Chapitre 3. Mots et langages.

1 Introduction et objectifs du chapitre

Ce chapitre traite des mots et des langages, un mot étant une séquence finie de symboles et un langage étant un ensemble de mots. La question qui nous intéressera le plus est la suivante : un mot donné appartient-il à un langage donné ?

Commençons par présenter quelques-uns des grands domaines d'application de ces notions, afin de saisir l'utilité de la théorie qui sera ensuite développée.

Exemple 1 : Compilation

La compilation désigne la tâche consistant à transformer un ensemble de commandes écrites dans un langage de programmation de haut niveau (comme Python ou OCaml) en une suite d'instructions exécutables par l'ordinateur (c'est-à-dire exprimées en langage machine).

Pour commencer, le compilateur devra identifier les séquences de caractères qui forment des mots-clés du langage ou des noms de variables licites ou encore des nombres (flottants, entiers...) : cette étape est l'*analyse lexicale*.

Ensuite, le compilateur doit contrôler qu'il n'y a pas d'erreurs de syntaxe dans le programme en vérifiant par exemple que les expressions arithmétiques sont bien formées, que les construction du langage sont respectées : c'est l'*analyse syntaxique*. En fait, le rôle de l'analyseur syntaxique va au delà de ces contrôles puisqu'il doit également transformer le programme en une suite d'instructions exécutables par la machine.

Exemple 2 : Bio-informatique

La génétique donne un exemple naturel d'objets modélisables par des séquences de symboles sur un alphabet fini. Par exemple, chaque chromosome porteur du capital génétique est essentiellement formé de deux brins d'ADN ; chacun de ces brins est une succession de nucléotides qui sont composés d'un phosphate, d'un sucre et d'une base azotée. Il existe quatre bases différentes : la guanine G, l'adénine A, la cytosine C et la thymine T. L'information encodée dans ces bases déterminant une partie importante de l'information génétique, une modélisation utile d'un brin de chromosome consistera en une séquence des bases qui le composent, soit un mot très long sur l'alphabet à quatre lettres {G,T,A,C}.

De cette modélisation découlent plusieurs questions : comment déterminer la présence ou non d'un gène dans le patrimoine génétique d'un individu (autrement dit rechercher une séquence particulière de nucléotides dans un chromosome) ou encore détecter des ressemblances entre plusieurs fragments d'ADN. Ces ressemblances vont servir par exemple à localiser des gènes remplissant des mêmes fonctions ou encore à faire des tests de familiarité entre plusieurs individus.

Rechercher des séquences ou mesurer des ressemblances sont donc deux problèmes de base de la bio-informatique.

Exemple 3 : Recherche de motifs dans un texte

La fonction "rechercher" des éditeurs de texte propose de déterminer les occurrences d'une chaîne de caractères dans un fichier texte. Certains proposent même une fonction élargie permettant de rechercher tout un ensemble de séquences vérifiant certaines

propriétés : dans ce cas, on utilise la notion d'*expression régulière* pour exprimer l'objet de la recherche. Par exemple l'expression régulière `(19|20)[0-9]{2}` décrit tous les entiers entre 1900 et 2099. L'expression

`[a-z]+(\.[a-z]+)@([a-z]+)\.fr`

décrit certaines adresses email en .fr

Notons que le motif de la recherche n'est pas forcément alphanumérique : par exemple on peut vouloir rechercher puis lire un code-barre dans une image . Il peut aussi être mutidimensionnel comme par exemple un QR code mais dans ce cours on se limitera à la recherche de motifs en dimension 1.

2 Mots

2.1 Alphabet et mots

Définition 1.

- *Un* alphabet Σ *est un ensemble fini. Les éléments de* Σ *sont appelés* lettres *ou* caractères.
- *Un* mot *sur un alphabet* Σ *est soit le mot vide noté* ε *(ou* 1_Σ*), soit une suite finie* $u_1u_2\dots u_n$ *avec pour tout* $k \in [\![1,n]\!]$, $u_k \in \Sigma$.
- *L'ensemble des mots non vides sur l'alphabet* Σ *est noté* Σ^+ *et on note* $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.
- *Si* u *est un mot sur l'alphabet* Σ*, on appelle* longueur *de* u*, notée* $|u|$ *l'entier qui vaut* 0 *si* $u = \varepsilon$ *et qui vaut* n *si* $u = u_1u_2\dots u_n$ *avec* $\forall\, k \in [\![1,n]\!]$, $u_k \in \Sigma$.
- *Si* $n \in \mathbb{N}$*, l'ensemble des mots de longueur* n *sur l'alphabet* Σ *est noté* Σ^n.
- *Si* u *est un mot sur l'alphabet* Σ *et* $a \in \Sigma$*, on appelle longueur en* a *de* u *et on note* $|u|_a$ *l'entier égal à* 0 *si* $u = \varepsilon$ *et égal à* $card\{k \in [\![1,n]\!] \,|\, u_k = a\}$ *si* $u = u_1u_2\dots u_n$ *avec* $\forall\, k \in [\![1,n]\!]$, $u_k \in \Sigma$. *Autrement dit* $|u|_a$ *est le nombre d'occurences de la lettre* a *dans le mot* u.

