq m \text{ et } 0 \leq n\}$
$L_2 = \{a^m b^m \mid 0 \leq m\}$
$L_3 = \{a^m b^n \mid 0 \leq m \leq n\}$
$L_4 = \{a^m b^n \mid 0 \leq m < n\}$
$L_5 = \{a^m b^n \mid 0 \leq n \leq m\}$
$L_6 = \{a^m b^n \mid 0 \leq n < m\}$

où a et b sont des symboles distincts l'un de l'autre.

Exercice 10. Propriétés de l'itération.

a) Terminer la démonstration des propriétés 8) de l'itération, c'est–à–dire, montrer que, quels que soient $L \subseteq \mathcal{A}^*$ et $M \subseteq \mathcal{A}^*$:

1) $(L+M)^* \subseteq (L^* + M^*)^*$
2) $(L^* + M^*)^* \subseteq (L^* M^*)^*$
3) $(L^* M^*)^* \subseteq L^*(ML^*)^*$
4) $L^*(ML^*)^* \subseteq (L+M)^*$

b) Montrer que, quels que soient $L \subseteq \mathcal{A}^*$ et $M \subseteq \mathcal{A}^*$:

1) $(L^*M)^* = \varepsilon + (L+M)^* M$
2) $(LM^*)^* = \varepsilon + L(L+M)^*$

Exercice 11. Mélange de mots.

Cette opération consiste à insérer de nouvelles occurrences de caractères à un mot

a) *Mélange de deux mots.* Soit $mel(u,v)$ l'ensemble des mélanges de $u \in \mathcal{A}^*$ et $v \in \mathcal{A}^*$ défini par la (double) récurrence suivante :

$$\begin{aligned} mel(u,\varepsilon) &= u \\ mel(\varepsilon,v) &= v \\ mel(ux,vy) &= mel(u,vy)x + mel(ux,v)y \end{aligned}$$

1) Intuitivement, $w \in mel(u,v)$ est un mot de longueur $|u|+|v|$ obtenu en faisant "glisser" les caractères de v dans u, en respectant leur ordre relatif. Pour s'en persuader, calculer $mel(abc,def)$.

2) Vérifier que $mel(u,v) = mel(v,u)$.

b) *Mélange de deux langages.* L'application $mel : \mathcal{A}^* \times \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ se prolonge aux langages par :

$$mel(L,M) = \sum_{(v,w)\in L\times M} mel(v,w)$$

c'est–à–dire : $u \in mel(L,M)$ ssi il existe $v \in L$ et $w \in M$ tels que $u \in mel(v,w)$.

Calculer $mel(a^*,b^*)$ et $mel(ab,a^*b^*)$.

Exercice 12. Division à gauche d'un langage par un autre.

Nous allons définir une opération qui se présente comme l'inverse de la concaténation, mais qui ne tient ses promesses que dans des cas très particuliers. La concaténation n'étant pas commutative, on doit envisager une division à gauche et une division à droite : nous n'envisagerons que la première (la seconde s'obtient en regardant la première dans un bon miroir).

Pour tout $L \subseteq \mathcal{A}^*$ et tout $M \subseteq \mathcal{A}^*$ on définit $M^{-1}L \subseteq \mathcal{A}^*$ par

$$w \in M^{-1}L \text{ ssi il existe } v \in M \text{ tel que } vw \in L.$$

Quelle condition M doit–il satisfaire pour que l'on ait $M(M^{-1}L) = L$ quel que soit L ?

Considérons l'opération $\diamond : \mathcal{A}^* \times \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ définie par la double récurrence suivante : pour tout x et $y \in \mathcal{A}$, tout u et $v \in \mathcal{A}^*$

$$u \diamond \varepsilon = u \qquad \text{et} \qquad u \diamond (vy) = (u \diamond v) \diamond y,$$

$$\varepsilon \diamond y = \emptyset \qquad \text{et} \qquad (xu) \diamond y = \begin{cases} u & \text{si } x = y, \\ \emptyset & \text{sinon.} \end{cases}$$

a) Montrer que $u \diamond v = w$ ssi $u = vw$ pour tout u, v et $w \in \mathcal{A}^*$.

En déduire que $L \diamond M = M^{-1}L$ pour tout L et $M \subseteq \mathcal{A}^*$.

b) Montrer que pour tout L_1, L_2 et $L \subseteq \mathcal{A}^*$ et tout $y \in \mathcal{A}$:

$$(L_1L_2) \diamond y = \begin{cases} (L_1 \diamond y)L_2 + L_2 \diamond y & \text{si } \varepsilon \in L_1 \\ (L_1 \diamond y)L_2 & \text{sinon} \end{cases} \qquad \text{et} \qquad L^* \diamond y = (L \diamond y)L^*.$$

Ces égalités sont évidemment difficiles à étendre au cas des mots et donc des langages!
Comme d'habitude, l'inversion d'une opération, même simple, est assez compliquée (dans la section 6.2.2 du chapitre 2, on verra comment calculer $M^{-1}L$ dans un cas particulier très intéressant).

Les substitutions.

Exercice 13.

Soient a, b et c trois symboles distincts et soit $L = \{a^m b^m c^m \mid 0 \leq m\}$.

Définir pour chacun des langages L_i ci–dessous, une substitution f vérifiant $f(L) = L_i$:

$L_1 = \{a^m b^m a^m \mid 0 \leq m\}$
$L_2 = \{a^m b^m c^{2m} \mid 0 \leq m\}$
$L_3 = \{a^m \mid 0 \leq m\}$
$L_4 = \{b^{2m} a^{3m} \mid 0 \leq m\}$

Exercice 14. Sous–mots.

L'effaçage d'occurrences de caractères à un mot est l'application de la substitution $sm : \mathcal{A} \to \mathcal{P}(\mathcal{A}^*)$ définie par $sm(x) = \varepsilon + x$ pour tout $x \in \mathcal{A}$. Pour $u \in \mathcal{A}^*$, tout $v \in sm(u)$ est appelé un *sous–mot de* u.

1) Calculer $sm(ababaa)$.

2) Montrer que : $v \in sm(u)$ ssi il existe $w \in \mathcal{A}^*$ tel que $u \in mel(v, w)$.

3) Réciproquement, montrer que $u \in sm(mel(u, v))$ pour tout $u \in \mathcal{A}^*$ et tout $v \in \mathcal{A}^*$.

Exercice 15.

Soit $f : \mathcal{A} \to \mathcal{P}(\mathcal{B}^*)$ une substitution.

a) Montrer que $f(fg(u)) \subseteq fg(f(u))$ pour tout $u \in \mathcal{A}^*$ (la première apparition de l'application "facteur gauche" fg est relative aux mots sur $\mathcal{A}$ alors que la seconde s'applique aux mots sur $\mathcal{B}$).

En utilisant une substitution telle que $f(a) = f(b) = ab$ où a et b sont des symboles distincts l'un de l'autre, montrer que l'inclusion inverse de la précédente n'est pas nécessairement vraie.

b) Montrer que si f est alphabétique, alors on a $f(fg(u)) = fg(f(u))$ pour tout $u \in \mathcal{A}^*$.

c) Reprendre les questions précédentes avec les opérations fd, ...

Exercice 16. Application inverse d'une application $f : \mathcal{A}^* \to \mathcal{P}(\mathcal{B}^*)$.

L'application inverse de f est $f^{-1} : \mathcal{B}^* \to \mathcal{P}(\mathcal{A}^*)$ définie pour tout $u \in \mathcal{A}^*$ et $v \in \mathcal{B}^*$ par

$$u \in f^{-1}(v) \text{ ssi } v \in f(u).$$

Remarque. Cette définition est intéressante mais f^{-1} n'a pas de propriété simple! En particulier, elle s'applique à une substitution (en considérant son extension aux mots) : mais, en général, l'application inverse d'une substitution n'est pas une substitution, même dans des cas très spéciaux (voir l'exercice suivant).

a) Montrer que, si l'on considère l'extension de f^{-1} aux langages, on a :

$$u \in f^{-1}(M) \text{ ssi } f(u) \cap M \neq \emptyset$$

pour tout $u \in \mathcal{A}^*$ et tout $M \subseteq \mathcal{B}^*$.

b) Comment cette propriété s'exprime–t–elle pour un homomorphisme $f : \mathcal{A} \to \mathcal{B}^*$?

c) *Application au calcul du mélange de deux langages* (cf. exercice 11). Soit $\overline{\mathcal{A}}$ l'alphabet obtenu en donnant le nouveau nom $\overline{x}$ à chaque $x \in \mathcal{A}$ (c'est–à–dire : $\overline{x} \in \overline{\mathcal{A}}$ ssi $x \in \mathcal{A}$), et soit $\mathcal{B} = \mathcal{A} + \overline{\mathcal{A}}$.

On considère les trois substitutions $f, g, h : \mathcal{B} \to \varepsilon + \mathcal{A}$ définies par :

$$f(x) = f(\overline{x}) = x \qquad g(x) = x, g(\overline{x}) = \varepsilon \qquad h(x) = \varepsilon, h(\overline{x}) = x$$

quel que soit $x \in \mathcal{A}$.

Montrer que pour tout $L \subseteq \mathcal{A}^*$ et tout $M \subseteq \mathcal{A}^*$ on a :

$$mel(L, M) = f(g^{-1}(L) \cap h^{-1}(M))$$

Exercice 17. Application inverse d'une substitution alphabétique.

Soit $f : \mathcal{A} \to \varepsilon + \mathcal{B}$ une substitution alphabétique (cf. section 6.2) : on se propose d'étudier son application inverse $f^{-1} : \mathcal{B}^* \to \mathcal{P}(\mathcal{A}^*)$. Pour cela, on considère :

- $Z \subseteq \mathcal{A}$ défini par : $x \in Z$ ssi $f(x) = \varepsilon$,
- $X_y \subseteq \mathcal{A}$ défini, pour chaque $y \in \mathcal{B}$, par : $x \in X_y$ ssi $f(x) = y$.

a) Montrer que $f^{-1}(\varepsilon) = Z^*$ et que, pour tout $v \in \mathcal{B}^*$ et tout $y \in \mathcal{B}$ on a $f^{-1}(vy) = f^{-1}(v)X_yZ^*$.

b) En déduire que, pour tout $L \subseteq \mathcal{B}^*$ et tout $M \subseteq \mathcal{B}^*$ on a :

$$f^{-1}(LM) = f^{-1}(L)f^{-1}(M) \qquad \text{et} \qquad f^{-1}(L^*) = Z^* + f^{-1}(L)^*.$$

On pourra illustrer cet exercice par l'exemple de sm (exercice 14).

Exercice 18.

On désigne par $\mathcal{B}$ l'alphabet $\mathcal{A} \times \mathcal{A}$ dont les caractères sont les couples de caractères de l'alphabet $\mathcal{A}$: il sera commode de représenter le couple $(x, y) \in \mathcal{B}$ par $[xy]$.

a) Soit $tc : \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ ("*transposition des couples*") l'application définie par :

- $tc(\varepsilon) = \varepsilon$,
- pour tout $u \in \mathcal{A}^*$ de longueur paire, tout x et tout $y \in \mathcal{A}$

$$tc(ux) = \emptyset \qquad tc(uxy) = tc(u)yx.$$

Calculer $tc(u)$ pour $u = ababab$ et $u = ababa$.

Trouver deux substitutions $f, g : \mathcal{B} \to \mathcal{A}^*$ telles que $tc(u) = g(f^{-1}(u))$ pour tout $u \in \mathcal{A}^*$.

b) Soit $usd : \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ ("*un sur deux*") l'application définie par :

- $usd(\varepsilon) = \varepsilon$,
- pour tout $u \in \mathcal{A}^*$ de longueur paire, tout x et tout $y \in \mathcal{A}$

$$usd(ux) = \emptyset \qquad usd(uxy) = usd(u)x.$$

Calculer $usd(u)$ pour $u = ababab$ et $u = ababa$.

Trouver deux substitutions $f, g : \mathcal{B} \to \mathcal{A}^*$ telles que $usd(u) = g(f^{-1}(u))$ pour tout $u \in \mathcal{A}^*$.

Matrices et systèmes d'équations linéaires en langages.

Exercice 19. Matrices de langages.

La notation matricielle habituelle n'est vraiment "parlante" que pour les petites dimensions : on prendra des exemples de dimensions 2 ou 3 pour illustrer les définitions qui suivent.

Une matrice de langages sur $\mathcal{A}$ est la donnée d'un quadruplet $(Q, R, \mathcal{A}, A)$ où

- Q et R sont deux alphabets,
- $\mathcal{A}$ est un alphabet,
- $A : QR \to \mathcal{P}(\mathcal{A}^*)$ est une application.

