Chapitre 6

Langages rationnels

Option informatique MP1, MP2 & MP3

Un des objectifs de ce chapitre est d'introduire un formalisme efficace nous permettant de décrire certains ensembles de mots (autrement dit des langages) que l'on peut être amené à rechercher dans un texte. La caractéristique de ces langages est la possibilité de les décrire par des motifs (patterns en anglais), c'est-à-dire par des formules qu'on appelle, dans ce contexte, expressions régulières.

I Recherche d'un mot

Le problème qui sert de cadre à ce chapitre est celui de la recherche d'un ou de plusieurs mots dans un texte. Dans le cas le plus simple on se donne un texte sous la forme d'une chaîne de caractères et une chaîne de caractère plus courte, le motif, dont on cherche les apparitions (les occurrences) dans le texte.

I.1 Algorithme simple

Nous avons déjà vu l'algorithme simple d'une telle recherche :

- si n est la longueur du texte et p celle de la chaîne recherchée
- on cherche, pour chaque position i possible (de 0 à $n-p$),
- si le mot cherché est égal à la portion de texte entre i et $i+p-1$.

```
let rechercheTexte motif texte =
   let n = String.length texte in
    let p = String.length motif in
   let resultat = ref [] in
   for i = 0 to (n - p) do
      if (sub_string texte i p) = motif
      then resultat := i :: !resultat done;
  !resultat;;
```

La fonction `sub_string` reçoit une chaîne de caractères `ch` et deux entiers `i` et `l` et renvoie la sous-chaîne de longueur `l` extraite de `ch` et commençant à l'indice `i`.

```
sub_string : string -> int -> int -> string
```

Exercice 1 - Complexité
Déterminer, en nombre de comparaisons de deux caractères, la complexité de `rechercheTexte` ; on négligera la complexité de `sub_string`.
Donner un exemple où la complexité atteint son maximum.

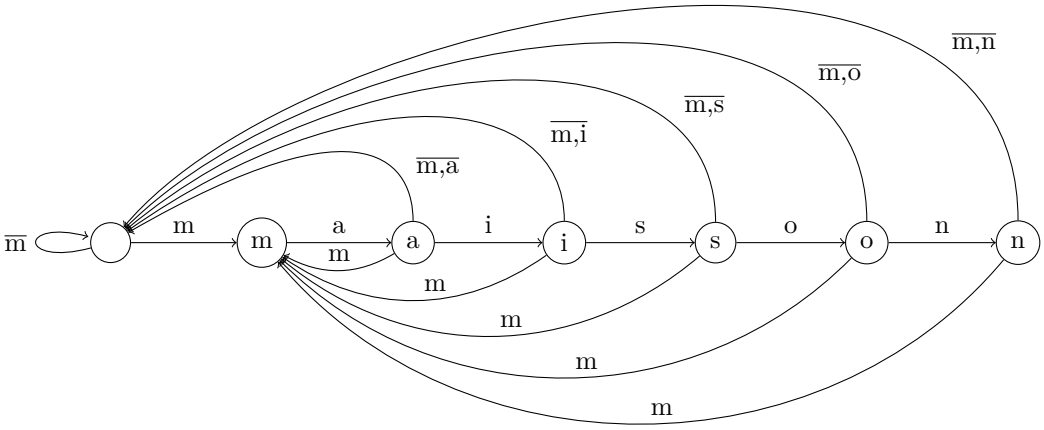
Cette complexité dans le pire des cas ne semble pas se produire souvent. On peut calculer une complexité moyenne, en faisant 2 hypothèses qui ne sont pas réalistes mais qui permettent de simplifier les calculs.

- On fait l'hypothèse que les caractères sont équiprobables : la probabilité que le i-ième caractère soit une certaine lettre est $\frac{1}{q}$ pour un alphabet de q lettres.
- On fait l'hypothèse que les caractères sont indépendants.

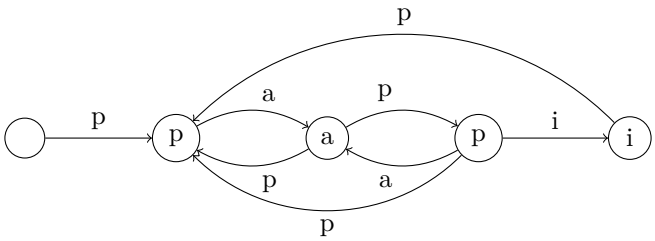
Exercice 2 - Complexité moyenne
Sous ces hypothèses que l'espérance du nombre de comparaisons de deux caractères dans `rechercheTexte` est majorée par $\frac{q-1}{q}n$.

I.2 Automates simples

La méthode ci-dessus ci consiste à faire glisser une fenêtre sur le texte et à comparer les lettres vues dans la fenêtre avec celles du motif.
Il existe une autre méthode qui lit le texte lettre par lettre et dans laquelle on garde en mémoire le nombre de lettres égales au motif auquel on est parvenu. Dans l'exemple ci-dessous les flèches indiquent les changement d'état et sont marquées par la lettre qui implique le changement. La notation $\overline{\alpha}$ indique toute lettre autre que α, de même $\overline{\alpha, \beta}$ indique les lettres qui ne sont ni α ni β.



L'automate peut être plus compliqué ; dans l'exemple suivant on n'indique pas les flèches vers l'état initial.



La situation se complique encore dans le cas de plusieurs mots.

II Langages rationnels

Nous allons, dans la suite de ce chapitre, caractériser des ensembles de mots que l'on peut définir simplement et dont le chapitre suivant montrera qu'ils peuvent être reconnus par des automates.

II.1 Mots et langages

Définition 1 - Alphabet

Un alphabet est un ensemble fini, ses éléments sont appelés **lettres**.

Dans les exemples, pour simplifier, nous considérerons souvent un alphabet à 2 lettres $\mathcal{A} = \{a, b\}$.

Définition 2 - Mot

Un mot sur $\mathcal{A}$ est une suite finie de lettres (on dit aussi chaîne de caractères) qu'on notera par la concaténation de ses lettres : $w = u_1 u_2 \cdots u_n$ avec $u_i \in \mathcal{A}$.
L'ensemble des mots sur un alphabet $\mathcal{A}$ est noté $\mathcal{A}^+$.

Définition 3 - Longueur

Le nombre de lettres d'un mot w est sa longueur, notée $|w|$.
L'ensemble des mots de longueur p se note $\mathcal{A}^p$.
Le nombre d'occurrences d'une lettre a dans un mot w est noté $|w|_a$.

On a donc $\mathcal{A}^+ = \bigcup_{p \in \mathbb{N}^*} \mathcal{A}^p$ et $|w| = \sum_{a \in \mathcal{A}} |w|_a$.

Définition 4 - Mot vide

On introduit le mot vide, il ne contient aucune lettre, sa longueur est nulle.
Il sera noté ε ou $1_{\mathcal{A}}$ (ou parfois Λ).
On note $\mathcal{A}^0 = \{\varepsilon\}$ et $\mathcal{A}^* = \mathcal{A}^+ \cup \{\varepsilon\} = \bigcup_{p \in \mathbb{N}} \mathcal{A}^p$.

Le mot de longueur n ne comportant que la lettre x peut être noté x^n.
En particulier $x^0 = \varepsilon$ et $x^1 = x$.

Définition 5 - Langage

Un langage est un ensemble de mots (qui peut contenir ε) , c'est une partie de $\mathcal{A}^*$.
Les **langages élémentaires** sur un alphabet $\mathcal{A}$ sont :

- l'ensemble vide, $\emptyset$,
- le langage réduit au mot vide, $\{\varepsilon\}$,
- les singletons $\{x\}$ pour toute lettre $x \in \mathcal{A}$.

II.2 Produit

Définition 6 - Produit

Le produit de deux mots est le mot obtenu par la concaténation de leurs lettres : si $u = x_1 \dots x_n$ et $v = y_1 \dots y_m$ alors $w = u.v = z_1 \cdots z_{m+n}$ avec $z_i = x_i$ pour $1 \leqslant i \leqslant n$ et $z_i = y_{i-n}$ pour $n+1 \leqslant i \leqslant n+m$.
On pose aussi $w.\varepsilon = \varepsilon.w = w$.
Si $w = u.v$ on dit que u est un **préfixe** de w et v est un **suffixe** de w.
Si $w = u.v.u'$ on dit que v est un **facteur** de w.

On définit, pour tout mot u, $u^0 = \varepsilon$ et, par récurrence, $u^{n+1} = u^n.u$ (d'où $u^1 = u$).

Exercice 3
Prouver que $w^n.w^p = w^p.w^n = w^{n+p}$.

Théorème 1 - Propriétés du produit

- Le produit de mots est associatif.
- $|u.v| = |u| + |v|$.
- Si $u.v = \varepsilon$ alors $u = v = \varepsilon$.
- Si $u.v = u.v'$ alors $v = v'$ (simplification à gauche).
- Si $u.v = u'.v$ alors $u = u'$ (simplification à droite).

L'associativité permet d'omettre les parenthèses et écrire $u.v.w$ pour $(u.v).w$ ou $u.(v.w)$.
Le produit est donc une loi interne dans $\mathcal{A}^*$, associative et admettant un élément neutre, ε, cela confère à $(\mathcal{A}^*, .)$ une structure de monoïde, comme $(\mathbb{N}, +)$ et $(A, \times)$ si $(A, +, \times)$ est un anneau.

Définition 7 - Produit de langages

Le produit des langages L_1 et L_2 est le langage $L_1.L_2$ défini par

$$L_1.L_2 = \{u_1.u_2 \ ; \ u_1 \in L_1, \ u_2 \in L_2\}$$

Exemples

- $\emptyset.L = L.\emptyset = \emptyset$.
- $\{\varepsilon\}.L = L.\{\varepsilon\} = L$.
- $\{u\}.\{v\} = \{u.v\}$.
- $\mathcal{A}^+ = \mathcal{A}.\mathcal{A}^*$.
- L'ensemble des mots de $\mathcal{A}$ qui commencent par a est $\{a\}.\mathcal{A}^*$.
- $\mathcal{A}^*.\{a\}.\mathcal{A}^*$ est le langage des mots contenant au moins une fois la lettre a
- $\{a\}.\{b, \varepsilon\}.\{a, b\} = \{a^2, ab, aba, ab^2\}$.
- Ne pas confondre $L.L$ et $\{w.w \ ; \ w \in L\}$.

Exercice 4 - Associativité
Prouver l'associativité du produit de mots.
En déduire que pour des langages L_1, L_2 et L_3 on a $(L_1.L_2).L_3 = L_1.(L_2.L_3)$.

Un mot $u = x1x_2 \cdots x_n$ est le produit des mots réduits à une lettre : $u = x_1.x_2.\cdots.x_n$. Ainsi un langage réduit à un mot est un produit de langages élémentaires : $\{u\} = \{x_1\}.\{x_2\}.\cdots.\{x_n\}$.

On en déduit qu'un langage fini est une union de produit de langages élémentaires.

Théorème 2 - Caractérisation des langages finis

L'ensemble des langages finis est le plus petit sous ensemble de $\mathcal{P}(\mathcal{A}^*)$ qui contient les langages élémentaires et qui est stable par produit et union de langages.

Cette caractérisation, un peu inutile et pédante, va nous servir de modèle pour généraliser les langages finis. On pourrait imaginer que la notion d'ensemble fini est suffisante pour une étude informatique mais des ensembles infinis sont parfois utiles pour rassembler des motifs qui, sinon, donneraient lieu à une énumération rébarbative de cas simples.
On va donc ajouter une (une seule) construction qui permet de définir des langages infinis "simples".

Si L est un langage on définit $L^0 = \{\varepsilon\}$ et, par récurrence, $L^{n+1} = L^n.L$ pour $n \geqslant 1$.
On notera, en particulier, que $\emptyset^0 = \{\varepsilon\}$ mais $\emptyset^n = \emptyset$ pour $n \geqslant 1$.