Exemple : Sur l'alphabet $\Sigma = \{a,b,c,d\}$, $u = cabada$ est de longueur $|u| = 6$ et sa longueur en a est $|u|_a = 3$.

Remarques : − on a $\Sigma^0 = \{\varepsilon\}$, $\Sigma^1 = \Sigma$, $\Sigma^* = \bigcup_{n\in\mathbb{N}} \Sigma^n$ et $\Sigma^+ = \bigcup_{n\in\mathbb{N}^*} \Sigma^n$.
− Pour tout mot u sur l'alphabet Σ, $|u| = \sum_{a\in\Sigma} |u|_a$.
− On utilise souvent les lettres du début de l'alphabet courant $\{a,b,c,\dots\}$ pour désigner des lettres et celles de la fin de l'alphabet pour désigner des mots $(u,v,w,...)$.

Définition 2 (Concaténation)**.**
Si $u = u_1\dots u_p$ *et* $v = v_1\dots v_q$ *sont deux mots sur l'alphabet* Σ *de longueurs* p *et* q*, on appelle* concaténation *de* u *et* v *le mot de longueur* $n+p$ *noté* $u.v$ *ou* uv *égal à*

$$u.v = u_1\dots u_pv_1\dots v_q.$$

Par ailleurs, pour tout mot $u \in \Sigma^*$, $u.\varepsilon = \varepsilon.u = u$.

Proposition
La concaténation est une loi de composition interne associative sur Σ^*, d'élément neutre ε.

Remarques : − Grâce à l'associativité de la concaténation, si $u \in \Sigma^*$, on peut définir par récurrence le mot u^n pour $n \in \mathbb{N}$ par : $u^0 = \varepsilon$ et pour tout $n \in \mathbb{N}$, $u^{n+1} = u^n.u$.
On a alors pour tout $(n,p) \in \mathbb{N}^2$, $u^{n+p} = u^n.u^p = u^p.u^n$.
− Seul le mot vide possède un symétrique pour la concaténation, mais tout mot est simplifiable à gauche ou à droite (c'est-à-dire que $uv = uw$ ou $vu = wu$ entraîne $v = w$)

2.2 Facteurs et sous-mots

Définition 3. *Si u et v sont deux mots sur l'alphabet Σ, on dit que u est un*
- préfixe *de v s'il existe un mot $w \in \Sigma^*$ tel que $v = uw$*
- suffixe *de v s'il existe un mot $w \in \Sigma^*$ tel que $v = wu$*
- facteur *de v s'il existe deux mots x et y de Σ^* tel que $v = xuy$*

Un préfixe (resp. suffixe) est dit propre si $w \neq \varepsilon$; un facteur est dit propre si $x \neq \varepsilon$ ou $y \neq \varepsilon$.
Enfin, un sous-mot *d'un mot $u = u_1 \dots u_n$ de longueur n est un mot $v = v_1 \dots v_p$ pour lequel il existe une application φ strictement croissante de $[\![1, p]\!]$ dans $[\![1, n]\!]$ telle que pour tout $k \in [\![1, p]\!]$, $v_k = u_{\varphi(k)}$.*

Exemples : "strophe" et "roi" sont des sous-mots de "claustrophobie" mais n'en sont pas des facteurs.

Quelques résultats

Lemme (Levi) Soient x, y, u, v quatre mots sur Σ tels que $xy = uv$. Alors il existe un unique mot $t \in \Sigma^*$ tel que l'une des deux conditions suivantes soit réalisée :
$-$ $u = xt$ et $y = tv$
$-$ $x = ut$ et $v = ty$

Shéma illustrant cette propriété :

Démonstration : $\diamond$ **Existence** Supposons par exemple que $|u| \geqslant |x|$. Alors x, qui est un préfixe de uv, est un préfixe de u donc il existe $t \in \Sigma^*$ tel que $u = xt$. Mais alors, l'égalité $xy = uv$ s'écrit $xtv = xy$ donc, en simplifiant, $tv = y$.
Le cas où $|u| \leqslant |x|$ se traite de manière analogue.
$\diamond$ **Unicité** S'il existe deux mots s et t tels que $u = xt = xs$ et $y = tv = sv$, alors $xt = xs$ donc $t = s$.
De même s'il existe deux mots s, t tels que $x = ut = us$ et $v = ty = sy$.
Enfin s'il existe deux mots t et s tels que $u = xt$, $y = tv$, $x = us$ et $v = sy$, alors $|u| = |x|$ donc $|t| = |s| = 0$ donc $t = s = \varepsilon$ ♯

Corollaire Si x et y sont deux préfixes d'un mot u, alors x est préfixe de y ou y est préfixe de x.