L'application $A : QR \to \mathcal{P}(\mathcal{A}^*)$ servira par la suite à désigner la matrice qu'elle définit.

Le couple (Q, R) est appelé *le format* de la matrice A.

L'alphabet Q sert à indexer les "lignes", R à indexer les "colonnes" et, le langage qui se trouve à l'intersection de la ligne $q \in Q$ et de la colonne $r \in R$ est $A(qr)$.

Cette convention fixée, on peut définir la somme et le produit de matrices "comme d'habitude".

- La somme de deux matrices d'un même format $A : QR \to \mathcal{P}(\mathcal{A}^*)$ et $A' : QR \to \mathcal{P}(\mathcal{A}^*)$ est la matrice $A + A' : QR \to \mathcal{P}(\mathcal{A}^*)$ définie par :

$$(A + A')(qr) = A(qr) + A'(qr) \text{ pour tout } q \in Q \text{ et tout } r \in R.$$

On peut définir de façon analogue la somme généralisée d'une famille de matrices d'un même format.

L'élément neutre pour la somme de matrices de format (Q, R) est défini par l'application qui envoie tout $qr \in QR$ sur $\emptyset$.

- Le produit de deux matrices de formats compatibles $A : QR \to \mathcal{P}(\mathcal{A}^*)$ et $B : RS \to \mathcal{P}(\mathcal{A}^*)$ est la matrice $AB : QS \to \mathcal{P}(\mathcal{A}^*)$ définie par :

$$(AB)(qs) = \sum_{r \in R} A(qr)B(rs) \text{ pour tout } q \in Q \text{ et tout } s \in S.$$

(On aura reconnu le produit "lignes par colonnes".)

L'élément neutre de format (Q, Q) (une matrice carrée) pour le produit de matrices est défini par l'application $\varepsilon_Q : Q^2 \to \mathcal{P}(\mathcal{A}^*)$ telle que

$$\varepsilon_Q(qr) = \begin{cases} \varepsilon & \text{si } q = r, \\ \emptyset & \text{sinon.} \end{cases}$$

- La relation d'inclusion entre deux matrices d'un même format comme ci–dessus est définie par :

$$A \subseteq A' \text{ ssi } A(qr) \subseteq A'(qr) \text{ pour tout } q \in Q \text{ et } r \in R.$$

Question. Transposer les propriétés de la somme et de la concaténation des langages au cas des matrices et vérifier les propriétés ainsi obtenues.

Exercice 20. Générateurs linéaires (cf. section 7.2.1).

La propriété principale exprime que la donnée d'un GL $A : Q^+ \to \mathcal{P}(\mathcal{A}^*)$ équivaut à celle d'une application $A : Q^2 \to \mathcal{P}(\mathcal{A}^*)$, c'est–à–dire, d'une matrice carrée!

On supposera fixé un GL défini par $A : Q^2 \to \mathcal{P}(\mathcal{A}^*)$ pour toute la suite de l'exercice.

a) Vérifier que $A(LqM) = A(Lq)A(qM)$, pour tout $L \subseteq Q^*$, tout $M \subseteq Q^*$ et tout $q \in Q$.

b) En déduire que $A(qLq)^* = A((qL)^*q) = A(q(Lq)^*)$ et en particulier que $A(qq)^* = A(q^*q) = A(qq^*)$.

c) Pour tout entier naturel i, on définit $A^i : Q^2 \to \mathcal{P}(\mathcal{A}^*)$ par la récurrence :

$$A^0 = \varepsilon_Q \text{ et } A^{i+1} = A^i A,$$

c'est–à–dire, pour tout $qr \in Q^2$:

$$A^0(qr) = \varepsilon_Q(qr) = \begin{cases} \varepsilon & \text{si } q = r \\ \emptyset & \text{sinon} \end{cases} \qquad \text{et} \qquad A^{i+1}(qr) = \sum_{s \in Q} A^i(qs)A(sr)$$

Enfin, on définit $A^* : Q^2 \to \mathcal{P}(\mathcal{A}^*)$ en posant $A^* = \sum_{0 \leq i} A^i$, c'est–à–dire $A^*(qr) = \sum_{0 \leq i} A^i(qr)$.

1) Soient $L \subseteq \mathcal{A}^*$ et $M \subseteq \mathcal{A}^*$.
Calculer G^* lorsque $Q = q + r$ est constitué de deux symboles distincts et
$$A(qq) = L, A(qr) = M, A(rq) = \emptyset \text{ et } A(rr) = \varepsilon.$$

Revenons maintenant au cas général.

2) Montrer que $A^{i+1}(qr) = A(qQ^ir)$ pour tout $qr \in Q^2$ et tout entier naturel i,
3) en déduire que pour tout $qr \in Q^2$: $A^*(qr) = A(Chem(q,r)) = \begin{cases} \varepsilon + A(qQ^*q) & \text{si } r = q, \\ A(qQ^*r) & \text{sinon.} \end{cases}$
4) Vérifier la transposition aux générateurs linéaires des propriétés de l'itération.

Exercice 21. Systèmes d'équations linéaires en langages.

La définition des matrices et des GL conduisent à modifier la présentation des systèmes linéaires. On remplace la notation indicielle, qui est souvent utilisée, par une notation "fonctionnelle" : chacun de nous est du reste bien habitué à écrire $u(i)$, ou plutôt $u[i]$, au lieu de u_i. Nous négligerons l'éventuelle valeur numérique de ces "indices", déjà maltraités, pour considérer ceux–ci comme de quelconques symboles, c'est–à–dire comme les éléments d'un alphabet, avec lesquels nous aurons plaisir à faire des chemins, puis des ensembles de chemins!

Voici :

Un système d'équations linéaires se présente sous la forme

$$(E) \qquad X(q) = \sum_{r \in Q} A(qr)X(r) + A'(q) \qquad \text{pour tout } q \in Q$$

où chaque $A(qr)$ et chaque $A'(q)$ est un langage sur un alphabet $\mathcal{A}$.

Une solution du système (E) se présente sous la forme d'une substitution $L : Q \to \mathcal{P}(\mathcal{A}^*)$ vérifiant

$$L(q) = \sum_{r \in Q} A(qr)L(r) + A'(q)$$

pour tout $q \in Q$.

a) Pour vérifier l'affirmation du début de la section 4.2, on est conduit à représenter A' sous la forme d'une matrice colonne : on introduit pour cela un nouveau symbole $\varrho \notin Q$ et on considère A' comme une application $A' : Q\varrho \to \mathcal{P}(\mathcal{A}^*)$, où $A'(q\varrho)$ n'est qu'une nouvelle façon d'écrire $A'(q)$.

En appliquant des résultats des deux exercices précédents, montrer que la matrice colonne $L = A^*A'$ vérifie l'égalité $L = AL + A'$ et que toute matrice colonne M vérifiant $M = AM + A'$ est telle que $L \subseteq M$.

On se propose maintenant de démontrer les principes (résolution partielle et substitution) sur lesquels est basée la résolution effective des systèmes d'équations dans la section 4.2. Pour ce faire, il est commode de regrouper A et A' en un seul GL (cf. section 7.2.1) qui, pour simplifier, sera encore désigné par A. Ceci se fait de la façon suivante, qui poursuit l'idée de **a)** :

Soit $\varrho \notin Q$ un nouveau symbole, alors, on étend la définition de A en celle d'un GL

$$A : (Q + \varrho)^2 \to \mathcal{P}(\mathcal{A}^*)$$

en posant :

- $A(q\varrho) = A'(q)$ pour tout $q \in Q$,
- $A(\varrho q) = \emptyset$ pour tout $q \in Q + \varrho$.

On dira que *ce GL est associé au système* (E).

On suppose qu'un système (E) comme ci–dessus est fixé et que A est le GL qui lui est associé.

Remarque. Tous les chemins qui nous intéressent mènent à ϱ!

Ceci a pour conséquence pratique que, pour une fois, il est indispensable de construire les mots et les chemins "à l'envers", c'est–à–dire par adjonction d'éléments à gauche : les définitions et les raisonnements par récurrence doivent donc tous suivre ce principe.

b) Montrer que $A((Q + \varrho)^*\varrho) = A(Q^*\varrho)$.

On définit une substitution $Gen : Q \to \mathcal{P}(\mathcal{A}^*)$ par :

$$Gen(q) = A(qQ^*\varrho)$$

pour tout $q \in Q$.

c) Montrer que Gen est la plus petite solution de (E).

d) Montrer que toute solution $M : Q \to \mathcal{P}(\mathcal{A}^*)$ du système (E) satisfait les égalités suivantes, pour tout entier naturel k :

$$M(q) = \sum_{r \in Q} A(qQ^k r) M(r) + \sum_{0 \leq i \leq k} A(qQ^i \varrho)$$

quel que soit $q \in Q$.

En déduire la propriété d'unicité : si, pour tout $qr \in Q^2$ on a $\varepsilon \notin A(qr)$, alors (E) admet une solution unique. (La méthode utilisée dans la section 4.1 sera un guide précieux.)

e) Opération de substitution.
Soient $s \in Q$ et $t \in Q$ tels que $s \neq t$ et soit (F) le système obtenu à partir de (E) en remplaçant, dans le second membre de l'équation de $X(t)$, l'occurrence de $X(s)$ par le second membre de son équation.

Montrer que la plus petite solution de (F) est égale à la plus petite solution de (E).

Indications. Soit $B : (Q + \varrho)^2 \to \mathcal{P}(\mathcal{A}^*)$ le GL associé à (F). On mettra en évidence une application $f : (Q + \varrho)^2 \to \mathcal{P}((Q + \varrho)^*)$ telle que :

$$B(qr) = A(f(qr))$$

pour tout $qr \in (Q + \varrho)^2$, que l'on étendra en une transformation de chemins ne modifiant pas leurs extrêmités (cf. section 7.2.2), puis on vérifiera que

- $A(f(\Gamma)) = B(\Gamma)$ pour tout ensemble de chemins $\Gamma \subseteq (Q + \varrho)^+$,
- $f(qQ^*\varrho) = qQ^*\varrho$ pour tout $q \in Q$.

f) Opération de résolution partielle.
Soit $s \in Q$ et soit (F) le système obtenu à partir de (E) en remplaçant l'équation en $X(s)$ de ce dernier (écrivons–la $X(s) = \mathbf{A}X(s) + \mathbf{B}$ en abrégé) par sa résolvante partielle (qui s'écrit alors $X(s) = \mathbf{A}^*\mathbf{B}$).

Montrer que la plus petite solution de (F) est égale à la plus petite solution de (E).

Systèmes d'équations algébriques en langages.

Exercice 22.

Décrire une construction par récurrence de la plus petite solution s du système ci–dessous, en utilisant la décomposition de s donnée à la section 6.6
Le système, qui s'écrit avec les alphabets :

- $\mathcal{A} = [\ +\ ] + \wedge + \vee + \to + \neg + \sum_{n \geq 0} p_n$ (2 parenthèses, 4 "symboles" et une infinité dénombrable de lettres $p_0, \ldots, p_n, \ldots$),
- $\mathcal{V}$ n'a qu'un seul élément X,

comporte une seule équation $X = l(X)$ où

$$l(X) = [X \wedge X] + [X \vee X] + [X \to X] + \neg X + \sum_{n \geq 0} p_n.$$

Exercice 23.

Soient $\mathcal{A}$ un alphabet et $l : \mathcal{A} \to \mathcal{P}(\mathcal{A}^*)$ une substitution.

Trouver la plus petite substitution $s : \mathcal{A} \to \mathcal{P}(\mathcal{A}^*)$ qui vérifie l'égalité $s = id + s \circ l$.

Indication. On s'inspirera de la méthode de la section 6.6, en remarquant qu'ici, $\mathcal{A}$ mène un double jeu.

Exercice 24. Systèmes d'équations linéaires en langages.

Adapter la méthode de résolution d'un système d'équations algébriques en langages au cas particulier d'un système linéaire (cf. sections 4, 6 et exercice 21).

2
Les langages réguliers et les automates finis.

$\mathcal{A}$ désigne un alphabet fini.

Exercice 1.

Soit $\mathcal{A} = \{a, b\}$ un alphabet à deux lettres.

a) Donner une expression régulière de chacun des langages sur $\mathcal{A}$ suivants :

- ensemble des mots se terminant par aa;
- ensemble des mots comportant le facteur aaa;
- ensemble des mots commençant par ab et se terminant par bb;
- ensemble des mots ayant un nombre pair d'occurrences de a;
- ensemble des mots ne comportant pas le facteur aa.
- ensemble des mots ne comportant pas le facteur aaa.

b) Montrer que $(a+bb^*a^2)^*b^*(\varepsilon+a)$ est une expression régulière de l'ensemble des mots ne comportant pas le facteur bab (ε est une abréviation pour $\emptyset^*$).