Théorème 3 - Propriétés des puissances de langage

- $L^1 = L$
- $L^n = \{w_1.w_2.\cdots.w_n ; w_i \in L \text{ pour tout } i\}$.
- $L^n.L^m = L^{n+m}$
- Si L n'est pas réduit à un mot alor $L^n \neq \{u^n \ ; \ u \in L\}$.

Définition 8 - Étoile de Kleene

L'étoile d'un langage L est $L^* = \bigcup_{n\in\mathbb{N}} L^n$.

On définit aussi $L^+ = \bigcup_{n\in\mathbb{N}^*} L^n$

Théorème 4 - Propriétés de l'étoile d'un langage

- L^* est l'ensemble des produits de mots de L.
- Si $L = \mathcal{A}$ on retrouve bien l'ensemble de tous les mots : la notation $\mathcal{A}^*$ est non–ambiguë.
- Si $L = \{w\}$ alors $L^* = \{w^n \ ; \ n \in \mathbb{N}\}$.
- $\emptyset^* = \{\varepsilon\}^* = \{\varepsilon\}$.
- Si L contient un mot non vide alors L^* est infini.

II.3 Langages rationnels

On commence par imiter la caractérisation des langages finis.

Définition 9 - Ensemble des langages rationnels

L'ensemble des langages rationnels est le plus petit sous ensemble de $\mathcal{P}(\mathcal{A}^*)$ qui contient les langages élémentaires et qui est stable par produit, union et étoile de langages.

Comme on a ajouté une construction, tout langage fini est rationnel.

Une conséquence utile de la définition est un mode de démonstration.

Théorème 5 - Induction structurelle

Si une propriété P portant sur des langages est telle que

- $P(\emptyset)$, $P(\{\varepsilon\})$ et $P(\{a\})$ pour $a \in \mathcal{A}$ sont vérifiées
- si $P(L_1)$ et $P(L_2)$ sont vraies
  alors $P(L_1 \cup L_2)$, $P(L_1.L_2)$ et $P(L_1^*)$ sont vraies

alors $P(L)$ est vraie pour tout langage rationnel.

Il suffit de considérer l'ensemble $\mathcal{L}$ des langages L tels que $P(L)$ est vérifiée.
En dehors de ce mode de démonstration, cette définition est peu utilisable pour déterminer si un langage donné est rationnel. Nous allons donner deux caractérisations qui mettent en évidence la construction pas-à-pas des langages rationnels.

La première méthode consiste à stratifier l'ensemble des langages rationnels.
Pour un alphabet donné $\mathcal{A}$, on définit par récurrence $\mathcal{R}_0 = \{\emptyset, \{\varepsilon\}\} \cup \bigcup_{a \in \mathcal{A}} \{a\}$ et

$$\mathcal{R}_{n+1} = \{L_1 \cup L_2 \ ; \ L_1, L_2 \in \mathcal{R}_n\} \cup \{L_1.L_2 \ ; \ L_1, L_2 \in \mathcal{R}_n\} \cup \{L^* \ ; \ L \in \mathcal{R}_n\}$$

Théorème 6 - Caractérisation des langages rationnels

L est rationnel si et seulement si il existe $n \in \mathbb{N}$ tel que $L \in \mathcal{R}_n$.

Démonstration : il suffit de montrer que l'ensemble des langages rationnels est $\mathcal{R} = \bigcup_{n \in \mathbb{N}} \mathcal{R}_n$. *Pour cela on montre par récurrence*

- $\{\varepsilon\} \in \mathcal{R}_n$ *pour tout* n *(*$\{\varepsilon\} = \{\varepsilon\}.\{\varepsilon\}$*).*
- $\mathcal{R}_n \subset \mathcal{R}_{n+1}$ *pour tout* n *(*$L = \{\varepsilon\}.L$*).*
- *Tous les éléments de* $\mathcal{R}_n$ *sont des langages rationnels.*

On en déduit que $\mathcal{R}$ *est inclus dans l'ensemble des langages rationnels.*
De plus $\mathcal{R}$ *contient les langages élémentaires (dans* $\mathcal{R}_1$*).*
Pour L_1 *et* L_2 *appartenant à* $\mathcal{R}$*, il existe* n_1 *et* n_2 *tels que* $L_1 \in \mathcal{R}_{n_1}$ *et* $L_2 \in \mathcal{R}_{n_2}$ *d'où* L_1 *et* L_2 *appartiennent à* $\mathcal{R}_n$ *avec* $n = max(n_1, n_2)$*.*
On a alors $L_1.L_2$*,* $L_1 \cup L_2$ *et* L_1^* *qui appartiennent à* $\mathcal{R}_{n+1} \subset \mathcal{R}$*.*
Ainsi $\mathcal{R}$ *contient tous les langages élémentaires et est stable par union, produit et étoile : il contient tous les langages élémentaires d'où la caractérisation.*

III Expressions régulières

Dans cette partie nous allons donner, pour chaque langage rationnel, une construction explicite à partir des langages élémentaires. L'alphabet $\mathcal{A}$ est fixé.

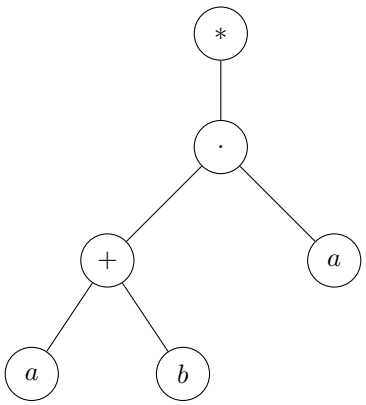
III.1 Arbre de construction

On considère les arbres dont les feuilles sont des langages élémentaires, on représentera le vide par $\boldsymbol{\varnothing}$ et ϵ par $\boldsymbol{\epsilon}$, et les nœuds internes sont

- soit un nœud binaire signifiant l'union, notée ici $+$,
- soit un nœud binaire signifiant le produit,
- soit un nœud unaire signifiant l'étoile.

Un arbre permet donc de construire un langage à partir de langages élémentaires à l'aide d'union, de produits et d'étoiles ; on obtient ainsi un langage rationnel.
Inversement une démonstration par récurrence sur n montre que tout élément de $\mathcal{R}_n$ est constructible à partir d'un arbre de hauteur n au plus donc tout langage rationnel est constructible par un arbre.

Par exemple le langage $\{ba, aa\}^*$ peut être construit par $\Big(\big(\{a\} \cup \{b\}\big).\{a\}\Big)^*$ donc on peut lui associer l'arbre



On considère qu'un tel arbre est un arbre binaire pour lequel le fils unique d'une étoile est considéré comme un fils gauche.
Le parcours infixe parenthésé de cet arbre est une chaîne de caractère :
l'arbre ci-dessus donne `(((a|b).a)*)`.
On va donner une expression formelle de cette chaîne de caractère.

III.2 Langages réguliers

Définition 10 - Expression régulière

$\mathcal{A}$ est un alphabet ne contenant pas les lettres "$\boldsymbol{\varnothing}$ " ,"$\boldsymbol{\epsilon}$", "*", "|", ".", "(" et ")". Une expression régulière sur $\mathcal{A}$ est une expression de la forme

- $\boldsymbol{\varnothing}$,
- $\boldsymbol{\epsilon}$,
- `a` avec $a \in \mathcal{A}$,
- `(r1|r2)` avec `r1` et `r2` expressions régulières,
- `(r1.r2)` avec `r1` et `r2` expressions régulières ou
- `(r*)` avec `r` expression régulière.

Une expression régulière fournit une construction d'un langage.

Définition 11 - Langage associé

À toute expression régulière e sur $\mathcal{A}$ on peut associer un langage par la construction récursive :

- $L[\texttt{ø}] = \emptyset$,
- $L[\texttt{ϵ}] = \{\varepsilon\}$,
- $L[\texttt{a}] = \{a\}$ pour $a \in \mathcal{A}$,
- $L[(\texttt{r1}|\texttt{r2})] = L[\texttt{r1}] \cup L[\texttt{r2}]$,
- $L[(\texttt{r}_1.\texttt{r}_2)] = L[\texttt{r1}].L[\texttt{r2}]$,
- $L[(\texttt{r}*)] = (L[\texttt{r}])^*$.

Un tel langage est appelé régulier.

Un langage régulier est construit à partir de langages élémentaires par des unions, produits et étoiles ; il est donc rationnel. Inversement le passage par l'arbre de construction montre qu'un langage rationnel peut être défini par une expression régulière.

Définition 12 - Dénotation

Les langages rationnels sont les langages réguliers.
Si $L = L[\texttt{r}]$ on dit que L est dénoté par l'expression régulière `r`.

Il faut noter que si le langage dénoté par une expression régulière est unique il peut exister plusieurs expressions régulières qui dénotent un langage donné.
Par exemple le langage donné en exemple est dénoté, comme on l'a vu, par `(((a|b).a)*)` mais aussi par `(((b.a)|(a.a))*)` ou `(((a.a)*).(((b.a).((a.a)*))*))`.
On voit que les notations deviennent vite lourdes : on pourra éliminer des parenthèses en utilisant la priorité de l'étoile sur le produit et la priorité du produit sur l'union. Les expressions régulières ci-dessus deviennent `((a|b).a)*`, `(b.a|a.a)*` et `(a.a)*.(b.a.(a.a)*)*`.

Définition 13 - Expressions équivalentes

Deux expressions régulières sont équivalentes si elles dénotent le même langage.

IV Langages locaux

IV.1 Définitions

Nous allons, dans cette partie, définir un type de langages caractérisé par une lecture locale des lettres (d'où le nom). Le principe est de ne lire les mots (et tester les lettres) qu'avec une fenêtre de lecture de largeur 2.

Définition 14 - Langage local

Un langage L sur l'alphabet $\mathcal{A}$ est local
s'il existe $P \subset \mathcal{A}$, $S \subset \mathcal{A}$ et $F \subset \mathcal{A}^2$
tels que $u = u_1 u_2 \cdots u_n \in L \setminus \{\varepsilon\}$ est équivalent à
$u_1 \in P$, $u_n \in S$ et $u_i u_{i+1} \in F$ pour tout $i \in \{1, 2, \ldots, n-1\}$.

L'appartenance de ε à L est facultative ; en plus des 3 ensembles il faut donc un quatrième déterminant pour indiquer si on a $\varepsilon \in L$.

Exemples

- $\emptyset$ et $\{\varepsilon\}$ sont locaux : ils sont définis par $P = F = S = \emptyset$.
- $L = \{a\}$ avec $a \in \mathcal{A}$ est local : $P = S = \{a\}$, $F = \emptyset$, $\varepsilon \notin L$.
- $L = \{(ab)^n \ ; \ n \in \mathbb{N}\}$ est local, défini par $P = \{a\}$, $F = \{ab, ba\}$, $S = \{b\}$ et $\varepsilon \in L$.
  Si on retire ε on obtient $\{(ab)^n \ ; \ n \in \mathbb{N}^*\}$.
- $L = \{a^n b \ ; \ n \in \mathbb{N}^*\}$ est local, défini par $P = \{a\}$, $F = \{ab, aa\}$, $S = \{b\}$ et $\varepsilon \notin L$.
- $L = \{a^n b^p \ ; \ n, p \in \mathbb{N}\}$ est local, défini par
  $P = \{a, b\}$, $F = \{aa, ab, bb\}$, $S = \{a, b\}$ et $\varepsilon \in L$.

Exercice 5 - Étoile d'un langage local
Prouver que l'étoile d'un langage local sur $\mathcal{A}$ est un langage local.

Exercice 6 - Intersection de langages locaux
Prouver que l'intersection de deux langages locaux sur $\mathcal{A}$ est un langage local.