Exercice 2. Conjugaison (cf. exercice 1.4.b).

Montrer que $Conj(L)$ est régulier pour tout langage régulier L.

Indication. Soit $\mathbf{A} = (Q, \mathcal{A}, \bullet, q_0, F)$ un AFDC et soit $L = \mathcal{L}(\mathbf{A})$. Pour chaque $q \in Q$ on considère les deux AFDC suivants :

$$\mathbf{B}(q) = (Q, \mathcal{A}, \bullet, q_0, q) \qquad \text{(la seule sortie est } q\text{)}$$
$$\mathbf{C}(q) = (Q, \mathcal{A}, \bullet, q, F) \qquad \text{(l'entrée est } q\text{)}$$

et on pose $M(q) = \mathcal{L}(\mathbf{B}(q))$ et $N(q) = \mathcal{L}(\mathbf{C}(q))$.

Montrer que l'on a $L = \sum_{q\in Q} M(q)N(q)$ et $Conj(L) = \sum_{q\in Q} N(q)M(q)$.

Exercice 3.

Soit $\mathcal{A} = a$ un alphabet à un seul symbole.

Pour chaque $i \in \{0, 1, 2\}$ on définit $L_i \subseteq a^*$ par : $u \in L_i$ ssi $|u| \, mod \, 3 = i$ pour tout $u \in a^*$.

Trouver un ensemble d'états Q, un état initial $q_0 \in Q$, une application $\bullet : Q \times \mathcal{A} \to Q$ et, pour chaque $i \in \{0, 1, 2\}$ un ensemble $F_i \subseteq Q$ de sorties, de telle façon que l'AFDC $\mathbf{A}_i = (Q, \mathcal{A}, \bullet, q_0, F_i)$ reconnaisse le langage L_i.

Exercice 4.

Soit a un symbole.

Montrer que pour tout langage régulier $L \subseteq a^*$, il existe deux langages **finis** A et B et un mot $\gamma \in a^*$ tels que $L = A + B\gamma^*$ (Réciproquement, un langage de cette forme est régulier!).

Indication. Trouver la forme générale des AFDC sur un alphabet à une seule lettre : le théorème de Kleene fera le reste.

Exercice 5. Lemme d'itération pour les langages réguliers.

a) Démontrer *le lemme d'itération* suivant :

si $L \subseteq \mathcal{A}^*$ est un langage régulier alors il existe un entier $N > 0$ tel que pour tout $u \in L$ vérifiant $|u| \geq N$, on peut trouver trois mots u_1, v et u_2 sur $\mathcal{A}$ tels que l'on ait les propriétés

1) $u = u_1 v u_2$,
2) $|v| > 0$ et $|v| \leq N$,
3) pour tout entier k, $u_1 v^k u_2 \in L$.

Indication. Considérer les "longs" chemins que l'on peut parcourir dans un AFDC reconnaissant L.

b) En déduire que les langages suivants ne sont pas réguliers :

- $L = \{a^m b^m \mid m \geq 0\}$
- $L = \{a^{m^2} \mid m \geq 0\}$

c) On considère les langages $L = (a+b)^* ba (a+b)^*$ et $M = \{a^m b^m \mid m \geq 0\}$.

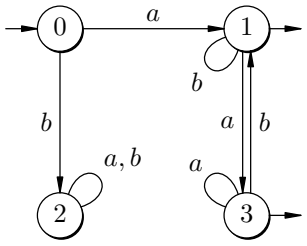
Montrer que le langage $L + M$ vérifie la conclusion du lemme d'itération ci–dessus mais qu'**il n'est pas régulier**.

Exercice 6. (cf. exercices 1.5, 1.6 et 1.7)

Soit f l'une des applications fg, fd, $fact$.

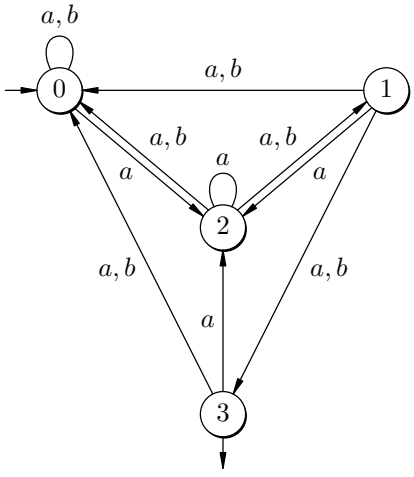
a) Montrer que f transforme un langage régulier en un langage régulier, sans utiliser aucun AF.

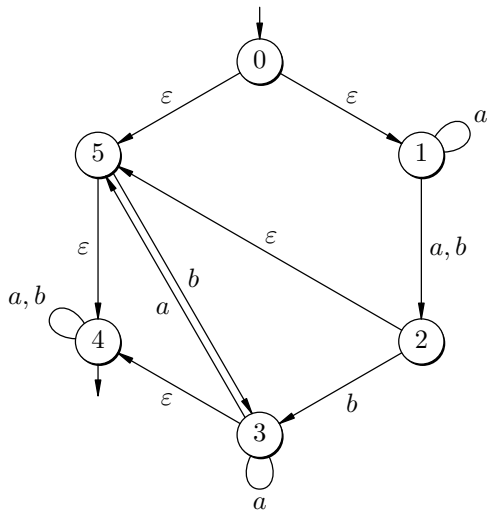
b) Construire un AF reconnaissant $f(L)$ à partir d'un AFDC reconnaissant L. Appliquer cette construction à l'AFDC ci–contre.



Exercice 7.

Déterminer l'AF et l'ε–AF suivants :





Exercice 8.

Construire des AFDC qui reconnaissent respectivement chacun des langages décrits dans l'exercice 1.a). **Indication.** On pourra commencer par construire un ε–AF reconnaissant "visiblement" ledit langage ou son complémentaire.

Exercice 9.

Soit $\mathcal{A} = a + b$ un alphabet à deux symboles.

a) Reprendre l'exercice 3 ci–dessus, mais cette fois pour les $L_i \subseteq \mathcal{A}^*$ définis par : $u \in L_i$ ssi $|u|_a \; mod \, 3 = i$ pour tout $u \in \mathcal{A}^*$.

b) Construire un AFDC sur $\mathcal{A}$ qui reconnaît l'ensemble des mots $u \in \mathcal{A}^*$ ne comportant pas le facteur bab et tels que $|u|_a \bmod 3 = 2$.

Exercice 10.

a) Soit $\mathbf{A}$ un AFDC sur un alphabet $\mathcal{A}$ et posons $L = \mathcal{L}(\mathbf{A})$.

Construire un ε–AF qui reconnaît l'ensemble des mots sur $\mathcal{A}$ dont aucun facteur n'est un élément de L.

b) Appliquer a) pour construire un AFDC reconnaissant l'ensemble des mots sur $\{a, b\}$ ne comportant pas le facteur bab, puis utiliser cet AFDC pour redémontrer le résultat de l'exercice 1.b) ci–dessus.

Exercice 11. Image par une substitution.

Soit $f : \mathcal{A} \to \mathcal{P}(\mathcal{B}^*)$ une substitution telle que $f(x)$ est régulier pour tout $x \in \mathcal{A}$.

a) Montrer que $f(L)$ est régulier pour tout $L \subseteq \mathcal{A}^*$ régulier, sans utiliser aucun AF.

b) Soit $\mathbf{A}$ un AFDC reconnaissant L et soit, pour chaque $x \in \mathcal{A}$, $\mathbf{A}(x)$ un AFDC reconnaisant $f(x)$: construire un ε–AF reconnaissant $f(L)$.

Appliquer cette construction à la substitution sm (cf. exercice 1.14)

Exercice 12. Image inverse par une substitution.

Soit $f : \mathcal{A} \to \mathcal{P}(\mathcal{B}^*)$ une substitution et soit $f^{-1} : \mathcal{B}^* \to \mathcal{P}(\mathcal{A}^*)$ son application inverse (cf. exercice 1.15), soit enfin $\mathbf{B} = (Q, \mathcal{B}, \underset{\mathbf{B}}{\bullet}, q_0, F)$ un AFDC : on considère alors l'ε–AF $\mathbf{A} = (Q, \mathcal{A}, \underset{\mathbf{A}}{\bullet}, q_0, F)$ dont l'action est définie par $q \underset{\mathbf{A}}{\bullet} x = q \underset{\mathbf{B}}{\bullet} f(x)$ (cf. section 6.2.2).

a) Montrer que si $M = \mathcal{L}(\mathbf{B})$ alors $\mathcal{L}(\mathbf{A}) = f^{-1}(M)$.

b) Etudier le cas où $f(x)$ est un langage régulier pour tout $x \in \mathcal{A}$.

c) Montrer, sans utiliser aucun AF, que l'image inverse d'un langage régulier par une subtitution alphabétique est un langage régulier (cf. exercice 1.16).

Exercice 13.

Appliquer les deux exercices précédents pour montrer que chacune des applications applications tc et usd de l'exercice 1.18 transforme un langage régulier en un langage régulier.

Exercice 14.

Pour tout $L \subseteq \mathcal{A}^*$ on définit $R(L) \subseteq \mathcal{A}^*$ par

$$u \in R(L) \text{ ssi il existe } v \in \mathcal{A}^* \text{ tel que } |v| = |u| \text{ et } uv \in L$$

pour tout $u \in \mathcal{A}^*$.
Le but de cet exercice est de montrer que $R(L)$ est régulier pour tout langage régulier L.

a) Appliquer directement la définition précédente pour calculer $R(\varepsilon + aba^*)$.

b) Soit $\mathbf{A} = (Q, \mathcal{A}, \bullet, q_0, F)$ un AFDC, on considère l'AF $\mathbf{R}$ qui est la partie accessible (on a aussi intérêt à ne conserver que les états productifs) de l'AF $(Q \times Q, \mathcal{A}, \circ, I, G)$ défini de la façon suivante :

- $(s,t) \in (q,r) \circ x$ ssi $q \bullet x = s$ et $t \in r \bullet \mathcal{A}^{-1}$ *(action)*
- $I = \{(q_0, r) \mid r \in F\}$ *(ensemble des entrées)*
- $G = \{(q,q) \mid q \in Q\}$ *(ensemble des sorties)*

Appliquer cette construction à un AFDC reconnaissant le langage $\varepsilon + aba^*$.

Montrer que pour tout $u \in \mathcal{A}^*$, on a :

$$(s,t) \in (q,r) \circ u \text{ ssi } q \bullet u = s \text{ et il existe } v \in \mathcal{A}^* \text{ tel que } |v| = |u| \text{ et } t \bullet v = r$$

quels que soient q, r, s et $t \in Q$.

En déduire que si $L = \mathcal{L}(\mathbf{A})$ alors $\mathcal{L}(\mathbf{R}) = R(L)$.

Utiliser ce résultat pour vérifier le calcul précédent de $R(\varepsilon + aba^*)$.

Exercice 15. Mélange de langages (cf. exercice 1.11).

Soient $\mathbf{A} = (Q^{\mathbf{A}}, \mathcal{A}, \underset{\mathbf{A}}{\bullet}, q_0^{\mathbf{A}}, F^{\mathbf{A}})$ et $\mathbf{B} = (Q^{\mathbf{B}}, \mathcal{A}, \underset{\mathbf{B}}{\bullet}, q_0^{\mathbf{B}}, F^{\mathbf{B}})$ deux AFD.
On définit l'AF $\mathbf{C} = (Q, \mathcal{A}, \circ, q_0, F)$ à une seule entrée de la façon suivante :

- *Etats.* $Q = Q^{\mathbf{A}} \times Q^{\mathbf{B}}$,
- *Action.* $(q,r) \circ x = (q \underset{\mathbf{A}}{\bullet} x, r) + (q, r \underset{\mathbf{B}}{\bullet} x)$ pour tout $q \in Q^{\mathbf{A}}$ et tout $r \in Q^{\mathbf{B}}$,
- *Entrée.* $q_0 = (q_0^{\mathbf{A}}, q_0^{\mathbf{B}})$,
- *Sorties.* $F = F^{\mathbf{A}} \times F^{\mathbf{B}}$.

a) Montrer que si $L = \mathcal{L}(\mathbf{A})$ et $M = \mathcal{L}(\mathbf{B})$ alors $\mathcal{L}(\mathbf{C}) = mel(L,M)$.

b) Applications.

- Vérifier les calculs de la question **b)** de l'exercice 1.11 en utilisant la construction précédente.
- Construire un AF qui reconnaît l'ensemble des mots $u \in (a+b)^*$ tels que $(|u|_a - |u|_b) \bmod 3 = 1$ (cf. exercice 9.a).

Exercice 16. Langages locaux.

Un langage M sur un alphabet $\mathcal{B}$ est dit local ssi il existe $X \subseteq \mathcal{B}$, $Y \subseteq \mathcal{B}$ et $Z \subseteq \mathcal{B}^2$ tels que

$$M - \varepsilon = (X\mathcal{B}^* \cap \mathcal{B}^*Y) - \mathcal{B}^*Z\mathcal{B}^*.$$

Pour décider si un mot $u \in \mathcal{B}^*$ appartient à un langage local, il suffit de tester les facteurs de u dont la longueur est au plus 2, aussi les appelle–t–on souvent *langages 2–reconnaissables*.

Un langage local est évidemment régulier.
L'objet de cet exercice est de donner une réciproque à cette propriété :

tout langage régulier est l'image d'un langage local par une substitution strictement alphabétique.

Soit L un langage régulier ne contenant pas ε et soit $\mathbf{A} = (Q, \mathcal{A}, \bullet, q_0, F)$ un AFDC reconnaissant L. On considère un alphabet $\mathcal{B}$ dont les symboles sont des éléments de $Q \times \mathcal{A} \times Q$:

$$\mathcal{B} = \{[q,x,r] \mid q \bullet x = r\}.$$

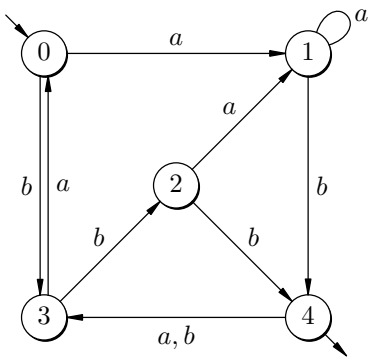
On considère alors les langages $X \subseteq \mathcal{B}$, $Y \subseteq \mathcal{B}$ et $Z \subseteq \mathcal{B}^2$ définis par :
$[q,x,r] \in X$ ssi $q = q_0$,
$[q,x,r] \in Y$ ssi $r \in F$,
$[q,x,r][s,y,t] \in Z$ ssi $r \neq s$,

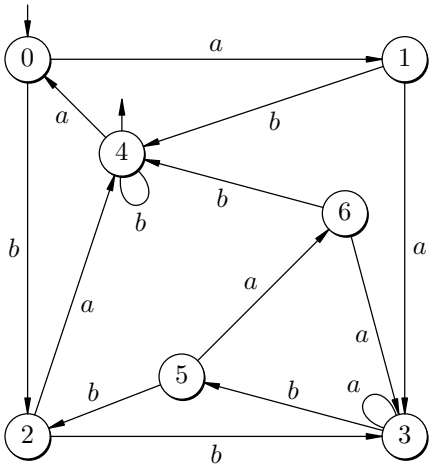
et la substitution $f : \mathcal{B} \to \mathcal{A}$ définie par $f([q,x,r]) = x$.

a) Montrer que $L = f((X\mathcal{B}^* \cap \mathcal{B}^*Y) - \mathcal{B}^*Z\mathcal{B}^*)$.
b) Appliquer la construction précédente à $L = b^2(a+b)^*a$.

Exercice 17.

Minimiser les AFDC suivants :





Exercice 18. AF spéciaux (AFS).

Un AFS sur $\mathcal{A}$ est un couple $(\mathbf{A}, \mathcal{E}ps)$ où :

- $\mathbf{A} = (Q, \mathcal{A}, \bullet, I, f)$ est un AF à une seule sortie, vérifiant les conditions suivantes :
  1) $f \notin I$,
  2) $f \bullet x = \emptyset$ pour tout $x \in \mathcal{A}$,
  3) pour tout $q \in Q - f$, il existe un unique $x \in \mathcal{A}$ tel que $q \bullet x \neq \emptyset$;
- $\mathcal{E}ps \subseteq \varepsilon$ (on a donc $\mathcal{E}ps = \emptyset$ ou $\mathcal{E}ps = \varepsilon$).

Le langage $\mathcal{L}(\mathbf{A}, \mathcal{E}ps) \subseteq \mathcal{A}^*$ reconnu par un AFS $(\mathbf{A}, \mathcal{E}ps)$ est défini par :

$$\mathcal{L}(\mathbf{A}, \mathcal{E}ps) = \mathcal{L}(\mathbf{A}) + \mathcal{E}ps$$

Remarques.

• Le fait que $\mathbf{A}$ a une seule sortie n'est pas très important et ne sert qu'à simplifier la construction **4)** ci–dessous.

• La condition 1) implique évidemment que $\varepsilon \notin \mathcal{L}(\mathbf{A})$: $\mathcal{E}ps$ sert à pallier cet éventuel défaut.

• La condition 2) signifie que l'on ne peut que sortir lorsqu'on est parvenu en f.

• La condition 3) implique que la table de $\mathbf{A}$ est très lacunaire : la ligne de f est vide et la ligne de chaque $q \neq f$ comporte une seule case non vide, ce qui est un avantage évident lorsqu'il s'agit de calculer un AFDC équivalent. On peut aussi profiter de cette condition pour simplifier les représentations de $\mathbf{A}$ (comment?).

L'objet de l'exercice est de vérifier que tout langage régulier est reconnaissable par un AFS.

a) Adapter l'algorithme de détermination des AF au cas des AFS ($\mathcal{E}ps$ doit intervenir!) et appliquer l'algorithme ainsi trouvé à des exemples simples d'AFS.

b) Les constructions suivantes permettent de construire un AFS reconnaissant un langage régulier $L \subseteq \mathcal{A}^*$: vérifier qu'elles sont correctes.

1) $L = \emptyset$ est reconnu par l'AFS $((f, \mathcal{A}, \bullet, \emptyset, f), \emptyset)$ à un seul état.

2) Pour tout $x \in \mathcal{A}$, $L = x$ est reconnu par l'AFS $((q_0 + f, \mathcal{A}, \bullet, q_0, f), \emptyset)$ à deux états, où $q_0 \bullet x = f$.

Pour la suite, on considère deux AFS $((Q_1, \mathcal{A}, \underset{1}{\bullet}, I_1, f_1), \mathcal{E}ps_1)$ et $((Q_2, \mathcal{A}, \underset{2}{\bullet}, I_2, f_2), \mathcal{E}ps_2)$, tels que $Q_1 \cap Q_2 = \emptyset$, et on appelle L_1 et L_2 les langages qu'ils reconnaissent respectivement.

3) $L = L_1 + L_2$ est reconnu par l'AFS construit de la façon suivante :

- $Q = (Q_1 - f_1) + (Q_2 - f_2) + f$ où $f \notin Q_1 + Q_2$ est un nouvel état,

- $I = I_1 + I_2$,
- f est le nouvel état introduit ci–dessus,
- pour chaque $i \in \{1,2\}$, si $q \in Q_i$ et $x \in \mathcal{A}$ sont tels que $f_i \in q \underset{i}{\bullet} x$

  alors $q \bullet x = (q \underset{i}{\bullet} x - f_i) + f$

  sinon $q \bullet x = q \underset{i}{\bullet} x$,

  (f_1 et f_2 ont été identifiées en f),
- $\mathcal{E}ps = \mathcal{E}ps_1 + \mathcal{E}ps_2$.

4) $L = L_1 L_2$ est reconnu par l'AFS construit de la façon suivante* :

- $Q = (Q_1 - f_1) + Q_2$,
- $I = I_1 + \mathcal{E}ps_1 I_2$,
- $f = f_2$,
- soient $q \in Q$ et $x \in \mathcal{A}$, alors :

  1) pour $q \in Q_1$: $q \bullet x = \begin{cases} (q \underset{1}{\bullet} x - f_1) + I_2 + f\mathcal{E}ps_2 & \text{si } f_1 \in q \underset{1}{\bullet} x, \\ q \underset{1}{\bullet} x & \text{sinon,} \end{cases}$

  2) pour $q \in Q_2$: $q \bullet x = q \underset{2}{\bullet} x$,

  (f_1 s'est multipliée pour s'indentifier respectivement à chacun des éléments de I_2),
- $\mathcal{E}ps = \mathcal{E}ps_1 \mathcal{E}ps_2$.

5) $L = L_1^*$ est reconnu par l'AFS construit de la façon suivante :

- $Q = Q_1$,
- $I = I_1$,
- $f = f_1$,
- soient $q \in Q_1$ et $x \in \mathcal{A}$, alors $q \bullet x = \begin{cases} q \underset{1}{\bullet} x + I_1 & \text{si } f_1 \in q \underset{1}{\bullet} x, \\ q \underset{1}{\bullet} x & \text{sinon,} \end{cases}$

  (les antécédents de la sortie sont branchés sur les entrées),
- $\mathcal{E}ps = \varepsilon$.

c) Appliquer la méthode précédente pour calculer un AFS, puis un AFDC sur $\{a,b\}$, reconnaissant le langage $b^* + (a+b)^* a^* a$.

Exercice 19. AF généralisés (AFG).

Voici une méthode "géométrique", pour construire un ε–AF reconnaissant un langage régulier : elle est assez intuitive mais nécessite la considération d'un type très général d'automates finis, dont nous décrirons seulement les graphes de transition.

Le graphe de transition d'un AFG $\mathbf{A}$ sur l'alphabet $\mathcal{A}$ est la donnée des éléments suivants :

- Un graphe de transition proprement dit, constitué de nœuds et d'arêtes étiquetés :
  - un nœud (q) pour chaque élément q d'un ensemble fini Q,
  - un ensemble fini d'arêtes $(q) \xrightarrow{L} (r)$ où $q \in Q, r \in Q$ et $L \subseteq \mathcal{A}^*$

    (il est inutile de faire figurer les arêtes vides $(q) \xrightarrow{\emptyset} (r)$),
- un ensemble d'entrées $I \subseteq Q$,
- un ensemble de sorties $F \subseteq Q$.

Le langage reconnu par un tel AFG se définit en adaptant la notion habituelle de dérivation :

Une transition $(q, vu) \overset{1}{\vdash} (r, u)$ est définie pour tout $v \in L$ tel que $(q) \xrightarrow{L} (r)$ soit une arête du graphe de $\mathbf{A}$. *Une dérivation* est un enchaînement de transitions.

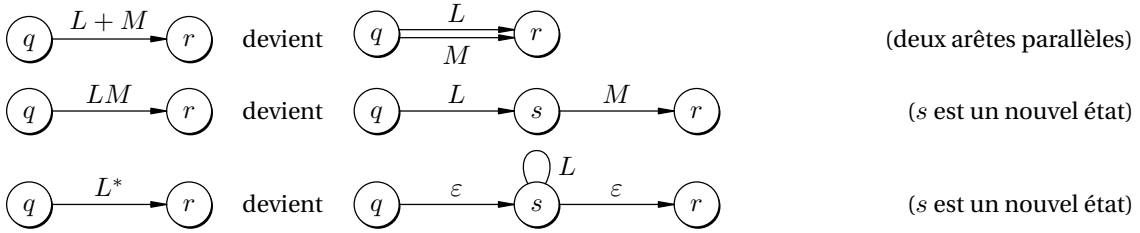
* Dans cette construction, tous les ensembles sont traités comme des alphabets.

Le langage $\mathcal{L}(\mathbf{A}) \subseteq \mathcal{A}^*$ reconnu par un AFG $\mathbf{A}$ est défini par :

$$u \in \mathcal{L}(\mathbf{A}) \text{ ssi il existe } s \in I \text{ et } r \in F \text{ tels que } (s,u) \overset{*}{\vdash\!\!-} (r,\varepsilon).$$

Deux AFG sont équivalents s'ils reconnaissent le même langage.

a) Vérifier que l'application de chacune des opérations suivantes (où q et r sont des états qui peuvent être égaux) transforme un AFG en un AFG équivalent.



(Cette dernière transformation ne s'applique utilement que lorsque $L \neq \emptyset$!)

b) En déduire une construction d'un ε–AF reconnaissant un langage régulier défini par une expression régulière.

Exercice 20. Système d'équations associé à un ε–AF. (cf. exercices 1.20 et 1.21)

On associe à tout ε–AF $\mathbf{A} = (Q, \mathcal{A}, \delta, I, F)$ le GL $A : (Q+\varrho)^2 \to \mathcal{P}(\mathcal{A}^*)$ (où $\varrho \notin Q$ est un nouveau symbole) qui est défini par :

$$A(qr) = \{x \in \varepsilon + \mathcal{A} \mid r \in \delta(q,x)\} \text{ pour tout } qr \in Q^2,$$
$$A(q\varrho) = \begin{cases} \varepsilon & \text{pour tout } q \in F \\ \emptyset & \text{pour tout } q \in Q - F, \end{cases}$$
$$A(\varrho q) = \emptyset \text{ pour tout } q \in Q + \varrho.$$

a) Vérifier que le GL ainsi défini est bien celui qui est associé au système d'équations correspondant à l'ε–AF $\mathbf{A}$ (cf. section 5.5 et exercice 1.21).

b) Pour vérifier les affirmations de la section 5.5 au sujet du calcul du langage reconnu par un ε–AF :

1) montrer que quels que soient $q \in Q$, $r \in Q$ et $u \in \mathcal{A}^*$ on a
$$(q,u) \overset{*}{\vdash\!\!-} (r,\varepsilon) \text{ ssi } u \in A(Chem(q,r)),$$
2) en déduire que $\mathcal{L}(\mathbf{A}) = A(IQ^*\varrho)$.