On peut remplacer l'ensemble F par l'ensemble $N = \mathcal{A}^2 \setminus F$, la condition sur les facteurs devient $u_i u_{i+1} \notin N$. Cela permet de donner une définition plus synthétique des langages locaux.

Théorème 7 - Caractérisation des langages locaux

Un langage L sur l'alphabet $\mathcal{A}$ est local s'il existe $P \subset \mathcal{A}$, $S \subset \mathcal{A}$ et $N \subset \mathcal{A}^2$ tels que $L \setminus \{\varepsilon\} = \left(P.\mathcal{A}^* \cap \mathcal{A}^*.S\right) \setminus \mathcal{A}^*.N.\mathcal{A}^*$.

Exercice 7 - Démonstration du théorème 7
Démontrer le résultat ci-dessus.

La caractérisation permet de construire un langage local à partir des langages rationnels $P.\mathcal{A}^*$, $\mathcal{A}^*.S$ et $\mathcal{A}^*.N.\mathcal{A}^*$ mais on utilise des opérations non rationnelles : l'intersection et la différence. On ne peut pas conclure sur la rationalité de ces langages avant les résultats du chapitre suivant.

IV.2 Langages linéaires

Cependant il existe un cas particulier qui sera utilisé dans le chapitre suivant.

Définition 15 - Expressions linéaires

Une expression régulière est linéaire si toutes les lettres qui la composent sont distinctes.

Théorème 8 - Localité des langages linéaires

Si e est une expression régulière linéaire alors $L[\mathtt{e}]$ est un langage local.

La démonstration repose sur les deux exercices suivants dans lesquels un langage $\mathcal{A}$ peut s'écrire sous la forme avec $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ avec $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$

Exercice 8 - Union de langages locaux à alphabets distincts
Montrer que si L_1 est un langage local sur $\mathcal{A}_1$ et L_2 est un langage local sur $\mathcal{A}_2$ alors $L_1 \cup L_2$ est un langage local sur $\mathcal{A}$.

Exercice 9 - Produit de langages locaux à alphabets distincts
Montrer que si L_1 est un langage local sur $\mathcal{A}_1$ et L_2 est un langage local sur $\mathcal{A}_2$ alors $L_1.L_2$ est un langage local sur $\mathcal{A}$.

Exercice 10 - Démonstration du théorème 8
Montrer que si $\mathtt{r}$ est une expression régulière linéaire alors $L[\mathtt{r}]$ est un langage local.

A Exemples de langages

Exercice 11 - Alphabet à une lettre
On considère $\mathcal{A} = \{a\}$ et $L_k = \{a^{3n+k}, n \in \mathbb{N}\}$ pour $k \in \mathbb{N}$.
Déterminer $L_k \cap L_j$.
Déterminer $L_k.L_j$.

Exercice 12 - Exemples de langages rationnels

- Prouver que l'ensemble des mots de longueur n au plus est rationnel.
- Prouver que l'ensemble des mots de longueur n au moins est rationnel.
- Prouver que l'ensemble des mots qui commencent et qui finissent par la même lettre est rationnel.

Exercice 13 - Contre-exemple
On pose $L_1 = \{ab\}$ et $L_2 = \{a^n \ ; \ n \in \mathbb{N}\}$.
Prouver que L_1 et L_2 sont locaux mais que ni $L_1 \cup L_2$ ni $L_1.L_2$ ne sont locaux.
On a donc des exemples de langages rationnels non locaux.

Exercice 14 - Racine d'un langage
Pour un langage L, sa racine est $\sqrt{L} = \{u \in \Sigma \ / \ u.u \in L\}$.
Déterminer $\sqrt{L_k}$ où $L_k = \{a^{3n+k}, n \in \mathbb{N}\}$.
Montrer que $\epsilon \in L$ si et seulement si $\epsilon \in \sqrt{L}$.

Exercice 15 - Transformations
L est un langage rationnel sur un alphabet $\mathcal{A}$.
On suppose que a est une lettre de $\mathcal{A}$.
Prouver que chacun des langages suivants est rationnel.

- Le langage des mots préfixes des mots de L.
- L'ensemble des mots de L qui ne contiennent pas a.
- L'ensemble des mots de L qui contiennent a.
- L'ensemble des mots obtenus en enlevant le premier a dans les mots de L contenant a
- L'ensemble des mots obtenus en enlevant un a dans les mots de L contenant a

Exercice 16 - Code
Un code sur un alphabet $\mathcal{A}$ est un langage L sur $\mathcal{A}$ tel que l'égalité $x_1.x_2.\cdots.x_p = y_1.y_2.\cdots.y_q$ avec $x_1,\ldots, x_p, y_1,\ldots, y_q$ dans L entraîne $p = q$ et $x_i = y_i$ pour tout i.

1. Déterminer les codes parmi les langages finis suivants :
   - $L_1 = \{ab, baa, abba, aabaa\}$,
   - $L_2 = \{b, ab, baa, abaa, aaaa\}$,
   - $L_3 = \{aa, ab, aab, bba\}$,
   - $L_4 = \{a, ba, bba, baab\}$.
2. Soit $u \in \mathcal{A}^*$, montrer que $\{u\}$ est un code si et seulement si $u \neq \varepsilon$.
3. Soit u et v deux mots distincts ; montrer que $\{u, v\}$ est un code si et seulement si $u.v \neq v.u$.
4. Soit L un langage ne contenant pas ϵ et tel qu'aucun mot de L ne soit préfixe d'un autre mot de L. Montrer que L est un code.

Exercice 17 - Lemme d'Arden
A et B sont deux langages sur un même alphabet. On suppose que L vérifie $L = A.L \cup B$.

1. Montrer que $A^*.B \subset L$.
2. Montrer que si $\epsilon \notin A$ alors $L = A^*.B$.
3. Montrer que si $\epsilon \in A$ alors les solutions sont les ensembles de la forme $A^*.C$ avec C contenant B.

Exercice 18 - Application du lemme d'Arden
Sur $\mathcal{A} = \{a, b\}$, on note L_1 le langage des mots ayant un nombre pair de b et L_2 le langage des mots ayant un nombre impair de b. Écrire deux relations linéaires liant L_1 et L_2 puis utiliser le lemme d'Arden pour en donner une expression.

B Langages dérivés

Définition 16 - Langage dérivé

Si L est un langage sur $\mathcal{A}$ et si u est un mot ($u \in \mathcal{A}^*$) le langage dérivé (à gauche) de L par u est $u^{-1}.L = \{v \in \mathcal{A}^* \ / \ u.v \in L\}$.
On dit aussi que $u^{-1}.L$ est un **résiduel** de L.

Exercice 19 - Première propriétés
Prouver les propriétés

1. ε appartient à $u^{-1}.L$ si et seulement si $u \in L$.
2. $\varepsilon^{-1}.L = L$.
3. $u^{-1}.\mathcal{A}^* = A^*$.
4. $v^{-1}.(u^{-1}.L) = (u.v)^{-1}.L$.

Exercice 20 - Un exemple
On note L_1 le langage sur $\mathcal{A} = \{a, b\}$ des mots ayant un nombre pair de b et L_2 le langage des mots ayant un nombre impair de b.
Calculer les dérivés $u^{-1}.L_1$ et $u^{-1}.L_2$ pour un mot de $\mathcal{A}^*$.

Exercice 21 - Un autre exemple
On pose $L = \{a^n b^n \ ; \ n \in \mathbb{N}\}$ sur $\mathcal{A} = \{a, b\}$.
Prouver que $(a^p)^{-1}.L = \{a^n b^{n+p} \ ; \ n \in \mathbb{N}\}$ pour $p \geqslant 1$.
Prouver que $(a^p.b^q)^{-1}.L = \{b^{p-q}\}$ pour $p \geqslant q \geqslant 1$.

Exercice 22 - Opérations rationnelles
Si L est un langage et u un mot on note $L^{-1}.u = \{w \in \mathcal{A}^* \ ; \ \exists v \in L, \ u = v.w\}$.

1. Prouver que $u^{-1}(L_1 \cup L_2) = u^{-1}L_1 \cup u^{-1}L_2$.
2. Prouver que $u^{-1}.(L_1.L_2) = (u^{-1}.L_1).L_2 \cup \bigcup_{w \in L_1^{-1}.u} w^{-1}.L_2$.
3. Prouver que $u^{-1}.(L_1^*) = \bigcup_{w \in (L_1^*)^{-1}.u} (w^{-1}.L_1).L_1^*$.

Exercice 23 - Rationalité
Prouver que $u^{-1}.L$ est rationnel si L est rationnel.

Exercice 24 - Critère de rationalité
Prouver qu'un langage rationnel admet un nombre fini de dérivés.

Exercice 25 - Contre-exemple
Prouver que $L = \{a^n b^n \ ; \ n \in \mathbb{N}\}$ n'est pas rationnel.

C Mots de Lyndon

On suppose que l'alphabet $\mathcal{A}$ est $\{0,1\}$.

Définition 17 - Comparaisons

- On pose $u \ll v$ si et seulement s'il existe un entier $k \leqslant \min\{|u|,|v|\}$ tel que, en notant $u = x_1x_2\cdots x_n$ et $v = y_1y_2\cdots y_m$, $x_i = y_i$ pour $1 \leqslant i < k$ et $x_k < y_k$.
- Un mot v est un **préfixe** de u s'il existe w tel que $u = v.w$.
- On pose $u \preceq v$ si et seulement si u est un préfixe de v ou $u \ll v$.
- On pose $u \prec v$ si et seulement si $u \preceq v$ mais $u \neq v$.

Exercice 26
Montrer que $\preceq$ est une relation d'ordre total.

Exercice 27
Écrire une fonction `inferieur : int list -> int list-> bool` qui prend pour deux mots u et v et renvoie pour résultat le booléen `true` si $u \prec v$ et `false` sinon.

Définition 18 - Conjugués

Un conjugué d'un mot u est un mot de la forme $v = u''.u'$ avec $u = u'.u''$, $u' \neq \varepsilon$ et $u'' \neq \varepsilon$.
Un mot non vide est un **mot de Lyndon** si $u \prec v$ pour tout conjugué v de u.

Exercice 28
Écrire la fonction `conjugue : int list -> int -> int list` telle que conjugue u i renvoie le conjugué $w.v$ de $u = v.w$ avec $|v| = i$.

Exercice 29
Écrire une fonction `lyndon : int list -> bool` qui teste si un mot est un mot de Lyndon.

Exercice 30
Soit u un mot. Démontrer que s'il existe un mot qui est à la fois un préfixe et un suffixe de u, alors u n'est pas un mot de Lyndon.

Exercice 31
Démontrer qu'un mot u est un mot de Lyndon si et seulement si pour tout suffixe propre h de u, on a $u \prec h$.

Exercice 32
Démontrer que si un mot u de longueur au moins 2 est un mot de Lyndon
alors il existe deux mots de Lyndon f et g tels que $u = f.g$ et $f \prec g$
et, réciproquement, si f et g sont des mots de Lyndon tels que $f \prec g$ alors $f.g$ est un mot de Lyndon.

Exercice 33
Donner la liste de tous les mots de Lyndon de longueur 5.

Exercice 34
Écrire la fonction `Lyndon n` qui calcule, dans un tableau de listes, tous le mots de Lyndon de longueur inférieure ou égale à n. Les mots de Lyndon de longueur k devront être ordonnés dans la composante d'indice k du tableau.

Définition 19 - Factorisation de Lyndon

Une factorisation de Lyndon d'un mot u est une suite $u^{(1)}, u^{(2)}, \ldots, u^{(p)}$ de mots de Lyndon telle que $u = u^{(1)}.u^{(2)}.\cdots.u^{(p)}$ et $u^{(p)} \preceq u^{(p-1)} \preceq \cdots \preceq u^{(1)}$.

Par exemple, pour $u = 10100101100100$, une factorisation de Lyndon est $1, 01, 001011, 011, 0, 0$.