Remarque. Après l'exercice 1.21, ceci signifie que le langage engendré par un ε–AF se calcule, comme pour les AFDC, en utilisant la plus petite solution du système qui lui est associé : on doit, bien entendu, tenir compte du fait qu'un ε–AF peut avoir plusieurs entrées.

3
Les grammaires et les langages algébriques.

Lorsqu'aucune autre précision n'est donnée, G désigne une grammaire $(\mathcal{V}, \mathcal{A}, R)$.

Exercice 1.

Montrer que les langages suivants sont algébriques (a et b sont distincts, $\mathcal{A}$ est un alphabet fini quelconque) :

$$\begin{aligned} L_1 &= \{a^m b^m \mid m \geq 0\} \\ L_2 &= \{a^m b^n \mid 0 \leq m < n\} \\ L_3 &= \{a^m b^n a^n \mid m \geq 0, n \geq 0\} \\ L_4 &= \{uc\tilde{u} \mid u \in \mathcal{A}^*\} \\ L_5 &= \{uv \mid u \in (a+b)^*, v \in (a+b)^*, |u| = |v|, v \neq \tilde{u}\} \end{aligned}$$

Exercice 2.

Montrer que si $L \subseteq \mathcal{A}^*$ est algébrique alors $\widetilde{L} = \{u \in \mathcal{A}^* \mid \tilde{u} \in L\}$ l'est aussi.

Exercice 3. Deux fois plus de a que de b dans le même mot.

Soit $G = (S, a+b, R)$ la grammaire pour laquelle R est défini par la règle globale

$$S \longrightarrow \varepsilon + SaSaSbS + SaSbSaS + SbSaSaS.$$

Le but de cet exercice est de montrer que $\mathcal{L}(G,S)$ est l'ensemble L des $u \in \mathcal{A}^*$ vérifiant $|u|_a = 2|u|_b$.

a) Montrer que $\mathcal{L}(G,S) \subseteq L$ en effectuant une induction sur la longueur des dérivations.

b) Montrer que si $u \in L - \varepsilon$ alors u a un facteur $xyz \in L$ de longueur 3.

Indication.

Tout $u \in L - \varepsilon$, sans aucun facteur $xyz \in L$ de longueur 3, devrait satisfaire les propriétés suivantes :

- $|u| \neq 3$;
- u ne comporte aucun facteur aa, d'où plus généralement que $|u|_a \leq |u|_b + 1$;

ce qui est difficilement conciliable avec le fait que $u \in L - \varepsilon$!

c) Pour tout $u \in \mathcal{A}^*$ on définit $\overline{u} \in (\mathcal{A}+S)^*$, de la façon suivante :

- $\overline{\varepsilon} = \varepsilon$,
- $\overline{x} = x$ pour tout $x \in \mathcal{A}$,
- $\overline{ux} = \overline{u}Sx$ pour tout $u \in \mathcal{A}^* - \varepsilon$, et pour tout $x \in \mathcal{A}$.

$\overline{u}$ est donc obtenu en intercallant S entre les lettres de u, par exemple $\overline{u} = aSaSb$ pour $u = aab$.

Vérifier que $\overline{uv} = \overline{u}S\overline{v}$ lorsque $u \neq \varepsilon$ et $v \neq \varepsilon$.

d) Montrer les deux propriétés suivantes :

- $\overline{u} \stackrel{*}{\Longrightarrow} u$ pour tout $u \in \mathcal{A}^*$.
- $S \stackrel{*}{\Longrightarrow} S\overline{u}S$ pour tout $u \in L - \varepsilon$ (utiliser **b)** pour faire une induction).

e) En déduire que $L \subseteq \mathcal{L}(G,S)$.

Exercice 4. Les opérations régulières.

a) Montrer que si L_1 et L_2 sont algébriques, alors $L_1 + L_2$ et $L_1 L_2$ le sont aussi.

b) Montrer que si L est algébrique, alors L^* l'est aussi.

La classe des langages algébriques est donc stable par les opérations régulières.

c) En déduire que les langages réguliers sont algébriques.

Remarque. Ce résultat a déjà été obtenu par la considération des grammaires linéaires à droite. La réciproque n'est pas vraie, par exemple : $L = \{a^m b^m \mid m \geq 0\}$ est algébrique mais **il n'est pas** régulier.

d) Construire une grammaire engendrant le langage $a^2(a + ab + ba)^*$ à partir de l'une de ses variables.

Exercice 5. Grammaires linéaires à droite.

Les grammaires linéaires à droite que nous avons utilisées dans la section 1.3 étaient très particulières. En général, on appelle ainsi une grammaire dont chaque règle est de l'une des formes suivantes :

$$X \longrightarrow uY \qquad X \longrightarrow u$$

où X et Y sont des variables et $u \in \mathcal{A}^*$.

Montrer que pour toute grammaire G linéaire à droite en ce sens, il existe une grammaire $G' = (\mathcal{V}', \mathcal{A}, R')$ linéaire à droite au sens de la section 1.3, telle que $\mathcal{V} \subseteq \mathcal{V}'$ et $\mathcal{L}(G', X) = \mathcal{L}(G, X)$ pour toute $X \in \mathcal{V}$.

Exercice 6. Image d'un langage algébrique par une substitution.

Soient $\mathcal{A}$ et $\mathcal{B}$ deux alphabets finis et $f : \mathcal{A} \to \mathcal{P}(\mathcal{B}^*)$ une substitution telle que $f(x)$ est algébrique pour chaque $x \in \mathcal{A}$.

Montrer que si $L \subseteq \mathcal{A}^*$ est algébrique alors $f(L) \subseteq \mathcal{B}^*$ l'est aussi.

Exercice 7.

Montrer que si L est algébrique alors $fg(L)$, $fd(L)$ et $fact(L)$ le sont aussi.

Appliquer les constructions utilisées pour faire ces preuves au langage $L = \{a^m c\, b^m \mid m \geq 0\}$, où a, b et c sont deux à deux distincts.

Exercice 8.

Mettre la grammaire $G = (A + B + C + D + E, a + b, R)$ dont les règles globales sont :

$$\begin{aligned} A &\longrightarrow aAb + a \\ B &\longrightarrow Ab + bCC \\ C &\longrightarrow DD + ba \\ D &\longrightarrow aDB \\ E &\longrightarrow bC \end{aligned}$$

sous une forme réduite successivement pour chacune de ses variables.

Exercice 9.

Mettre la grammaire $G = (A + B + C + D + E + F, a + b, R)$ dont les règles globales sont :

$$\begin{aligned} A &\longrightarrow BC + DE + F \\ B &\longrightarrow BB + E \\ C &\longrightarrow aC + D \\ D &\longrightarrow C + EFaD \\ E &\longrightarrow aE + \varepsilon \\ F &\longrightarrow E + b \end{aligned}$$

sous une forme propre.

Exercice 10. Des grammaires généralisées.

Dans la définition d'une grammaire G, une règle globale pour $X \in \mathcal{V}$ est de la forme $X \longrightarrow \mathbf{l}(X)$ où $\mathbf{l}(X) \subseteq (\mathcal{V}+\mathcal{A})^*$ est supposé **fini**.
On dira que G est une *grammaire généralisée* si l'on suppose seulement que $\mathbf{l}(X)$ est algébrique pour chaque $X \in \mathcal{V}$.

Montrer que les langages engendrés par une grammaire généralisée sont algébriques. Appliquer ce résultat à un exemple simple.

Exercice 11.

Soit $X \in \mathcal{V}$ telle que $\mathcal{L}(G,X)$ soit **fini**.

a) Construire une grammaire $G' = (\mathcal{V}-X, \mathcal{A}, R')$ telle que pour toute $Y \in \mathcal{V}-X$ on ait $\mathcal{L}(G',Y) = \mathcal{L}(G,Y)$.

b) En déduire que si L est un langage algébrique infini alors il existe une grammaire G telle que $\mathcal{L}(G,X)$ est infini pour toute $X \in \mathcal{V}$.

Exercice 12.

Soit $L \subseteq \mathcal{A}^*$ un langage sur $\mathcal{A}$. Une nouvelle lettre $c \notin \mathcal{A}$ étant donnée, on définit $M \subseteq (\mathcal{A}+c)^*$ par

$$v \in M \text{ ssi il existe } u \in L \text{ tel que } v = uc\tilde{u}$$

où $\tilde{u}$ désigne l'image miroir de u.

Montrer que si L est régulier alors M est algébrique.

Exercice 13.

On se propose de démontrer la réciproque de l'énoncé de l'exercice 12.
soient c, L et M sont comme dans l'exercice 12 :

$$\text{si } M \text{ est algébrique alors } L \text{ est régulier.}$$

Lorsque M est fini, L l'est aussi et donc est régulier, sinon, soient $G = (\mathcal{V}, \mathcal{A}+c, R)$ une grammaire et $S \in \mathcal{V}$ telles que $M = \mathcal{L}(G,S)$. On suppose que

(I) G est réduite par rapport à S,
(II) pour toute $X \in \mathcal{V}$, le langage $\mathcal{L}(G,X)$ est infini,

(on sait, grâce à la propriété de réduction des grammaires et l'exercice 11, que ces hypothèses ne sont pas restrictives) ;
de plus, il est utile de considérer $\mathcal{W} \subseteq \mathcal{V}$ défini par

$$X \in \mathcal{W} \text{ ssi il existe } \alpha, \beta \in (\mathcal{A}+\mathcal{V})^* \text{ tels que } X \stackrel{*}{\Longrightarrow} \alpha c \beta$$

et de poser $\mathcal{V}' = \mathcal{V} - \mathcal{W}$.

a) Supposons qu'il existe λ et $\mu \in (\mathcal{A}+c+\mathcal{V})^*$ et $\xi \in c+\mathcal{W}$ tels que $S \stackrel{*}{\Longrightarrow} \lambda\xi\mu$.

1) Utiliser (I) pour montrer que $\lambda \in (\mathcal{A}+\mathcal{V}')^*$ (un argument symétrique montrerait que l'on a aussi $\mu \in (\mathcal{A}+\mathcal{V}')^*$).
2) Utiliser (II) pour montrer qu'en fait on a $\lambda \in \mathcal{A}^*$ (un argument symétrique montrerait que l'on a aussi $\mu \in \mathcal{A}^*$).

b) En déduire que toute règle de G est de la forme $X \longrightarrow u\xi v$ où $u, v \in \mathcal{A}^*$ et $\xi \in c+\mathcal{W}$, et en particulier que $\mathcal{W} = \mathcal{V}$.

c) Construire une grammaire linéaire à droite $H = (\mathcal{V}, \mathcal{A}, \Sigma)$ telle que $\mathcal{L}(H,S) = L$.

Exercice 14.

L'exercice précédent se généralise facilement et utilement, de la façon suivante.

Soit $c \notin \mathcal{A}$ une nouvelle constante. Soient $L \subseteq \mathcal{A}^*$ un langage et $f : \mathcal{A}^* \to \mathcal{P}(\mathcal{A}^*)$ une application telle que $f(v)$ est fini pour tout $v \in \mathcal{A}^*$. Considérons maintenant $M \subseteq \mathcal{A}^* c \mathcal{A}^*$ (chaque élément de M comporte exactement une occurrence de c) le langage défini par

$$ucv \in M \text{ ssi } v \in L \text{ et } u \in f(v)$$

pour tout $u, v \in \mathcal{A}^*$.

a) Montrer que si L est régulier alors M est algébrique.

b) Réciproquement, montrer que si M est algébrique alors L est régulier.

Indication. La méthode de l'exercice précédent fait encore merveille.

Remarque. L'hypothèse sur f est vérifiée par les applications de ce type qui ont été considérées dans le premier chapitre ($Conj$, fg, ...) et par les substitutions finies (celles qui envoient chaque caractère sur un ensemble fini de mots).

Exercice 15.

$X \in \mathcal{V}$ est dite *récursive* ssi il existe $\alpha \in (\mathcal{A}+\mathcal{V})^*$ et $\beta \in (\mathcal{A}+\mathcal{V})^*$ tels que $\alpha\beta \neq \varepsilon$ et $X \overset{*}{\underset{G}{\Longrightarrow}} \alpha X \beta$.

a) Montrer que si G est propre et si toutes ses variables sont productives, alors les deux propriétés suivantes sont vraies pour toute $X \in \mathcal{V}$:

1) si X est récursive, alors $\mathcal{L}(G, X)$ est infini.
2) $\mathcal{L}(G, X)$ est infini ssi il existe $Y \in \mathcal{V}$ qui est récursive et accessible à partir de X.

b) Déduire de 2) ci–dessus que si L est un langage algébrique infini, il existe une grammaire G et $X \in \mathcal{V}$ telles que

- $\mathcal{L}(G, X) = L - \varepsilon$,
- toute $Y \in \mathcal{V}$ est récursive.

Exercice 16.

Soit $G = (S, a+b, R)$ la grammaire pour laquelle R est constitué de la règle globale $S \longrightarrow aSS + b$.

Montrer que $\mathcal{L}(G, S)$ est le plus petit $X \subseteq \mathcal{A}^*$ satisfaisant $X = aXX + b$.

Exercice 17.

L'objet de cet exercice est de prouver que l'intersection d'un langage algébrique et d'un langage régulier est algébrique.

Il faut marier la carpe et le lapin ou plutôt une grammaire $G = (\mathcal{V}, \mathcal{A}, R)$ et un AFDC $\mathbf{A} = (Q, \mathcal{A}, \bullet, q_0, F)$; pour éviter toute incompatibilité, on supposera que $Q \cap (\mathcal{A}+\mathcal{V}) = \emptyset$.

On construit une grammaire $H = (\mathcal{W}, \mathcal{A}, \Sigma)$ de la façon suivante :

- $\mathcal{W} = Q \times \mathcal{V} \times Q$: on désignera le symbole $(q, X, r) \in \mathcal{W}$ par $[qXr]$.

Cette notation s'étend de la façon suivante : pour tout q et $r \in Q$ et pour tout $\alpha \in (\mathcal{A}+\mathcal{V})^*$, on définit l'ensemble fini $[q\alpha r] \subseteq (\mathcal{A}+\mathcal{W})^*$ en posant tout d'abord, pour tout $x \in \mathcal{A}$,

- $[qxr] = \begin{cases} x & \text{si } q \bullet x = r \\ \emptyset & \text{sinon} \end{cases}$

puis, la récurrence :

- $[qr] = \begin{cases} \varepsilon & \text{si } q = r \\ \emptyset & \text{sinon} \end{cases}$
- $[q\alpha\xi r] = \sum_{s \in Q} [q\alpha s][s\xi r]$ pour tout $\alpha \in (\mathcal{A}+\mathcal{V})^*$ et tout $\xi \in \mathcal{A}+\mathcal{V}$.

- si $X \underset{G}{\longrightarrow} \mathbf{l}(X)$ est la règle globale de X dans G, alors $[qXr] \underset{H}{\longrightarrow} [q\mathbf{l}(X)r]$ est la règle globale de $[qXr] \in \mathcal{W}$ dans H (on a étendu la définition précédente aux langages!).

a) Appliquer cette construction à un exemple simple.

b) Montrer que pour tout $[qXr] \in \mathcal{W}$ et tout $u \in \mathcal{A}^*$: $[qXr] \overset{*}{\underset{H}{\Longrightarrow}} u$ ssi $X \overset{*}{\underset{G}{\Longrightarrow}} u$ et $q \bullet u = r$.

c) En déduire que l'intersection d'un langage algébrique et d'un langage régulier est algébrique.

Remarque. Ceci implique que $L - M$ est algébrique lorsque L est algébrique et M régulier, car le complémentaire d'un langage régulier est régulier (cf. la section 6.1 du chapitre 2).

Attention. L'intersection de deux langages algébriques n'est pas nécessairement algébrique. Par exemple $L = \{a^m b^m a^n \mid m \geq 0, n \geq 0\}$ et $M = \{a^m b^n a^n \mid m \geq 0, n \geq 0\}$ sont algébriques (cf. exercice 1) mais on verra que $L \cap M = \{a^m b^m a^m \mid m \geq 0\}$ n'est pas algébrique (cf. exercice 19).

Exercice 18.

Montrer que si L est algébrique et M régulier alors $M^{-1}L$ est algébrique (cf. exercice 1.12 et la section 6.2.2 du chapitre 2).

Exercice 19.

Montrer que le langage $L = \{a^m b^m a^m \mid m \geq 0\}$, où a et b sont distinctes, n'est pas algébrique.

Exercice 20. Langages algébriques sur un alphabet à une seule lettre.

Soient a une lettre et $L \subseteq a^*$ un langage algébrique. Désignons par k un entier dont le lemme d'itération (section 5.1) assure l'existence à propos de L et posons $n = k!$ (tout entier i tel que $0 < i \leq k$ divise n).

a) Déduire du lemme d'itération que, pour tout $u \in L$, $|u| \geq k$ implique $u(a^n)^* \subseteq L$.

b) On pose $L_i = L \cap a^{k+i}(a^n)^*$ pour tout entier i tel que $0 < i \leq k$.
Montrer que si $L_i \neq \emptyset$ alors il existe $u_i \in \mathcal{A}^*$ tel que $L_i = u_i(a^n)^*$.

c) En déduire que L est régulier.

Exercice 21. Une méthode de dérécursion à gauche.

Soit $A \in \mathcal{V}$.
La règle globale de A peut s'écrire $A \underset{G}{\longrightarrow} A\mathbf{p} + \mathbf{q}$ avec $\mathbf{p} \subseteq (\mathcal{A} + \mathcal{V})^*$ et $\mathbf{q} \subseteq (\mathcal{A} + \mathcal{V})^* - A(\mathcal{A} + \mathcal{V})^*$ (les éléments de $\mathbf{q}$ ne commencent pas par la variable A). On peut considérer la grammaire généralisée (cf. exercice 10) $H = (\mathcal{V}, \mathcal{A}, \Sigma)$ dont les règles globales sont identiques à celles de G sauf pour la variable A dont la règles globale est maintenant $A \underset{H}{\longrightarrow} \mathbf{q}\mathbf{p}^*$.

a) Montrer que pour toute $X \in \mathcal{V}$ on a $\mathcal{L}(H, X) = \mathcal{L}(G, X)$.

b) Construire une grammaire (non généralisée) équivalente à H (on peut s'inspirer de l'exercice 4 ou de l'exercice 10).

Exercice 22. Dérécursion à gauche par une méthode matricielle.

Le second membre de la règle globale $X \longrightarrow \mathbf{l}(X)$ de toute $X \in \mathcal{V}$ peut s'écrire sous la forme

$$\mathbf{l}(X) = \sum_{Y \in \mathcal{V}} Y\mathbf{p}(YX) + \mathbf{q}(X)$$

où $\mathbf{p}(YX) \subseteq (\mathcal{A} + \mathcal{V})^*$ pour toute $Y \in \mathcal{V}$ et où $\mathbf{q}(X) \subseteq (\mathcal{A} + \mathcal{V})^* - \mathcal{V}(\mathcal{A} + \mathcal{V})^*$ est l'ensemble des éléments de $\mathbf{l}(X)$ qui ne commencent pas par une variable.

La dérécursion à gauche (cf. section 5.3 et exercice précédent) ne s'intéresse qu'à l'élimination des appels à gauche par une variable dans ses propres règles. La décomposition de $\mathbf{l}$ ci–dessus met en évidence tous les appels à gauche et permet de traiter cette question de façon globale.

Pour ce faire, on considère l'alphabet $\mathcal{W} = \mathcal{V} \times \mathcal{V}$ (on notera $[XY]$ le couple $(X, Y) \in \mathcal{W}$), la substitution $\mathbf{m} : \mathcal{V} + \mathcal{W} \to \mathcal{P}((\mathcal{A} + \mathcal{V} + \mathcal{W})^*)$ définie par

- $\mathbf{m}(X) = \sum_{Y \in \mathcal{V}} \mathbf{q}(Y)[YX] + \mathbf{q}(X)$ pour toute $X \in \mathcal{V}$,
- $\mathbf{m}([XY]) = \sum_{Z \in \mathcal{V}} \mathbf{p}(XZ)[ZY] + \mathbf{p}(XY)$ pour toute $[XY] \in \mathcal{W}$,

et enfin, $H = (\mathcal{V} + \mathcal{W}, \mathcal{A}, \Sigma)$ la grammaire dont les règles globales sont définies par $\mathbf{m}$.

a) Appliquer cette construction à la grammaire sur $\mathcal{A} = a + b$ et $\mathcal{V} = A_1 + A_2 + A_3$ dont les règles globales sont :

$$\begin{aligned} A_1 &\longrightarrow A_2A_3 + a \\ A_2 &\longrightarrow A_3A_1 + A_1b \end{aligned}$$

$$A_3 \longrightarrow A_1A_2 + A_2A_1 + b$$

b) Montrer que l'on a $\mathcal{L}(H, X) = \mathcal{L}(G, X)$ pour toute $X \in \mathcal{V}$.

c) Montrer comment on peut utiliser cette transformation pour mettre une grammaire propre sous une forme normale de Greibach. Appliquer cette methode à la grammaire ci–dessus (comparer au résultat de la section 5. sur cette même grammaire).

d) Montrer, en utilisant une forme normale de Chomsky que, pour toute grammaire $G = (\mathcal{V}, \mathcal{A}, R)$ qui ne produit pas ε, il existe une grammaire équivalente (en un sens à préciser) dont chacune des règles a l'une des formes suivantes :

$$X \longrightarrow xYZ \qquad X \longrightarrow xY \qquad X \longrightarrow x$$

où $x \in \mathcal{A}$ et X, Y et Z sont des variables.

(Il manque la notion d'automate à pile : des cas particuliers utiles sont considérés au chapitre 4.)

4
Analyse syntaxique.

Exercice 1.

Montrer que la grammaire suivante est $LR(0)$:

$1 : S \longrightarrow CC$ $\quad$ $2 : C \longrightarrow \mathbf{c}\, C$ $\quad$ $3 : C \longrightarrow \mathbf{d}$

Exercice 2.

Montrer que la grammaire suivante est $LR(0)$:

$1 : S \longrightarrow \mathbf{f}\, SS$ $\quad$ $2 : S \longrightarrow \mathbf{g}\, S$ $\quad$ $3 : S \longrightarrow \mathbf{a}$

Faire l'analyse de **g f f a g a g a** et construire l'arbre de dérivation correspondant.

Exercice 3.

La grammaire suivante est–elle $LR(0)$? $SLR(1)$?

$1 : S \longrightarrow SS\,\mathbf{f}$ $\quad$ $2 : S \longrightarrow S\,\mathbf{g}$ $\quad$ $3 : S \longrightarrow \mathbf{a}$

Faire l'analyse de **a g a g a f f g** et construire l'arbre de dérivation correspondant.

Exercice 4.

Construire la table $SLR(1)$ de la grammaire suivante :

$1 : E \longrightarrow TF$ $\quad$ $2 : F \longrightarrow \varepsilon$ $\quad$ $4 : T \longrightarrow (E)$
$3 : F \longrightarrow \oplus TF$ $\quad$ $5 : T \longrightarrow$ **id**

La grammaire en question est–elle $SLR(1)$? Si oui, utiliser la table pour faire l'analyse de **id** $\oplus$ (**id**) et construire l'arbre de dérivation correspondant.

Exercice 5.

Voici quelques grammaires sur la nature desquelles vous pourrez vous interroger. Au passage, vérifiez que, pour toute grammaire G :

- si G est $LR(0)$ alors G est $SLR(1)$,
- si G est $SLR(1)$ alors G est $LALR(1)$,
- si G est $LALR(1)$ alors G est $LR(1)$.