Exercice 35 -
Prouver qu'un mot admet au plus une décomposition de Lyndon.

Pour construire une factorisation de Lyndon d'un mot $u = u_1u_2\cdots u_n$ on construit une suite de décompositions de u en mots de Lyndon (mais pas encore une décomposition de Lyndon) à partir de la suite $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$ avec $u^{(i)} = u_i$.
À chaque étape $v^{(1)}, v^{(2)}, \ldots, v^{(p)}$ de mots de Lyndon telle que $u = v^{(1)}.v^{(2)}.\cdots.v^{(p)}$ on cherche s'il existe un indice k tel que $v^{(k)} \prec v^{(k+1)}$: si oui, on remplace la suite par
$v^{(1)}, \ldots, v^{(k-1)}, v^{(k)}.v^{(k+1)}, v^{(k+2)}, \ldots, v^{(p)}$.
S'il n'y en a pas, on a fini : la séquence obtenue est une factorisation de Lyndon.

Exercice 36
Écrire la fonction `factorisation` qui calcule une factorisation de Lyndon du mot u passé en paramètre et renvoie le résultat sous forme de la liste des facteurs.

D Manipulations d'expressions régulières

Exercice 37 - 3 lettres
Donner une expression régulière dénotant l'ensemble L des mots sur l'alphabet $\mathcal{A} = \{a, b, c\}$ tels que deux lettres consécutives soient toujours distinctes.

Exercice 38 - Complémentaire
Donner une représentation régulière dénotant $\overline{L}$ avec L dénoté par `(a|b)*.a.b.a.(a|b)*`.

Exercice 39 - Expressions équivalentes

- Montrer que `(a|b)*`, `(a*.b)*.a*`, `a*|a*.b.(a|b)*` et `b*.(a.a*.b.b*)*.a*` sont équivalentes.
- Prouver que `(r1.r2)*` et `ϵ|r1.(r2.r1)*.r2` sont équivalentes.

Exercice 40 - Détermination d'expressions régulières

- L'ensemble des mots qui contiennent au moins un a,
- L'ensemble des mots qui contiennent au plus un a,
- L'ensemble des mots tels que toute série de a soit de longueur paire,
- L'ensemble des mots dont la longueur n'est pas divisible par 3,
- L'ensemble des mots tels que deux lettres consécutives soient toujours distinctes,
- L'ensemble des mots contenant au moins un a et un b.

Parmi les langages élémentaires nous avons ajouté $\emptyset$ et $\{\varepsilon\}$. Nous allons voir qu'en fait ils servent uniquement à fabriquer l'ensemble vide ou, dans certains cas, à ajouter le mot vide.

Exercice 41 - Élimination de zéro
Montrer que toute expression régulière est équivalente à `∅` ou à une expression régulière ne contenant pas `∅` .

Définition 20 - Expression réduite

Une expression régulière est appelée réduite si elle est de la forme `∅` , `ϵ`, `r` ou `r|ϵ` avec `r` ne contient ni `∅` ni `ϵ`.

Exercice 42 - Réduction
Prouver que tout langage rationnel est dénoté par une expression régulière réduite.

E Fonctions Caml

Le type des expressions régulières est semblable à celui des arbres

```
type expreg = |Zero
              |Un
              |Lettre of string
              |Union of expreg * expreg
              |Produit of expreg * expreg
              |Etoile of expreg;;
```

Exercice 43
Écrire une fonction `ecrire : expreg -> string` qui renvoie l'expression régulière associée à un arbre en utilisant le parcours infixe parenthésé.

Exercice 44 - Langage vide
Écrire une fonction `langageVide : expreg -> bool` qui répond à la question :
"Le langage L[r] est-il vide ?"

Exercice 45 - Mot vide
Écrire une fonction `motVide : expreg -> bool` qui répond à la question :
"Le langage L[r] contient-il le mot vide ?"

Exercice 46 - Transformations, reprise de l'exercice 15
Écrire des fonctions de type `string -> expreg -> expreg` qui transforment un expression régulière dénotant L en une un expression régulière dénotant respectivement

- L'ensemble des mots de L qui ne contiennent pas a.
- L'ensemble des mots de L qui contiennent a.
- L'ensemble des mots obtenus en enlevant le premier a dans les mots de L contenant a
- L'ensemble des mots obtenus en enlevant un a dans les mots de L contenant a

Exercice 47 - Prefixes, reprise de l'exercice 15
Écrire une fonction `prefixes : expreg -> expreg` qui transforment un expression régulière dénotant L en une un expression régulière dénotant le langage des mots préfixes des mots de L.

Exercice 48 - Vers les langages locaux
Écrire des fonctions `prefixes1 : expreg -> string list`, `suffixes1 : expreg -> string list` et `facteurs2 : expreg -> string list` qui calculent les ensembles des préfixes de taille 1, des suffixes de taille 1 et des facteurs de taille 2 d'un langage dénoté par une expression régulière.

Exercice 49 - Réduction, reprise de l'exercice 42
Écrire une fonction `reduire : expreg -> expreg` qui transforme un expression régulière dénotant L en une un expression régulière réduite dénotant L.

Solutions

Solution de l'exercice 1 - Complexité
On note $T(n,p)$ le nombre de comparaisons effectuées dans la recherche d'un mot de longueur p dans un texte de longueur n.
On effectue $n-p+1$ comparaisons de mots de longueur p.
Ces comparaisons demandent au plus p comparaisons de caractères
donc $T(n,l) \leqslant (n-p+1).p = \mathcal{O}(np)$.
Le pire des cas advient avec `texte = "aaaaaaaaaaaaaaaaaaa"` et `motif = "aaaaab"`.
Dans ce cas on doit attendre la dernière comparaison de caractère pour voir que le motif n'apparaît pas pour chaque position : le nombre de comparaison est égal à $(n-p+1).p$.

Solution de l'exercice 2 - Complexité moyenne
On note t_i $(0 \leqslant i < n)$ les lettres de `texte` et m_j $(0 \leqslant j < p)$ celles de `motif`.
La comparaison des deux chaînes `sub_string texte i p` et `motif` fait k comparaisons si les $k-1$ premières lettres sont égales et les k-ièmes lettres sont distinctes sauf pour $k=p$; il y a p comparaisons quand les $k-1$ premières lettres sont égales car la dernière comparaison donne le résultat.
Si c_k est la probabilité de faire exactement k comparaisons de caractères on a donc
$c_k = \left(\frac{1}{q}\right)^{k-1} \frac{q-1}{q}$ pour $1 \leqslant k < p$ et $c_p = \left(\frac{1}{q}\right)^{p-1}$.
L'espérance du nombre de comparaisons de caractères (pour une fenêtre) est donc

$$\overline{c} = \sum_{k=1}^{p} k c_k = \sum_{k=1}^{p-1} k \left(\frac{1}{q}\right)^{k-1} \frac{q-1}{q} + p\left(\frac{1}{q}\right)^{p-1} = \frac{q-1}{q}\left(1 - \frac{1}{q^p}\right) \leqslant \frac{q-1}{q}$$

Solution de l'exercice 3
On prouve, par récurrence sur p, que $w^n.w^p = w^{n+p}$ en utilisant l'associativité.
On a alors $w^p.w^n = w^{p+n} = w^{n+p} = w^n.w^p$.

Solution de l'exercice 4 - Associativité
On note $u = u_1u_2\cdots u_n$, $v = v_1v_2\cdots v_p$ et $w = w_1w_2\cdots w_q$.
Les deux produits $(u.v).w$ et $u.(v.w)$ s'écrivent $t_1t_2\cdots t_{n+p+q}$ avec $t_i = u_i$ pour $1 \leqslant i \leqslant n$, $t_i = v_{i-n}$ pour $n+1 \leqslant i \leqslant n+p$ et $t_i = w_{i-n-p}$ pour $n+p+1 \leqslant i \leqslant n+p+q$.

Les deux produits de langages sont égaux à $\{w1.w2.w3\ ;\ w_1 \in L_1,\ w_2 \in L_3,\ w_3 \in L_3\}$.

Solution de l'exercice 5 - Étoile d'un langage local
Si $L \setminus \{\varepsilon\}$ est défini par (P,S,F) alors L^* contient ϵ et est défini par $(P,S,F \cup S.F)$.

Solution de l'exercice 6 - Intersection de langages locaux
Si $L_1 \setminus \{\varepsilon\}$ et $L_2 \setminus \{\varepsilon\}$ sont associés aux triplets (P_1,S_1,F_1) et (P_2,S_2,F_2) alors $(L_1 \cap L_2) \setminus \{\varepsilon\}$ est local avec les paramètres $(P_1 \cap P_2, S_1 \cap S_2, F_1 \cap F_2)$.

Solution de l'exercice 7 - Démonstration du théorème 7
On note $L_1 = \left(P.\mathcal{A}^* \cup \mathcal{A}^*.S\right) \setminus \mathcal{A}^*.N.\mathcal{A}^*$ et L_2 le langage local défini par P, S, $F = \mathcal{A}^2 \setminus N$ et $\varepsilon \notin L$. On va prouver que $L_1 = L_2$.

- Si $u = u_1u_2\cdots u_n \in L_1$ alors $u = u_1v \in P.\mathcal{A}^*$ donc $u_1 \in P$ et $u = wu_n \in \mathcal{A}^*.S$ donc $u_n \in S$.
  Si on n'avait pas $u_iu_{i+1} \in F$ alors on aurait $u_iu_{i+1} \in N$ donc $u = u'u_iu_{i+1}u" \in \mathcal{A}^*.N.\mathcal{A}^*$ ce qui est impossible. Ainsi $u_iu_{i+1} \in F$ pour tout i d'où $u \in L_2$. On a prouvé $L_1 \subset L_2$.
- Si $u = u_1u_2\cdots u_n \in L_2$ alors $u_1 \in P$ donc $u \in P.\mathcal{A}^*$ et $u_n \in S$ donc $u \in \mathcal{A}^*.S$.
  Si on avait $u \in \mathcal{A}^*.N.\mathcal{A}^*$ alors il existerait i tel que $u_iu_{i+1} \in N = \mathcal{A}^2 \setminus F$ ce qui est impossible.
  Ainsi $u \in \left(P.\mathcal{A}^* \cup \mathcal{A}^*.S\right) \setminus \mathcal{A}^*.N.\mathcal{A}^* = L_2$. On a prouvé $L_2 \subset L_1$.

Solution de l'exercice 8 - Union de langages locaux à alphabets distincts
L_1 est défini par (S_1, P_1, F_1) et L_2 est défini par (S_2, P_2, F_2).
$L_1 \cup L_2 \setminus$ est inclus dans le langage défini par $(S_1 \cup S_2, P_1 \cup P_2, F_1 \cup F_2)$.
Soit $u = x_1 x_2 \cdots x_n$ appartenant au langage local défini par $(S_1 \cup S_2, P_1 \cup P_2, F_1 \cup F_2)$.
Si $x_1 \in \mathcal{A}_1$ alors $x_1 x_2 \in F_1 \cup F_2$ impose $x_1 x_2 \in F_1$ donc $x_2 \in \mathcal{A}_1$ et toutes les autres lettres appartiennent à $\mathcal{A}_1$ donc $u \in L_1$.
De même si $x_1 \in \mathcal{A}_2$ alors $u \in L_2$. Ainsi $u \in L_1 \cup L_2$ d'où l'égalité demandée.