G_5 : $1 : S \longrightarrow \mathbf{a}\, A\, \mathbf{c}$ $\quad$ $2 : A \longrightarrow A\, \mathbf{b}\, \mathbf{b}$ $\quad$ $3 : A \longrightarrow \mathbf{b}$

G_6 : $1 : A \longrightarrow \mathbf{a}\, S$ $\quad$ $2 : S \longrightarrow \mathbf{b}\, S$ $\quad$ $3 : S \longrightarrow \mathbf{a}\, \mathbf{a}\, \mathbf{b}$

G_7 : $1 : A \longrightarrow BA$ $\quad$ $2 : A \longrightarrow \mathbf{a}$ $\quad$ $3 : B \longrightarrow AB$ $\quad$ $4 : B \longrightarrow \mathbf{b}$

G_8 : $1 : S \longrightarrow S\, \mathbf{a}\, S\, \mathbf{b}$ $\quad$ $2 : S \longrightarrow \varepsilon$

G_9 : $1 : S \longrightarrow AB$ $\quad$ $2 : A \longrightarrow \mathbf{a}\, A\, \mathbf{b}$ $\quad$ $4 : B \longrightarrow \mathbf{b}\, B$
$3 : A \longrightarrow \varepsilon$ $\quad$ $5 : B \longrightarrow \mathbf{b}$

G_{10} : $1 : S \longrightarrow AB$ $\quad$ $3 : B \longrightarrow CD$ $\quad$ $5 : C \longrightarrow \mathbf{a}\, \mathbf{b}$ $\quad$ $7 : E \longrightarrow \mathbf{b}\, \mathbf{b}\, \mathbf{a}$
$2 : A \longrightarrow \mathbf{a}$ $\quad$ $4 : B \longrightarrow \mathbf{a}\, E$ $\quad$ $6 : D \longrightarrow \mathbf{b}\, \mathbf{b}$

G_{11} : $1 : S \longrightarrow S \oplus A$ $\quad 2 : S \longrightarrow A$ $\quad 3 : A \longrightarrow (S)$ $\quad 4 : A \longrightarrow \mathbf{a}\,(S)$ $\quad 5 : A \longrightarrow \mathbf{a}$

G_{12} : $1 : S \longrightarrow \mathbf{a}\,D\,\mathbf{;}\,I\,\mathbf{b}$ $\quad 2 : D \longrightarrow D\,\mathbf{;}\,\mathbf{d}$ $\quad 3 : D \longrightarrow \mathbf{d}$ $\quad 4 : I \longrightarrow \mathbf{i}\,\mathbf{;}\,I$ $\quad 5 : I \longrightarrow \mathbf{i}$

Utilisation de grammaires ambiguës.

Les langages de programmation permettent l'usage de quelques ambiguïtés. Les programmes y gagnent en simplicité mais ne seraient pas analysables par une méthode déterministe si certaines conventions n'étaient pas posées, par exemple : il est en général convenu que l'expression **id** + **id** + **id** sera calculée comme (**id** + **id**) + **id** .

Il existe deux méthodes pour faire l'analyse syntaxique de telles expressions :

1) On utilise une grammaire ambiguë, qui suit exactement la syntaxe des espressions en question : les conflits, qui se présentent inévitablement dans la table d'analyse, sont résolus en choisissant (une fois pour toute!) dans chaque état de l'AFD, l'action qui correspond à la façon dont on prétend faire l'analyse. Yacc est capable de faire ce choix, dans des cas simples, lorsqu'on lui donne des informations sur l'associativité (ou la non associativité) des opérations et sur leur préséance relative.
   Cette opération consiste en fait à sélectionner certaines dérivations, parmi toutes celles qui sont possibles : on court ainsi le risque de ne plus pouvoir analyser des phrases qui sont pourtant dans le langage engendré par la grammaire!
2) On utilise une grammaire non ambiguë qui est capable de faire l'analyse correctement. Cette méthode paraît plus saine que la précédente mais, sa mise en œuvre nécessite une grammaire plus complexe, en particulier comportant plus de variables; les analyses dans une telle grammaire sont souvent beaucoup plus longues que dans la méthode 1).

Les deux exercices qui suivent ont pour but de montrer comment on pratique la première méthode. La seconde méthode est seulement illustrée par la donnée d'une grammaire permettant sa mise en œuvre : les vérifications utiles sont laissées à votre initiative personnelle!

Exercice 6. Ambiguïté des expressions arithmétiques.

Calculer la table $SLR(1)$ de la grammaire

$1 : E \to E \oplus E$ $\qquad 2 : E \to E * E$ $\qquad 3 : E \to (E)$ $\qquad 4 : E \to$ **id**

puis résoudre les conflits que l'on peut y observer de telle façon que $\oplus$ et $*$ soient associatives à gauche et que $*$ ait une préséance supérieure à $\oplus$.

(La grammaire

$E \to E \oplus T$ $\qquad T \to T * F$ $\qquad F \to (E)$
$E \to T$ $\qquad T \to F$ $\qquad F \to$ **id**

équivalente à la précédente, est $SLR(1)$ et prend en compte les conventions précédentes.)

Exercice 7. Ambiguïté du *"sinon en suspens"***.**

La grammaire suivante décrit les instruction conditionnelles :

$I \to$ **si** E **alors** I **sinon** I $\qquad I \to$ **si** E **alors** I $\qquad I \to$ **autre**

Pour l'étudier, nous en considérons la forme abrégée suivante :

$1 : I \to$ **i** I **e** I $\qquad 2 : I \to$ **i** I $\qquad 3 : I \to$ **a**

Calculer la table $SLR(1)$ de cette grammaire puis, résoudre le conflit que l'on peut y observer de telle façon que la règle habituelle soit respectée : "un **sinon** est associé au dernier **alors** en suspens", c'est–à–dire, par exemple, que la phrase **i i a e a** soit analysée comme **i** [**i a e a**].

Serait–il possible de faire un choix autre que celui qui vient d'être fait?

(La grammaire

$I \to J$	$J \to$ **i** J **e** J	$K \to$ **i** I
$I \to K$	$J \to$ **a**	$K \to$ **i** J **e** K

équivalente à la précédente, est $SLR(1)$ et prend en compte la convention précédente.)

Exercice 8. Récursion et pile d'analyse.

On considère les grammaires sur l'alphabet de terminaux composé des digits binaires **0** et **1** et du point "binaire" **.** pour représenter les rationnels en notation binaire et l'alphabet de variables $\mathcal{V} = S+E+D+B$ où S est choisie comme axiome :

G dont les règles sont

$1 : S \longrightarrow E$ **.** D	$3 : E \longrightarrow EB$	$5 : D \longrightarrow BD$	$7 : B \longrightarrow$ **0**
$2 : S \longrightarrow E$	$4 : E \longrightarrow B$	$6 : D \longrightarrow B$	$8 : B \longrightarrow$ **1**

et G' dont les règles sont celles de G à l'exception de 5 : qui est remplacée par 5' : $D \longrightarrow DB$.

Les grammaires sont assez simples et admettent une analyse ascendante : il est facile de simuler un analyseur de type LR pour traiter la question qui suit.

Faire une analyse LR de **1 . 0 0 1 1 1** dans les deux grammaires et comparer l'évolution des piles respectives.

(Il manque la définition intrinsèque des grammaires $LR(k)$)

Analyse LL.

(Il faudrait ajouter des exercices sur l'analyse descendante.)

5 Traduction et production de code intermédiaire.

L'objet de ces exercices est de construire, par exemple à l'aide de Lex et Yacc, un traducteur du langage *Galileo* en l'un des codes dont il a été question dans les généralités (P–code, Code à registres ou Code à trois adresses) et éventuellement, d'écrire un programme (dans votre langage favori) capable d'exécuter un tel code.

Présentation de la grammaire.

Galileo est, en français, un langage impératif très simple inspiré de Pascal, dont la sémantique est par conséquent connue de tous (deux points, inspirés de C sont signalés dans ce qui suit).

- La définition précise des unités lexicales est donnée à la fin, dans un style Lex.
- La grammaire exclut l'imbrication des procédures et des fonctions (ceci tient à l'absence de Dp dans les règles 8 et 9) ainsi, un nom ne peut–il être que local ou global : cet attribut est suffisant pour déterminer l'adresse d'une référence et le lien statique n'a pas lieu d'être explicité dans la pile d'exécution.
- L'usage du mot réservé **fin** permet d'éviter un cas classique d'ambiguïté (règles 22 et 23), plus généralement, **fin** joue le rôle d'une parenthèse fermante qui donne à la grammaire une structure très simple. Les "parenthèses ouvrantes" se présentent sous différents aspects, par exemple **Programme**, **var**, ...
- Les types autorisés sont les entiers et les tableaux d'entiers (on n'a donc pas jugé utile de mentionner **entier** dans les règles 9 et 15). En réalité, l'emploi du type tableau est limité à la définition des listes d'identificateurs (règles 4 et 5) et donc, des listes de paramètres (règle 13) et de certaines expressions. L'introduction d'une expression de type tableau ne peut se produire que par application de la règle 39, mais il n'est pas question de faire des calculs sur un tel objet (les calculs sur un tableau se font sur ses composantes!)
- Pour en finir avec les tableaux : le domaine d'un tableau de type **tableau** `[ n ]` est l'ensemble des entiers `0, ..., n - 1` comme en C.
- Comme en C, les entiers sont interprétés comme des booléens lorsque c'est nécessaire : 0 pour faux et $\neq 0$ pour vrai; la négation transforme respectivement ces valeurs en 1 et 0.

L'évaluation des booléens se fera par un calcul complet mais les instructions de branchement provenant des règles 22, 23 et 24 nécessitent l'emploi d'une méthode de reprise : il vous faudra marquer les terminaux "; ", **alors**, **sinon**, **tantque** et **faire** de façon convenable.

- **La portée est statique**.
- **Les paramètres sont tous transmis par valeur** et le langage ne comporte pas de pointeur.
- Une liste d'instructions ne peut pas être vide :
  - le retour d'une fonction se fait toujours par une instruction **retourne** (E),
  - le retour du programme ou d'une procédure se fait nécessairement par une instruction **retourne**

- Il peut arriver, pour certaines valeurs des paramètres, que l'exécution n'aboutisse à aucune instruction de retour : ceci est le fruit d'une erreur grave dans le programme mais qui n'est pas décelable pendant la traduction.
- Une autre erreur classique peut se produire dans la transmition des paramètres : lors de la traduction, il est nécessaire (et évidemment possible) de vérifier que le nombre et le type des paramètres utilisés dans un appel de fonction ou de procédure correspond bien à la déclaration de celle–ci.

La grammaire de Galileo

Programmes.

$1 : P \rightarrow \textbf{programme id } Dv\ Dp\ Li\ \textbf{fin}$

Déclarations de **v**ariables.

$2 : Dv \rightarrow \varepsilon$

$3 : Dv \rightarrow \textbf{var } Lid\ \textbf{fin}$

Listes d'**id**entificateurs.

$4 : Lid \rightarrow \textbf{id} : Ty$

$5 : Lid \rightarrow Lid\ ,\ \textbf{id} : Ty$

Déclarations de **p**rocédures et de fonctions.

$6 : Dp \rightarrow \varepsilon$
$7 : Dp \rightarrow Dp\ R$
$8 : R \rightarrow Pro\ (\ Lp\ )\ Dv\ Li\ \textbf{fin}$
$9 : R \rightarrow Fon\ (\ Lp\ )\ Dv\ Li\ \textbf{fin}$
$10 : Pro \rightarrow \textbf{procedure id}$
$11 : Fon \rightarrow \textbf{fonction id}$

Listes de **p**aramètres.

$12 : Lp \rightarrow \varepsilon$

$13 : Lp \rightarrow Lid$

Types.

$14 : Ty \rightarrow \textbf{entier}$

$15 : Ty \rightarrow \textbf{tableau } [\ \textbf{nb}\ ]$

Listes d'**i**nstructions.

$16 : Li \rightarrow I$

$17 : Li \rightarrow Li\ ;\ I$

Instructions.

$18 : I \rightarrow V\ :=\ E$
$19 : I \rightarrow \textbf{id}\ (\ Le\ )$
$20 : I \rightarrow \textbf{retourne}$
$21 : I \rightarrow \textbf{retourne}\ (\ E\ )$
$22 : I \rightarrow \textbf{si } E\ \textbf{alors } Li\ \textbf{fin}$
$23 : I \rightarrow \textbf{si } E\ \textbf{alors } Li\ \textbf{sinon } Li\ \textbf{fin}$
$24 : I \rightarrow \textbf{tantque } E\ \textbf{faire } Li\ \textbf{fin}$
$25 : I \rightarrow \textbf{ecrire}\ (\ E\ )$

Variables.

$26 : V \rightarrow \textbf{id}$

$27 : V \rightarrow \textbf{id}\ [\ E\ ]$

Listes d'**e**xpressions.

$28 : Le \rightarrow \varepsilon$
$29 : Le \rightarrow Le'$
$30 : Le' \rightarrow E$
$31 : Le' \rightarrow Le'\ ,\ E$

Expressions (expressions **S**imples, **T**ermes, **F**acteurs).