Solution de l'exercice 9 - Produit de langages locaux à alphabets distincts

- Si $\epsilon \notin L_1$ et $\epsilon \notin L_2$ alors $L_1.L_2$ est défini par $(P_1, S_2, F_1 \cup F_2 \cup S_1.P_2)$.
- Si $\epsilon \in L_1$ et $\epsilon \notin L_2$ alors $L_1.L_2$ est défini par $(P_1 \cup P_2, S_2, F_1 \cup F_2 \cup S_1.P_2)$.
- Si $\epsilon \notin L_1$ et $\epsilon \in L_2$ alors $L_1.L_2$ est défini par $(P_1, S_1 \cup S_2, F_1 \cup F_2 \cup S_1.P_2)$.
- Si $\epsilon \in L_1$ et $\epsilon \in L_2$ alors $L_1.L_2$ est défini par $(P_1 \cup P_2, S_1 \cup S_2, F_1 \cup F_2 \cup S_1.P_2)$.

Solution de l'exercice 10 - Démonstration du théorème 8
On procède par induction structurelle sur l'expression régulière.

- Si `r` est $\varnothing$, ϵ ou `a` avec $a \in \mathcal{A}$ alors $L[\texttt{r}]$ est local.
- Si `r = (r1*)` et si `r` est linéaire alors `r1` est linéaire.
  Si $L[\texttt{r1}]$ est local on a vu qu'alors $L[\texttt{r}] = \big(L[\texttt{r1}]\big)^*$ est local.
- Si `r = (r1|r2)` et si `r` est linéaire alors `r1` et `r2` sont linéaires.
  Si $L[\texttt{r1}]$ et $L[\texttt{r2}]$ sont locaux alors, comme les lettres de `r1` et de `r2` appartiennent à des ensembles distincts, l'exercice ci-dessus montre que $L[\texttt{r}] = L[\texttt{r1}] \cup L[\texttt{r2}]$ est local.
- De même si `r = (r1.r2)`

Solution de l'exercice 11 - Alphabet à une lettre
$L_k \cap L_j = \emptyset$ si $k - j$ n'est pas un multiple de 3, $L_k \cap L_j = L_p$ avec $p = \min(k, j)$ sinon.
$L_k.L_j = L_{k+j}$

Solution de l'exercice 12 - Exemples de langages rationnels

- L'ensemble des mots de longueur n au plus est fini donc rationnel.
- L'ensemble des mots de longueur n exactement, $\mathcal{A}^n$, est fini donc rationnel. L'ensemble des mots de longueur n au moins est le produit des langages rationnels $\mathcal{A}^n$ et $\mathcal{A}^*$ donc est rationnel
- L'ensemble des mots qui commencent et qui finissent par la même lettre est l'union des $\{x\}.\mathcal{A}^*.\{x\}$ pour x décrivant Σ. Il est rationnel.

Solution de l'exercice 13 - Contre-exemple
L_1 est défini par $(\{a\}, \{b\}, \{ab\})$ et L_2 par $(\{a\}, \{a\}, \{aa\})$. Aucun ne contient ε.

- Si $L_1.L_2 = \{aba^n \ ; \ n \in \mathbb{N}\}$ était local on aurait $P = \{a\}$, $S = \{b, a\}$ et $F = \{ab, ba, aa\}$ donc $L_1.L_2$ devrait contenir $abab$ ce qui n'est pas le cas : il n'est pas local.
- Si $L_1 \cup L_2$ était local alors on aurait $P = \{a\}$, $S = \{b, a\}$ et $F = \{ab, ba, aa\}$. Ainsi aab appartiendrait à $L_1 \cup L_2$ ce qui n'est pas le cas : $L_1 \cup L_2$ n'est pas local.

Solution de l'exercice 14 - Racine d'un langage
$\sqrt{L_{2p}} = L_p$, $\sqrt{L_{2p+1}} = L_{p+2}$.

Si $\epsilon \in L$ alors $\epsilon.\epsilon = \epsilon \in L$ donc $\epsilon \in \sqrt{L}$.
Si $\epsilon \in \sqrt{L}$ alors $\epsilon = \epsilon.\epsilon \in L$.

Solution de l'exercice 15 - Transformations
On va définir par induction structurelle des transformations de langages f_1, f_2, f_3, f_4 et f_5 pour chaque question.

- $f_1(\emptyset) = \emptyset$,
  $f_1(\{\epsilon\}) = \{\epsilon\}$,
  $f_1(\{x\}) = \{\epsilon, x\}$,
  $f_1(L_1 \cup L_2) = f_1(L_1) \cup f_1(L_2)$,
  $f_1(L_1.L_2) = f_1(L_1) \cup L_1.f_1(L_2)$,
  $f_1(L^*) = L^*.f_1(L)$.
- $f_2(\emptyset) = \emptyset$,
  $f_2(\{\epsilon\}) = \{\epsilon\}$,
  $f_2(\{x\}) = \{x\}$ si $x \neq a$ et $f_2(\{a\}) = \emptyset$,
  $f_2(L_1 \cup L_2) = f_2(L_1) \cup f_2(L_2)$,
  $f_2(L_1.L_2) = f_2(L_1).f_2(L_2)$,
  $f_2(L^*) = \big(f_2(L)\big)^*$.
- $f_3(\emptyset) = \emptyset$,
  $f_3(\{\epsilon\}) = \emptyset$,
  $f_3(\{x\}) = \emptyset$ si $x \neq a$ et $f_3(\{a\}) = \{a\}$,
  $f_3(L_1 \cup L_2) = f_2(L_1) \cup f_3(L_2)$,
  $f_3(L_1.L_2) = f_3(L_1).L_2 \cup L_1.f_3(L_2)$,
  $f_3(L^*) = L^*.f_3(L).L^*$.
- $f_4(\emptyset) = \emptyset$,
  $f_4(\{\epsilon\}) = \emptyset$,
  $f_4(\{x\}) = \emptyset$ si $x \neq a$ et $f_4(\{a\}) = \{\epsilon\}$,
  $f_4(L_1 \cup L_2) = f_4(L_1) \cup f_4(L_2)$,
  $f_4(L_1.L_2) = f_4(L_1).L_2 \cup f_2(L_1).f_4(L_2)$,
  $f_4(L^*) = \big(f_2(L)\big)^*.f_4(L).L^*$.
- $f_5(\emptyset) = \emptyset$,
  $f_5(\{\epsilon\}) = \emptyset$,
  $f_5(\{x\}) = \emptyset$ si $x \neq a$ et $f_5(\{a\}) = \{\epsilon\}$,
  $f_5(L_1 \cup L_2) = f_5(L_1) \cup f_5(L_2)$,
  $f_5(L_1.L_2) = f_5(L_1).L_2 \cup L_1.f_5(L_2)$,
  $f_5(L^*) = L^*.f_5(L).L^*$.

Solution de l'exercice 16 - Code

1. - $L_1 = \{ab, baa, abba, aabaa\}$: $ab.(baa)^3 = abba.ab.aabaa$, L_1 n'est pas un code
   - $L_2 = \{b, ab, baa, abaa, aaaa\}$:
     on suppose que $x_1.x_2.\cdots.x_p = y_1.y_2.\cdots.y_q$ avec $x_1 \neq y_1$ (sinon on simplifie).
     Si, par exemple, $|x_1| \leqslant |y_1|$ on voit que les seules possibilités sont $x_1 = b$ et $y_1 = baa$ ou $x_1 = ab$ et $y_1 = abaa$ ce qui impose que x_2 commence par aa donc $x_2 = aaaa$.
     y_2 doit alors commencer par aa donc $y_2 = aaaa$ et on voit que tous les termes restant sont $aaaa$ sans jamais pouvoir égaler les longueurs. L_2 est un code
   - $L_3 = \{aa, ab, aab, bba\}$:
     on suppose que $x_1.x_2.\cdots.x_p = y_1.y_2.\cdots.y_q$ avec $x_1 \neq y_1$ (sinon on simplifie).
     Si, par exemple, $|x_1| \leqslant |y_1|$ on voit que $x_1 = aa$ et $y_1 = aab$ donc $x_2 = bba$ et y_2 devrait commencer par ba ce qui est impossible. L_3 est un code
   - $L_4 = \{a, ba, bba, baab\}$: $ba.a.bba = baab.ba$, L_4 n'est pas un code
2. $u^n = u^p$ impose $n|u| = p|u|$ donc $n = p$ si $|u| \neq 0$. Ainsi $\{u\}$ est un code si $u \neq \varepsilon$.
   Par contre $\varepsilon = \varepsilon.\varepsilon$ donc $\{\varepsilon\}$ n'est pas un code.

3. Si $u.v = v.u$ alors on a 2 décompositions d'un même mot : $\{u, v\}$ n'est pas un code.
   On suppose que $\{u, v\}$ n'est pas un code. On peut écrire $x_1.x_2.\cdots.x_p = y_1.y_2.\cdots.y_q$ avec $x_1 \neq y_1$. Par exemple $x_1 = u$ et $y_1 = v$. Ainsi u et v sont préfixes d'un même mot donc, par exemple, u est préfixe de v : on pose $v = u.v'$ on a $v' \neq \epsilon$ car $u \neq v$.
   Si $v' = u$ alors $v = u^2$ donc $u.v = v.u$.
   Si $u \neq v'$ on est alors ramené, en remplaçant v par $u.v'$ à un mot dans $\{u, v'\}$ qui admet deux écritures distinctes avec u et v' distincts dans $\{u, v'\}^+$.
   On peut alors conclure par récurrence sur $|u| + |v|$.
   En effet On a $|u| + |v'| < |u| + |v|$ donc $u.v' = v'.u$ d'où $u.v = u.u.v' = u.v'.u = v.u$.
4. Si L n'est pas un code alors on peut écrire $x_1.x_2.\cdots.x_p = y_1.y_2.\cdots.y_q$ avec $x_1 \neq y_1$ (sinon on simplifie). Si, par exemple, $|x_1| \leqslant |y_1|$ on voit que x_1 est un préfixe de y_1. Ainsi, si aucun mot de L n'est préfixe d'un autre mot de L, alors L est un code.

Solution de l'exercice 17 - Lemme d'Arden

1. On a $B \subset A.L \cup B = L$. Alors $A.B \subset A.L \subset A.L \cup B = L$.
   Par récurrence, si $A^n.B \subset L$ alors $A^{n+1}.B = A.(A^n.B) \subset A.L \subset A.L \cup B = L$.
   Ainsi $A^n.B \subset L$ pour tout n donc $A^*.B = \bigcup_{n\in\mathbb{N}} A^n.B \subset L$.
2. On suppose que $\epsilon \notin A$ et que $L \neq A^*.B$. On a donc, comme $A^*.B \subset L$, $L \setminus A^*.B \neq \emptyset$.
   Soit u un mot de longueur minimale appartenant à $L \setminus A^*.B$.
   $u \in L = A.L \cup B$ donc
   - soit $u \in B \subset A^*.B$ ce qui est impossible
   - soit $u \in A.L$ donc $u = u_1u_2$ avec $u_1 \in A$ et $u_2 \in L$. On a $u_1 \neq \epsilon$ car $\epsilon \notin A$ donc $|u_1| > 0$ puis $|u_2| < |u|$. On a ainsi $u \in L$ de longueur strictement inférieure à celle de $|u|$ donc $u_2 \in A^*.B$ puis $u = u_1.u_2 \in A.(A^*.B) \subset A^*.B$ ce qui est impossible aussi.

   On doit donc avoir $L \subset A^*.B$ d'où l'égalité.
3. On a $A^0.L = L \subset L$. Si $A^n.L \subset L$ alors $A^{n+1}.L = A.(A^n.L) \subset AL \subset L$ donc, par récurrence sur n on a $A^n.L \subset L$ pour tout n puis $A^*.L \subset L$.
   L'inclusion inverse est évidente donc $L = A^*.L$ avec $B \subset A.L \cup B = L$.

   On suppose maintenant que $\epsilon \in A$; soit $C \subset \Sigma^*$ tel que $B \subset C$.
   On veut montrer que $L = A^*.C$ vérifie $A.L \cup B = L$.
   On a $A.(A^*.C) = A^+.C \subset A^*.C$ et, comme $\epsilon \in A$, $A^*.C = \{\epsilon\}.(A^*.C) \subset A.(A^*.C)$ donc $A^*.C = A.(A^*.C)$. De plus $B \subset C \subset A^*.C$ donc
   $A.L \cup B = A.(A^*.C) \cup B = A^*.C \cup B = A^*.C = L$: $A^*.C$ est bien solution.

Solution de l'exercice 18 - Application du lemme d'Arden
Si un mot contient un nombre pair de b alors il peut être ϵ ou est de la forme $a.u$ avec u contenant un nombre pair de b ou est de la forme $b.u$ avec u contenant un nombre impair de b. Ainsi $L_1 = \{\epsilon\} \cup \{a\}.L_1 \cup \{b\}.L_2$.
De même $L_2 = \{a\}.L_2 \cup \{b\}.L_1$.
Comme $\epsilon \notin \{b\}$ la dernière équation impose $L_2 = \{a\}^*.\{b\}.L_1$ d'où
$L_1 = \{\epsilon\} \cup \{a\}.L_1 \cup \{b\}.\{a\}^*.\{b\}.L_1 = (\{a\} \cup \{b\}.\{a\}^*.\{b\}).L_1 \cup \{\epsilon\}$.
ϵ n'appartient pas à $\{a\} \cup \{b\}.\{a\}^*.\{b\}$ donc on doit avoir $L_1 = (\{a\} \cup \{b\}.\{a\}^*.\{b\})^*.\{\epsilon\} = (\{a\} \cup \{b\}.\{a\}^*.\{b\})^*$ puis $L_2 = \{a\}^*.\{b\}.(\{a\} \cup \{b\}.\{a\}^*.\{b\})^*$

Solution de l'exercice 19 - Première propriétés

1. v appartient à $u^{-1}.L$ si et seulement si $u.v$ appartient à L donc $\varepsilon \in u^{-1}.L$ si et seulement si $u = u.\varepsilon \in L$.
2. $u \in \varepsilon^{-1}.L$ si et seulement si $u = \varepsilon.u \in L$: $\varepsilon^{-1}.L = L$.
3. $u^{-1}.\mathcal{A}^* \subset A^*$; si $v \in \mathcal{A}^*$ alors $u.v \in \mathcal{A}^*$ donc $v \in u^{-1}.\mathcal{A}^*$ d'où $\mathcal{A}^* \subset u^{-1}.\mathcal{A}^*$ puis $\mathcal{A}^* = u^{-1}.\mathcal{A}^*$.
4. $w \in v^{-1}.(u^{-1}.L) \iff v.w \in u^{-1}.L \iff u.(v.w) \in L \iff (u.v).w \in L \iff w \in (u.v)^{-1}.L$.

Solution de l'exercice 20 - Un exemple
En comptant le nombre de b on montre que $a^{-1}.L_1 = L_1$, $b^{-1}.L_1 = L_2$, $a^{-1}.L_2 = L_2$ et $b^{-1}.L_2 = L_1$. En appliquant le résultat ci-dessus on en déduit que
$u^{-1}.L_1 = L_1$ et $u^{-1}.L_2 = L_2$ si $u \in L_1$; $u^{-1}.L_1 = L_2$ et $u^{-1}.L_2 = L_1$ si $u \in L_2$.

Solution de l'exercice 21 - Un autre exemple
$u \in (a^p)^{-1}.L \iff a^p.u \in L \iff \exists n \in N,\ a^p.u = a^n.b^n \iff u = a^{n-p}.b^n$
$(a^p.b^q)^{-1}.L = (b^q)^{-1}.\left((a^p)^{-1}.L\right)$ or le seul mot de $(a^p)^{-1}.L$ qui commence par un b est b^p. L'ensemble est non vide si et seulement si $p \geqslant q$ et alors son seul élément est b^{p-q}.

Solution de l'exercice 22 - Opérations rationnelles

Démonstration de $u^{-1}(L_1 \cup L_2) \subset u^{-1}L_1 \cup u^{-1}L_2$

Si m appartient à $u^{-1}(L_1 \cup L_2)$ alors $u.m$ appartient à $L_1 \cup L_2$ donc
soit $m.u \in L_1$ d'où $m \in u^{-1}L_1 \subset u^{-1}L_1 \cup u^{-1}L_2$,
soit $m.u \in L_2$ d'où $m \in u^{-1}L_2 \subset u^{-1}L_1 \cup m^{-1}L_2$.
Dans les deux cas on a $m \in u^{-1}L_1 \cup u^{-1}L_2$ d'où $u^{-1}(L_1 \cup L_2) \subset u^{-1}L_1 \cup u^{-1}L_2$.

Démonstration de $u^{-1}L_1 \cup u^{-1}L_2 \subset u^{-1}(L_1 \cup L_2)$

Inversement on a $L_1 \subset L_1 \cup L_2$ donc $u^{-1}L_1 \subset u^{-1}(L_1 \cup L_2)$.
De même $u^{-1}L_2 \subset u^{-1}(L_1 \cup L_2)$ d'où $u^{-1}L_1 \cup u^{-1}L_2 \subset u^{-1}(L_1 \cup L_2)$.
On en déduit l'égalité $u^{-1}(L_1 \cup L_2) = u^{-1}L_1 \cup u^{-1}L_2$

Démonstration de $u^{-1}.(L_1.L_2) \subset (u^{-1}.L_1).L_2 \cup \bigcup_{w \in L_1^{-1}.u} w^{-1}.L_2$

Soit m appartenant à $u^{-1}.(L_1.L_2)$ alors $u.m \in L_1.L_2$: $u.m = u_1.u_2$ avec $u_1 \in L_1$ et $u_2 \in L_2$.

1. Si on a $|u| \leqslant |u_1|$ alors $u_1 = u.v$ donc $v \in u^{-1}L_1$ et $u.m = u.v.u_2$ d'où $m = v.u_2 \in (u^{-1}.L_1).L_2$.
2. Si on a $|u| \geqslant |u_1|$ alors $u = u_1.w$ avec $u_1 \in L_1$ donc $w \in L_1^{-1}.u$.
   On a ensuite $u.m = u_1.w.m = u_1.u_2$ d'où $w.m = u_2 \in L_2$ c'est-à-dire $m \in w^{-1}.L_2$

Démonstration de $(u^{-1}.L_1).L_2 \cup \bigcup_{w \in L_1^{-1}.u} w^{-1}.L_2 \subset u^{-1}.(L_1.L_2)$

1. Si $m \in (u^{-1}.L_1).L_2$ alors $m = u_1.u_2$ avec $u_1 \in u^{-1}.L_1$ donc $u.u_1 \in L_1$ et $u_2 \in L_2$ d'où $u.m = (u.u_1).u_2 \in L_1.L_2$, c'est-à-dire $m \in u^{-1}.(L_1.L_2)$.
2. Si $m \in w^{-1}.L_2$ alors $w.m \in L_2$. Pour $w \in L_1^{-1}.u$, il existe $v \in L_1$ tel que $v.w = u$; on a ainsi $u.m = v.w.m$ avec $v \in L_1$ et $w.m \in L_2$ donc $u.m \in L_1.L_2$ et $m \in u^{-1}.(L_1.L_2)$.

Dans les deux cas on a bien $m \in u^{-1}.(L_1.L_2)$.
On en déduit l'égalité $u^{-1}.(L_1.L_2) = (u^{-1}.L_1).L_2 \cup \bigcup_{w \in L_1^{-1}.u} w^{-1}.L_2$

Démonstration de $u^{-1}.(L_1^*) \subset \bigcup_{w\in(L_1^*)^{-1}.u} \left(w^{-1}.L_1\right).L_1^*$

Soit m appartenant à $u^{-1}.(L_1^*)$ alors $u.m \in L_1^*$: $u.m = u_1.u_2.\ldots.u_p$ avec $u_i \in L_1$ pour tout i.
Il existe donc un indice $k \geqslant 2$ tel que $|u_1|+\cdots+|u_{k-1}| \leqslant |u| < |u_1|+\cdots+|u_{k-1}|+|u_k|$. On en déduit que $u_k = w_k.v_k$ (w_k pouvant être le mot vide) avec $u = u_1.\ldots.u_{k-1}.w_k$ et $m = v_k.u_{k+1}\ldots.u_p$.
Or $u_1.\ldots.u_{k-1}$ appartient à L_1^* donc $w_k \in (L_1^*)^{-1}.u$.
$w_k.v_k = u_k \in L_1$ donc $v_k \in w_k^{-1}.L_1$ puis $m = v_k.u_{k+1}.\cdots u_p \in \left(w_k^{-1}.L_1\right).L_1^*$.
Ainsi $m \in \bigcup_{w\in(L_1^*)^{-1}.u} \left(w^{-1}.L_1\right).L_1^*$, ce qui prouve l'inclusion.

Démonstration de $\bigcup_{w\in(L_1^*)^{-1}.u} \left(w^{-1}.L_1\right).L_1^* \subset u^{-1}.(L_1^*)$

Si $m \in \left(w^{-1}.L_1\right).L_1^*$ avec $w \in (L_1^*)^{-1}.u$ alors $u = u'.w$ avec $u' \in L_1^*$ et $m = v.u''$ avec $v \in w^{-1}.L_1$ donc $w.v \in L_1$ et $u'' \in L_1^*$. On a alors $u.m = (u'.w).(v.u'') = u'.(w.v).u''$ avec $u' \in L_1^*$, $w.v \in L_1$ et $u'' \in L_1^*$ d'où $u.m \in L_1^*$ puis $m \in u^{-1}.(L_1^*)$.

Solution de l'exercice 23 - Rationalité
On montre par induction structurelle que $u^{-1}.L$ est rationnel pour tout mot u si L est rationnel en utilisant l'exercice ci-dessus car les dérivés de $L_1 \cup L_2$, $L_1.L_2$ et L_1^* s'expriment avec des opérations rationnelles en fonctions des dérivés de L_1 et de L_2.

Solution de l'exercice 24 - Critère de rationalité
Ici encore on procède par induction structurelle.

$\emptyset$ n'admet que $\emptyset$ comme langage dérivé.
$\{\varepsilon\}$ n'admet que $\emptyset$ ou $\{\varepsilon\}$ comme langage dérivé.
Pour $a \in \mathcal{A}$, $\{a\}$ n'admet que $\emptyset$, $\{\varepsilon\}$ ou $\{a\}$ comme langage dérivé.
Les langages élémentaires n'admettent donc qu'un nombre fini de langages dérivés.

On suppose que L_1 n'admet qu'un nombre fini de langage dérivés, notés $L'_{1,i}$ pour $1 \leqslant i \leqslant n_1$, et que L_n n'admet qu'un nombre fini de langage dérivés, notés $L'_{2,i}$ pour $1 \leqslant i \leqslant n_2$

1. Les dérivés de $L_1 \cup L_2$ sont parmi les ensembles $L'_{1,i} \cup L'_{2,j}$; il y en a $n_1.n_2$ au plus.
2. Les dérivés de $L_1.L_2$ sont parmi les ensembles $L'_{1,i}.L_2 \cup \Lambda_{2,J}$ avec $\Lambda_{2,J} = \bigcup_{j\in J} L'_{2,j}$ pour $J \subset \{1, 2, \ldots, n_2\}$; il y en a $n_1.2^{n_2}$ au plus.
3. Les dérivés de L_1^* sont parmi les ensembles $\Lambda^*_{1,I}$ avec $\Lambda^*_{1,J} = \bigcup_{j\in J} L'_{1,j}.L_1^*$ pour $I \subset \{1, 2, \ldots, n_1\}$; il y en a 2^{n_1} au plus.

Ainsi la finitude du nombre de dérivés se transmet par les opérations rationnelles : elle est vraie pour tout langage rationnel.

Solution de l'exercice 25 - Contre-exemple
On a vu que L avait une infinité de dérivés.