$32 : E \rightarrow S$
$33 : E \rightarrow S\ \textbf{oprel}\ S$
$34 : S \rightarrow T$
$35 : S \rightarrow S\ \textbf{opadd}\ T$
$36 : S \rightarrow S\ -\ T$
$37 : T \rightarrow F$
$38 : T \rightarrow T\ \textbf{opmult}\ F$
$39 : F \rightarrow \textbf{id}$
$40 : F \rightarrow \textbf{id}\ (\ Le\ )$
$41 : F \rightarrow \textbf{lire}\ (\ )$
$42 : F \rightarrow \textbf{id}\ [\ E\ ]$
$43 : F \rightarrow (\ E\ )$
$44 : F \rightarrow -\ F$
$45 : F \rightarrow \textbf{non}\ F$
$46 : F \rightarrow \textbf{nb}$

Les unités lexicales.

- Les mots clefs (**programme**, **var**, **procedure**, **fonction**, **entier**, **tableau**, **retourne**, **si**, **alors**, **sinon**, **tantque**, **faire**, **lire**, **ecrire**, **fin**) sont évidemment réservés et chacun d'eux peut constituer une unité lexicale à soi seul.
- $\mathbf{id} = [a-zA-Z][a-zA-Z0-9]*$ est l'unité lexicale des identificateurs : des chaînes non vides de chiffres et de lettres, commençant par une lettre.
- $\mathbf{nb} = [-+]?[0-9]+$ est l'unité lexicale des nombres : des chaînes non vides de chiffres, éventuellement précédées d'un signe.
- $\mathbf{oprel} = " = " \mid " <> " \mid " <= " \mid " < " \mid " >= " \mid " > "$
- $\mathbf{opadd} = " + " \mid \mathbf{ou}$
- $\mathbf{opmult} = " * " \mid \mathbf{et}$
- Chacun des autres symboles (le signe moins $" - "$, la négation **non**, l'opérateur d'affectation $" := "$, les parenthèses $"(", ")"$, $"[", "]"$ et les séparateurs $", ", "; ", " : "$) constitue une unité lexicale à lui seul.
- $\mathbf{bl} = [\ \backslash n \backslash t]+$ est l'unité lexicale des "blancs" : des chaînes constituées d'un nombre quelconque d'espaces, de fins de lignes et de tabulations. Ceux qui figurent dans la grammaire ne sont nécessaires qu'avant et après un mot clef, sauf si un symbole (parmi ceux qui sont décrits dans les points précédents) peut jouer le même rôle.
- $\mathbf{comm} = "\%" .* \backslash n$ est l'unité lexicale des commentaires : tout ce qui, dans une ligne, suit le symbole $\%$ est considéré comme un commentaire et doit être négligé dans l'analyse syntaxique (ceci explique que les commentaires ne figurent pas dans la grammaire précédente).

Exemple : P–code.

Ce type de code intermédiaire est intéressant car il met bien en evidence un mode très courant d'exécution d'un programme.

Un P–code est une liste de P–instructions, dont la plupart agissent directement sur la pile d'exécution, qui se présente généralement sous la forme d'un simple fichier.

- Une P–instruction comporte un opérateur (représenté par un mnémonique de quelques lettres, qui devra être codé par un entier) et éventuellement un argument (noté p dans les descriptions ci–dessous).
- On dispose de "registres" (qui pourront être implantés sous la forme de variables globales de catégorie `registre`) :

  `CP` (Code Pointer)
  `SP` (Stack Pointer)
  `MP` (Mark Pointer)
  `DP` (Data Pointer)
  `RCP` (Return CP).

- Le registre `CP` est l'adresse, dans le P–code, de l'instruction à exécuter (le déplacement à effectuer, en partant du début du fichier, pour atteindre le début de la P–instruction en question).

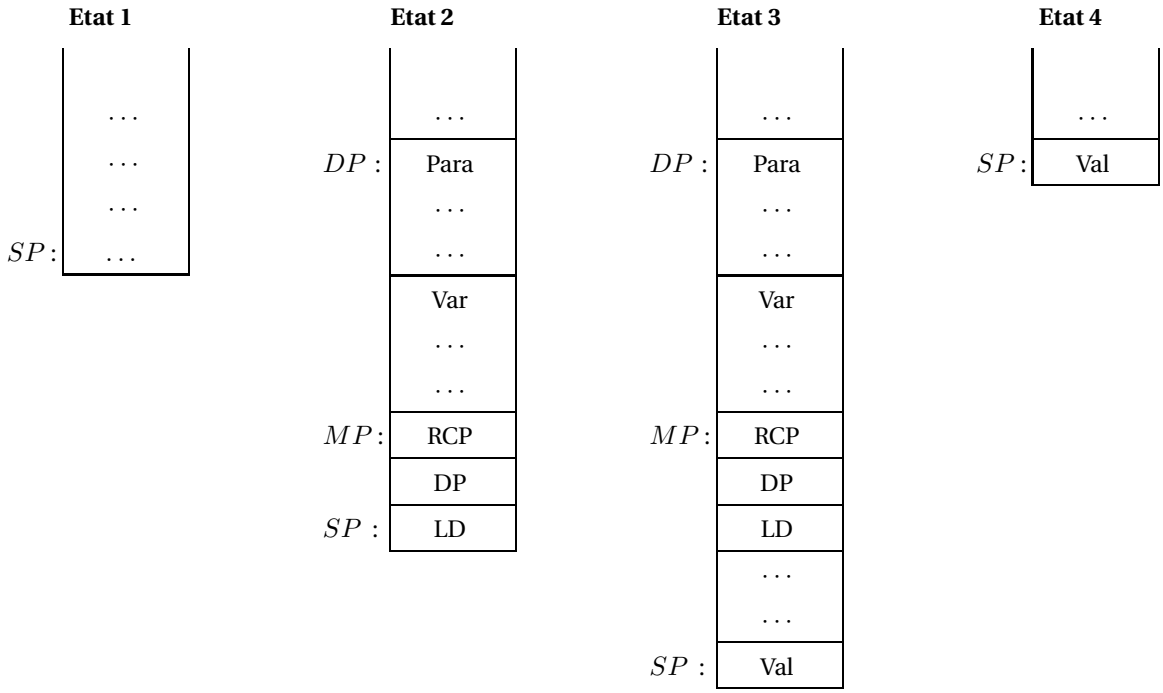
Les instructions d'un P–code s'exécutent séquentiellement, à partir de la première, sauf dans le cas des P–instructions qui peuvent déterminer un saut (`UJP`, `TJP`, `FJP`, `CALL`, `RETF`, `RETP`) et l'instruction de sortie pour cause d'erreur `ERREUR`; il faut donc incrémenter en conséquence le registre `CP` après l'exécution d'une P–instruction. Cette incrémentation peut être représentée schématiquement par `CP := CP + 1` mais il faut comprendre ce `1` comme étant la longueur de la P–instruction que l'on vient d'exécuter (il en est de même pour la variable globale `Indice` utilisée dans la traduction). Si l'on veut que l'incrémentation de `CP` ait toujours le même pas, il faut choisir de réserver un espace constant pour toute P–instruction (un nombre entier de mots machine).

- La valeur maximale de `CP` est désignée par $Codemax$

La pile d'exécution.

La pile d'exécution est désignée comme une partie initiale d'un tableau PILE : l'indice de son sommet est la valeur du registre SP.

- La valeur maximale de SP est désignée par $Maxstr$: cette valeur doit être choisie de façon judicieuse! Toute P–instruction incrémentant SP doit commencer par un test de non débordement de pile (qui n'est pas mentionné dans la liste qui est donnée plus bas).



Evolution de la pile lors d'un appel (1 → 2) et lors d'un retour (3 → 4) de fonction.

- Le principe d'exécution d'une opération sur une pile est bien connu :
  - les arguments de l'opération sont empilés au sommet, dans l'ordre correspondant à la déclaration de cette opération,
  - l'exécution de l'opération a pour effet global de dépiler les arguments et d'empiler le résultat, si l'opération en produit un.
- Ce principe s'étend au cas des fonctions et des procédures.
  - APPEL : les arguments sont les arguments proprement dits (valeur actuelle des paramètres) qui doivent être évalués avant l'appel de la fonction et les variables locales. Ceci est figuré par le passage de l'état 1 à l'état 2 :
    - La P–instruction PARA détermine le début du bloc des arguments proprement dits,
    - la P–instruction IEA achève l'installation de l'enregistrement d'activation en empilant le bloc réservé aux variables locales et le bloc des sauvegardes.
  - RETOUR : l'enregistrement d'activation est dépilé et une valeur de retour est empilée, dans le cas d'une fonction. Ceci est figuré par le passage de l'état 3 à l'état 4.
    Les P–instructions RETF et RETP se chargent de plus de réactualiser les registres SP, MP et CP.

Les P–instructions.

Dans la liste des P–instructions qui suit, la mention d'un type de base qui vient souvent qualifier l'opérateur, (E pour entier, B pour booléen, A pour adresse, ...) ne sera pas utile ici puisque les objets qui sont manipulés sont toujours des entiers.

Le test de non débordement de pile, qu'il est indispensable d'effectuer avant l'exécution de toute P–instruction qui incrémente SP, n'est pas mentionné.

Expressions arithmétiques.

```
ADD E : (E,E) → E.
            PILE[SP - 1] := PILE[SP - 1] +E PILE[SP] ;
            SP := SP - 1 ;
```

...

Expressions booléennes.

```
AND : (B,B) → B.
            PILE[SP - 1] := PILE[SP - 1] et PILE[SP] ;
            SP := SP - 1 ;
```

...

```
EQU T : (T,T) → B.
            PILE[SP - 1] := PILE[SP - 1] =T PILE[SP] ;
            SP := SP - 1 ;
```

...

Adressage direct des données.

```
LOC p : vide → A.
            SP := SP + 1 ;
            PILE[SP] := PILE[MP + 1] + p ;

GLO p : vide → A.
            SP := SP + 1 ;
            PILE[SP] := p ;
```

Adressage indexé des données.

```
IXA : (E,A) → A.
           PILE[SP - 1] := PILE[SP] + PILE[SP - 1] ;
            SP := SP - 1 ;
```

Test de domaine.

```
CHK p : E → E, Type(p) = E.
           si ( PILE[SP] < 0 ou PILE[SP] ≥ p ) alors
                     erreur('hors du domaine')
           fin ;
```

Lecture (Load) et écriture (Store) en mémoire.

LDO T p : vide $\rightarrow$ T, p $\in [0, Maxstr]$.

```
SP := SP + 1 ;
PILE[SP] := PILE[p] ;
```

LDC T p : vide $\rightarrow$ T, Type(p) = T.

```
SP := SP + 1 ;
PILE[SP] := p ;
```

SRO T p : T $\rightarrow$ vide, p $\in [0, Maxstr]$.

```
PILE[p] := PILE[SP] ;
SP := SP - 1 ;
```

STO T : (A, T) $\rightarrow$ vide.

```
PILE[PILE[SP - 1]] := PILE[SP] ;
SP := SP - 2 ;
```

Copie de blocs (tableaux).

MOV p : A $\rightarrow$ (T, . . . , T).

```
pour i décroissant de p - 1 à 0 faire
          PILE[SP + i] := PILE[PILE[SP] + i]
fin ;
```

Branchement.

UJP p : p $\in [0, Codemax]$.

```
CP := p ;
```

TJP p : B $\rightarrow$ vide, p $\in [0, Codemax]$.

```
si PILE[SP] alors CP := p fin ;
SP := SP - 1 ;
```

FJP p : B $\rightarrow$ vide, p $\in [0, Codemax]$.

```
si ( non PILE[SP] ) alors CP := p fin ;
SP := SP - 1 ;
```

Initialisation des registres.

INIT.

```
SP := -1 ;
MP := -1 ;
DP := 0 ;
RCP := -1 ;
```

Appel d'une procédure ou d'une fonction.

CALL p.

```
RCP := CP + 1 ;
CP := p ;
```

Installation d'un enregistrement d'activation.

```
PARA p.
          DP := SP - p + 1 ;

IEA p : vide → (U, ..., U, A, A, A).
          SP := SP + p + 3 ;
          PILE[SP - 2] := RCP ;
          PILE[SP - 1] := DP ;
          PILE[SP] := MP ;
          MP := SP - 2 ;
```

Retour de fonction ou de procédure.

```
RETF : 'EA' → T.
           CP := PILE[MP] ;
           PILE[PILE[MP + 1]] := PILE[SP] ;
           SP := PILE[MP + 1] ;
           MP := PILE[MP + 2] ;

RETP : 'EA' → vide.
           CP := PILE[MP] ;
           SP := PILE[MP + 1] - 1 ;
           MP := PILE[MP + 2] ;

ERREUR.
          erreur('sortie imprévue') ;
```

Entrées et sorties.

```
ECRIRE : E → vide.
          écrire l'entier PILE[SP] à l'écran ;
           SP := SP - 1 ;

LIRE : vide → E.
          lire un entier e à l'écran ;
           SP := SP + 1 ;
           PILE[SP] := e ;
```