Solution de l'exercice 26
Si $u \ll v$ on note $\delta(u, v)$ l'entier k qui intervient dans la définition.

Réflexivité u est un préfixe de u donc $u \preceq u$ pour tout mot u.

Antisymétrie Si on a $u \ll v$ alors on ne peut pas avoir $v \ll u$ par définition. De plus u et v diffèrent en $\delta(u, v)$ donc on ne peut pas avoir v préfixe de u (ni u préfixe de v). Ainsi on ne peut pas avoir $v \preceq u$ si $u \ll v$.
On en déduit que si on a $u \preceq v$ et $v \preceq u$ alors u est un préfixe de v et v est un préfixe de u. Dans ce cas on $v = u.u'$ et $u = v.v'$ donc $v = v.v'.u'$ d'où $v'.u' = \varepsilon$ et ona vu qu'alors $v' = u' = \varepsilon$. On conclut qu'on a $u = v$

Transitivité On suppose qu'on a $u \preceq v$ et $v \preceq w$.

- Si u est un préfixe de v et v un préfixe de w alors u est un préfixe de w : $u \preceq w$.
- Si u est un préfixe de v et si $v \ll w$, il y a deux cas.
  Si $|u| < \delta(v,w)$ alors u est un préfixe de w donc $u \preceq w$.
  Si $|u| \geqslant \delta(v,w)$ alors $u \ll w$ avec $\delta(u,w) = \delta(v,w)$.
- Si $u \ll v$ et si v est un préfixe de w alors $u \ll w$ avec $\delta(u,w) = \delta(u,v)$.
- Si $u \ll v$ et si $v \ll w$ alors $u \ll w$ avec $\delta(u,w) = \min\{\delta(u,v), \delta(v,w)\}$.

La relation est bien une relation d'ordre.

u et v sont deux mots tels que $|u| \leqslant |v|$.
Si u n'est pas un préfixe de v alors il existe un indice k qui est le premier en lequel les lettres de u, x_k, et de v, y_k, diffèrent. Selon qu'on a $x_k < y_k$ ou $y_k < x_k$ on aura $u \ll v$ ou $v \ll u$. Dans tous les cas u et v sont comparables.
De même si on a $|v| \leqslant |u|$: la relation d'ordre est totale.

Solution de l'exercice 27

```
let rec inferieur l1 l2 =
   match l1, l2 with
   |_, [] -> false
   |[], _ -> true
   |t1::q1, t2::q2 when t1 < t2 -> true
   |t1::q1, t2::q2 when t1 = t2 -> inferieur q1 q2
   |t1::q1, t2::q2 -> false;;
```

Solution de l'exercice 28

```
let rec decoupe liste i =
   if i = 0
   then [], liste
   else match liste with
        |[] -> failwith "La liste est trop courte"
        |t::q -> let l1, l2 = decoupe q (i-1) in (t::l1), l2;;
```

```
let conjugue liste i =
   let l1, l2 =  decoupe liste i in l2@l1;;
```

Solution de l'exercice 29

```
let lyndon liste =
   let n = List.length liste in
   let reponse = ref true in
   for i = 1 to (n-1) do
      if inferieur liste (conjugue liste i))
      then reponse := false done;
   !reponse;;
```

Solution de l'exercice 30
Soit $u = w.v = v.w'$.

- Si $w = w'$, u est égal à un de ses conjugué : il n'est pas un mot de Lyndon.
- Si $w \neq w'$, comme w et w' sont de même longueur, w n'est pas un préfixe de w' et w' n'est pas un préfixe de w ; on a donc $w \ll w'$ ou $w' \ll w$.
  - $w \ll w'$ implique $v.w \ll v.w' = u$ avec $v.w$ conjugué de u.
  - $w' \ll w$ implique $w'.v \ll w.v = u$ avec $w'.v$ conjugué de u.

Dans les deux cas u n'est pas un mot de Lyndon.

Solution de l'exercice 31

- Si $u \prec h$ pour tout suffixe propre alors, comme $h \prec h.w$, on a $u \prec h.w$ pour $u = w.h$, u est de Lyndon.
- Si u est un mot de Lyndon on a $u \prec h.w$ pour $u = w.h$. On sait que h ne peut pas être un préfixe donc une lettre au moins diffère entre u et h ; dans ce cas on doit avoir $u \prec h$.

Solution de l'exercice 32

- On suppose que u est un mot de Lyndon.
  Soit g le plus long suffixe propre de u qui soit un mot de Lyndon ; on a $u \prec g$.
  On a alors $u = f.g$ et $f \prec f.g = u \prec g$.
  Soit h un suffixe de f ; $h.g$ n'est pas un mot de Lyndon en raison de la maximalité de g donc il existe un suffixe h' de $h.g$ telle que $h' \prec h.g$.
  La remarque essentielle est que h ne peut pas être un préfixe de h' car sinon on peut écrire $h' = h.t$ et $h' \prec h.g$ implique $t \prec g$ alors que t est un suffixe du mot de Lyndon g : c'est impossible.
  $h' \prec h.g$ avec h' non préfixe de h implique soit h' préfixe de h soit il existe une lettre distincte à la position k pour h et h' donc $h \ll h'$: dans les deux cas on a $h' \prec h$.
  g est un suffixe de u, mot de Lyndon, et h' est un suffixe de g, mot de Lyndon, donc on a $f \prec f.g = u \prec g \prec h' \prec h$ pour tout suffixe de f donc f est un mot de Lyndon.
- On suppose que f et g sont deux mots de Lyndon tels que $f \prec g$.
  - Si f est un préfixe de g alors $g = f.f'$ donc $g \prec f'$. On en déduit $f.g \prec f.f' = g$
  - Sinon $f \ll g$, ce qui implique $f.g \ll g$.

  On a toujours $f.g \ll g$.
  h est un suffixe de $u = f.g$.
  - Si h est un suffixe propre de g alors $g \prec h$ donc $u = f.g \prec g \prec h$.
  - Si $h = g$ on a prouvé $u = f.g \prec g = h$.
  - Si g est un suffixe propre de h alors $h = h'.g$ avec h' suffixe propre de f. On a alors $f \ll h'$ donc $u = f.g \ll h'.g = h$.

  Dans tous les cas on a $u \prec h$: u est un mot de Lyndon.

Solution de l'exercice 33
On écrit les mots de Lyndon de taille 1 à 4

```
[0]  [1]
[0;  1]
[0;  0;  1]  [0;  1;  1]
[0;  0;  0;  1]  [0;  0;  1;  1]]  [0;  1;  1;  1]
```

On combine ensuite pour les mots de taille 5

```
[0;  0;  0;  0;  1]  [0;  0;  0;  1;  1]  [0;  0;  1;  1;  1]
[0;  1;  0;  1;  1]  [0;  0;  1;  0;  1]  [0;  1;  1;  1;  1]
```

Solution de l'exercice 34
On commence par l'insertion ordonnée

```
let rec insere_mot u liste =
  match liste with
  |[] -> [u]
  |t::q when t = u -> liste
  |t::q when inferieur u t -> u::liste
  |t::q -> t :: (insere_mot u q);;
```

On généralise à une liste

```
let rec insere_liste liste1 liste2 =
  match liste1 with
  |[] -> liste2
  |t::q -> insere_liste q (insere_mot t liste2);;
```

On fusionne un mot à une liste en ne gardant que les mots de Lyndon

```
let rec fusionne_mot u liste =
  match liste with
  |[] -> []
  |t::q when inferieur u t -> (u@t) :: (fusionne_mot u q)
  |t::q -> fusionne_mot u q;;
```

On peut alors passer aux listes

```
let rec fusionne_listes liste1 liste2 =
  match liste1 with
  |[] -> []
  |t::q -> insere_liste (fusionne_mot t liste2) (fusionne_listes q
      liste2) ;;
```

```
let lyndon n =
  let lynd = Array.make (n+1) [] in
  lynd.(1) <- [[0]; [1]];
  for i = 2 to n do
    for k = 1 to (i - 1) do
      let liste = fusionne_listes lynd.(k) lynd.(i - k) in
      lynd.(i) <- insere_liste liste lynd.(i)
    done;
  done;
  lynd;;
```

Solution de l'exercice 35 -
Par récurrence sur la longueur de u.
Si u est de longueur 1, la seule décomposition possible est u.
On suppose que tout mot de longueur au plus n admette au plus 1 décomposition de Lyndon.
Soit u de longueur $n+1$ admettant les décomposition de Lyndon
$u = u^{(1)}.\cdots.u^{(p)}$ avec $u^{(i)}$ mot de Lyndon et $u^{(i)} \preceq u^{(i-1)}$,
$u = v^{(1)}.\cdots.v^{(q)}$ avec $v^{(j)}$ mot de Lyndon et $v^{(j)} \preceq v^{(j-1)}$.
On suppose $|u^{(1)}| < |v^{(1)}|$ donc $u^{(1)}$ est un préfixe propre de $v^{(1)}$: $u^{(1)} \prec v^{(1)}$.
De plus $v^{(1)} = u^{(1)}.\cdots.u^{(k)}.w$ avec w préfixe non vide de $u^{(k+1)}$.
w est un suffixe propre de $v^{(1)}$ donc $v^{(1)} \prec w$. w est un préfixe de $u^{(k+1)}$ donc $w \preceq u^{(k+1)}$.
On a $u^{(k+1)} \preceq u^{(k)} \preceq \cdots \preceq u^{(1)}$. Ainsi $v^{(1)} \prec u^{(1)}$ ce qui est incompatible avec $u^{(1)} \prec v^{(1)}$.

De même $|u^{(1)}| > |v^{(1)}|$ est impossible donc $|u^{(1)}| = |v^{(1)}|$ puis $u^{(1)} = v^{(1)}$.
$u^{(2)}.\cdots.u^{(p)}$ et $v^{(2)}.\cdots.v^{(q)}$ sont alors deux décompositions de Lyndon d'un même mot de longueur n au plus, elle sont égales d'après l'hypothèse de récurrence donc les deux décompositions de u sont égales.

Solution de l'exercice 36

```
let factorisation liste =
   let init = List.map (fun x -> [x]) liste in
   let rec diminuer liste =
      match liste with
      |[] -> [], false
      |[u] -> [u], false
      |u::v::reste when inferieur u v -> (u@v)::reste, true
      |t::q -> let q', b = diminuer q in t::q', b in
   let rec traiter liste =
      let liste', b = diminuer liste in
      if b
      then traiter liste'
      else liste' in
   traiter init;;
```

Solution de l'exercice 37 - 3 lettres
On utilise un résultat ci-dessus. On isole les lettres c : les mots de L sont le la forme $u_0.c.u_1.c.\cdots.u_{n-1}.c.u_n$ où u_i est un mot du langage sur l'alphabet $\{a, b\}$ tels que deux lettres consécutives sont toujours distinctes.
On note `r1=(ϵ|b).(a.b)*.(ϵ|a)`.
On doit avoir $u_i \neq \varepsilon$ pour $1 \leqslant i \leqslant n-1$ dans ce cas l'ensemble peut être dénoté par `r2 = a.(b.a)*(ϵ|b)|b.(a.b)*.(ϵ|a)`
Une expression régulière dénotant le langage est donc `r1.(c.r2)*.c.r1|r1`.

Solution de l'exercice 38 - Complémentaire
Le complémentaire est l'ensemble des mots qui ne contiennent pas aba.
Si on isole les a et les b toute suite non vide de a doit être suivie d'une suite d'au moins deux b donc les facteurs de base sont de la forme $a^n b^p$ avec $n \geqslant 1$ et $p \geqslant 2$. De plus on peut commencer par des b et finir par des a puis des b d'où $\overline{L}$ est dénoté par `b*.(a.a*.b.b.b*)*.a*.b*`.

Solution de l'exercice 39 - Expressions équivalentes

- Pour la seconde expression on isole le lettres b ; elles sont séparées par des a^k.
  Pour la troisième on a séparé les mots ne contenant pas de b.
  Pour la dernière on isole les lettres.
- Les mots du langage dénoté par `(r1.r2)*` sont soit le mot vide soit de la forme $u_1.v_1.u_2.v_2.\cdots.u_n.v_n$ avec $u_1 \in L[\texttt{r1}]$, $v_n \in L[\texttt{r2}]$ et $v_1.u_2.v_2.\cdots.u_n \in L[(\texttt{r1.r2})*]$.

Solution de l'exercice 40 - Détermination d'expressions régulières

- `(a|b)*.a.(a|b)*`
- `b*|b*.a.b*`
- `(a.a|b)*`
- `((a|b).(a|b).(a|b))*.(a|b|a.a|a.b|b.a|b.b)`
- `(ϵ|b).(a.b)*.(ϵ|a)` ou `(a.b)* | a.(a.b)* | a.(a.b)*.b | (a.b)*.b`
- S'il y a un a avant un b alors il y a un facteur ab, sinon il y a un facteur ba d'où l'expression régulière `(a|b)*.(a.b|b.a).(a|b)*`
  On peut aussi distinguer la première lettre : `(a.a*.b|b.b*.a).(a|b)*`

Solution de l'exercice 41 - Élimination de zéro
$P(L)$ est la propriété :
L est vide ou est dénoté par un expression régulière ne contenant pas $\varnothing$.

- Pour les langages élémentaires la propriété est évidente.
- On suppose que $P(L_1)$ et $P(L_2)$ sont vraies.
  - Si L_1 est vide alors $L_1 \cup L_2 = L_2$ donc $P(L_1 \cup L_2)$ est vraie
  - De même si L_2 est vide
  - Si L_1 et L_2 sont non vides alors ils sont dénotés par des expressions régulières `r1` et `r2` ne contenant pas $\varnothing$. $L_1 \cup L_2$ est alors dénoté par `(r1|r2)` qui ne contient pas $\varnothing$. Ainsi $P(L_1 \cup L_2)$ est vraie.
- On suppose que $P(L_1)$ et $P(L_2)$ sont vraies.
  - Si L_1 ou L_2 est vide alors $L_1.L_2$ est vide donc $P(L_1.L_2)$ est vraie
  - Si L_1 et L_2 sont non vides alors ils sont dénotés par des expressions régulières `r1` et `r2` ne contenant pas $\varnothing$. $L_1.L_2$ est alors dénoté par `(r1.r2)` qui ne contient pas $\varnothing$. Ainsi $P(L_1 \cup L_2)$ est vraie.
- On suppose que $P(L)$ est vraie.
  - Si L est vide alors $L^* = \{\varepsilon\}$ est dénoté par ϵdonc $P(L^*)$ est vraie.
  - Si L est non vide alors il est dénoté par une expression régulière `r` ne contenant pas $\varnothing$. L^* est alors dénoté par `(r*)` qui ne contient pas $\varnothing$. Ainsi $P(L^*)$ est vraie.

Solution de l'exercice 42 - Réduction
La démonstration se fait par induction structurelle.
La propriété est immédiate pour les langages élémentaires.
Dans le tableau on étudie les cas d'expressions régulières qui dénotent L_1 et L_2 et on donne une expression régulière pour $L_1 \cup L_2$, $L_1.L_2$ et L_1^*. `r1` et `r2` sont réduites.

L_1	L_2	$L_1 \cup L_2$	$L_1.L_2$	L_1^*
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	ϵ
$\varnothing$	ϵ	ϵ	$\varnothing$	ϵ
$\varnothing$	r2	r2	$\varnothing$	ϵ
$\varnothing$	r2|ϵ	r2|ϵ	$\varnothing$	ϵ
ϵ	$\varnothing$	ϵ	$\varnothing$	ϵ
ϵ	ϵ	ϵ	ϵ	ϵ
ϵ	r2	r2|ϵ	r2	ϵ
ϵ	r2|ϵ	r2|ϵ	r2|ϵ	ϵ
r1	$\varnothing$	r1	$\varnothing$	r1*
r1	ϵ	r1|ϵ	r1	r1*
r1	r2	r1|r2	r1.r2	r1*
r1	r2|ϵ	r1|r2|ϵ	r1.r2|r1	r1*
r1|ϵ	$\varnothing$	r1|ϵ	$\varnothing$	r1*
r1|ϵ	ϵ	r1|ϵ	r1|ϵ	r1*
r1|ϵ	r2	r1|r2|ϵ	r1.r2|r2	r1*
r1|ϵ	r2|ϵ	r1|r2|ϵ	r1.r2|r1|r2|ϵ	r1*

Solution de l'exercice 43

```
let rec ecrire arbre =
  match arbre with
  |Vide -> "0"
  |Eps -> "1"
  |Feuille c -> c
  |Etoile f ->  "(" ^ (ecrire f) ^ "*)"
  |Union (fg, fd) -> "(" ^ (ecrire fg) ^ "|" ^ (ecrire fd) ^ ")"
  |Produit (fg, fd) -> "(" ^ (ecrire fg) ^ "." ^ (ecrire fd) ^ ")"
     ;;
```

Solution de l'exercice 44 - Langage vide

```
let rec lVide r =
  match r with
  |Zero -> true
  |Un -> false
  |Lettre _ -> false
  |Union(e1,e2) -> langageVide e1 && langageVide e2
  |Produit(e1,e2) -> langageVide e1 || langageVide e2
  |Etoile e -> false;;
```

Solution de l'exercice 45 - Mot vide

```
let rec motVide r =
  match r with
  |Zero -> false
  |Un -> true
  |Lettre _ -> false
  |Union(r1, r2) -> motVide r1 || motVide r2
  |Produit(r1, r2) -> motVide r1 && motVide r2
  |Etoile r -> true;;
```

Solution de l'exercice 46 - Transformations, reprise de l'exercice 15

```
let rec sans_a a r =
  match r with
  |Zero -> Zero
  |Un -> Un
  |Lettre x when x = a -> Zero
  |Lettre x -> Lettre x
  |Union(r1, r2) -> Union(sans_a a r1, sans_a a r2)
  |Produit(r1, r2) ->  Produit(sans_a a r1, sans_a a r2))
  |Etoile r -> Etoile (sans_a a r);;
```

```
let rec avec_a a r =
  match r with
  |Zero -> Zero
  |Un -> Zero
  |Lettre x when x = a -> Lettre a
  |Lettre x -> Zero
  |Union(r1, r2) -> Union(avec_a a r1, avec_a a r2)
  |Produit(r1, r2) ->  Union(Produit(avec_a a r1, r2), Produit(r1,
      avec_a r2))
  |Etoile r -> Produit(Produit(Etoile r, avec_a a r), Etoile r);;
```

```
let rec  sansPrem_a a r =
  match r with
  |Zero -> Zero
  |Un -> Zero
  |Lettre x when x = a -> Un
  |Lettre x -> Zero
  |Union(r1, r2) -> Union(sansPrem_a a r1, sansPrem_a a r2)
  |Produit(r1, r2) ->  Union(Produit(sansPrem_a a r1, r2), Produit
     (sans_a a r1, sansPrem_a r2))
  |Etoile r -> Produit(Produit(Etoile(sans_a a r), sansPrem_a a r)
     , Etoile r);;
```

```
let rec  sansUn_a a r =
  match r with
  |Zero -> Zero
  |Un -> Zero
  |Lettre x when x = a -> Un
  |Lettre x -> Zero
  |Union(r1, r2) -> Union(sansUn_a a r1, sansUn_a a r2)
  |Produit(r1, r2) ->  Union(Produit(sansUn_a a r1, r2), Produit(
     r1, sansUn_a r2))
  |Etoile r -> Produit(Produit(Etoile r, sansUn_a a r), Etoile r)
     ;;
```

Solution de l'exercice 47 - Prefixes, reprise de l'exercice 15

```
let rec prefixes r =
  match r with
  |Zero -> Zero
  |Un -> Un
  |Lettre x -> Union(Un, Lettre x)
  |Union(r1, r2) -> Union(prefixes r1, prefixes r2)
  |Produit(r1, r2) ->  Union(prefixes r1, Produit(r1, prefixes r2)
     )
  |Etoile r -> Produit(Etoile r, prefixes r);;
```

Solution de l'exercice 48 - Vers les langages locaux

```
let rec prefixes1 r =
   match r with
   |Zero -> []
   |Un -> []
   |Lettre a -> [a]
   |Union(r1, r2) -> (prefixes1 r1)@(prefixes1 r2)
   |Produit(r1, r2) -> if motVide r1
                       then (prefixes1 r1)@(prefixes1 r2)
                       else prefixes1 r1
   |Etoile r -> prefixes1 r;;
```

```
let rec suffixes1 r =
   match r with
   |Zero -> []
   |Un -> []
   |Lettre a -> [a]
   |Union(r1, r2) -> (suffixes1 r1)@(suffixes1 r2)
   |Produit(r1, r2) -> if motVide r2
                      then (suffixes1 r1)@(suffixes1 r2)
                      else suffixes1 r1
   |Etoile r -> suffixes1 r;;
```

```
let rec assembler liste1 liste2 =
   let coller x y = (string_of_char x)^(string_of_char y) in
   let ajouter x = map (coller x) liste2 in
   match liste1 with
   |[] -> []
   |t::q -> (ajouter t)@(assembler q liste2);;
```

```
let rec facteurs2 r =
   match r with
   |Zero -> []
   |Un -> []
   |Lettre a -> []
   |Union(r1, r2) -> (facteurs2 r1)@(facteurs2 r2)
   |Produit(r1, r2) -> ((facteurs2 r1)@(facteurs2 r2))
                      @(assembler (suffixes1 r1) (prefixes1 r2))
   |Etoile r ->  (facteurs2 r)
                @(assembler (suffixes1 r) (prefixes1 r));;
```

Solution de l'exercice 49 - Réduction, reprise de l'exercice 42
Pour gérer le grand nombre de cas, on sépare les union, produits et étoile d'expression réduites.

```
let etoile_reduite r =
   match r with
   |Zero -> Un
   |Un -> Un
   |Union(r1, Un) -> Etoile r1
   |_ -> Etoile r;;
```

```
let union_reduite r1 r2 =
   match r1, r2 with
   |Zero, _ -> r2
   |_, Zero -> r1
   |Un, Un -> Un
   |Un, Union(r4, Un) -> Union(r4, Un)
   |Un, _ -> Union(r2, Un)
   |Union(r3, Un), Un -> Union(r3, Un)
   |_, Un -> Union(r1, Un)
   |Union(r3, Un), Union(r4, Un) -> Union(Union(r3, r4), Un)
   |Union(r3, Un), _ -> Union(Union(r3, r2), Un)
   |_, Union(r4, Un) -> Union(Union(r1, r4), Un)
   |_ -> Union(r1, r2);;
```

```
let produit_reduit r1 r2 =
   match r1, r2 with
   |Zero, _ -> Zero
   |_, Zero -> Zero
   |Un, _ -> r2
   |_, Un -> r1
   |Union(r3, Un), Union(r4, Un)
      -> Union(Union(Union(Produit(r3, r4), r3), r4), Un)
   |Union(r3, Un), _ -> Union(Produit(r3, r2), r2)
   |_, Union(r4, Un) -> Union(Produit(r1, r4), r1)
   |_ -> Produit(r1, r2);;
```

La fonction elle-même est alors simple

```
let rec reduire r =
  match r with
  |Union(r1, r2) -> union_reduite (reduire r1) (reduire r2)
  |Produit(r1, r2) -> produit_reduit (reduire r1) (reduire r2)
  |Etoile r1 -> etoile_reduite (reduire r1)
  |_ -> r;;
```