Théorème Soient $x, y, z \in \Sigma^*$ tels que $xy = yz$ et $x \neq \varepsilon$. Alors il existe deux mots u, v et $k \in \mathbb{N}$ tels que :

$$x = uv,\ y = (uv)^k u = u(vu)^k \text{ et } z = vu$$

Démonstration : $\diamond$ Si $|x| \geqslant |y|$, on peut appliquer le théorème de Levi : il existe $t \in \Sigma^*$ tel que $x = yt$ et $z = ty$ d'où le résultat souhaité avec $u = y$, $v = t$ et $k = 0$.
$\diamond$ Si $|x| \leqslant |y|$ on montre le résultat par récurrence sur $|y|$.
- Si $|y| = 1$, on a nécessairement $|x| = 1$ (car $x \neq \varepsilon$) et alors $x = y = z$. Le résultat est acquis avec $u = y$, $v = \varepsilon$ et $k = 0$.
- Soit $j \in \mathbb{N}^*$ tel que le résultat soit vrai pour tout mot y de longueur $\leqslant j$ et soit y un mot de longueur égale à $j + 1$. D'après le lemme de Levi, il existe $t \in \Sigma^*$ tel que $y = xt$ et $y = tz$. On a donc $xt = tz$ et puisque $x \neq \varepsilon$, $|t| \leqslant j$. D'après l'hypothèse de récurrence, il existe deux mots u et v et $k \in \mathbb{N}$ tels que $x = uv$, $t = (uv)^k u = u(vu)^k$ et $z = vu$. Il reste à calculer $y = xt = tz$ pour obtenir $y = (uv)^{k+1} u = u(vu)^{k+1}$ ♯

Théorème Soient x et y deux mots sur Σ tels que $xy = yx$, $x \neq \varepsilon$ et $y \neq \varepsilon$.
Alors il existe un mot $u \in \Sigma^*$ et $(i, j) \in \mathbb{N}^2$ tels que $x = u^i$ et $y = u^j$.

Démonstration : Par récurrence sur $|xy|$
$\diamond$ Si $|xy| = 2$, alors $|x| = |y| = 1$ et $x = y$. Le résultat est acquis en posant $u = x = y$ et $i = j = 1$.
$\diamond$ Soit $m \geqslant 2$ tel que la propriété soit vraie pour tout couple (x, y) de mots tels que $|xy| \leqslant m$. Soit $x, y \in \Sigma^+$ tels que $xy = yx$ et $|xy| = m + 1$: en appliquant le théorème précédent, il existe deux mots v, w et $k \in \mathbb{N}$ tels que $x = vw = wv$ et $y = (vw)^k v = v(wv)^k$.

- Si $v = \varepsilon$ alors $y = x^k$ et on a le résultat souhaité avec $u = x$, $i = 1$ et $j = k$.
- Si $w = \varepsilon$ alors $y = x^{k+1}$ et on a le résultat souhaité avec $u = x$, $i = 1$ et $j = k + 1$
- Sinon, $|vw| = |x| < |xy|$ (car $y \neq \varepsilon$) donc on peut appliquer l'hypothèse de récurrence : il existe un mot u et deux entiers i et j tels que $v = u^i$ et $w = u^j$ et alors $x = u^{i+j}$ et $y = u^{(k+1)i+kj}$ ♯

2.3 Algorithmes naïfs de reconnaissance d'un motif

But : On veut écrire une fonction déterminant le nombre d'occurences d'un motif donné dans un texte.

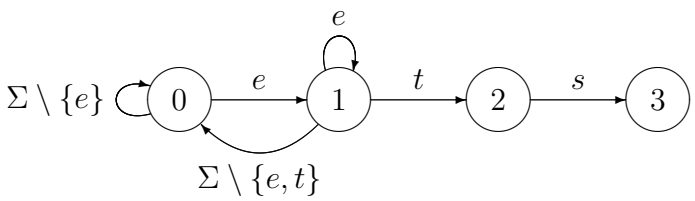
On se donne deux chaînes de caractères, l'une représentant le texte à analyser et l'autre le mot chercher et on veut que la fonction retourne la liste des positions dans le texte à partir desquelles on trouve le mot donné. La première idée consiste à faire glisser sur le texte une fenêtre de longueur égale à celle du mot à chercher et à comparer les lettres de la fenêtre avec celles du mot : pour cela, on utilise la fonction `String.sub`

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

Exemple :

```
# let m = "ets" ;;
val m : string = "ets"
# let t = "il met son pull car il ets frileux et ne veut pas avoir froid
pendant les derniers sets";;
val t : string = "il met son pull car ... sets"
# recherche m t;;
- : int list = [84; 23]
```

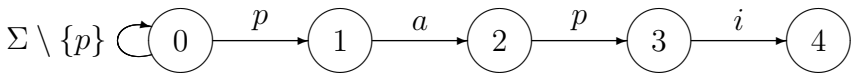
Il existe une autre méthode qui consiste à lire le texte lettre par lettre et dans laquelle on garde en mémoire le nombre de lettres du motif auquel on est parvenu. On peut pour l'exemple précédent représenter cette méthode à l'aide d'un automate dont les états (entourés) indiquent le nombre de lettres du motif auquel on est parvenu et les flèches indiquent les changements d'état. Compléter le schéma ci dessous :



Avec cette idée la fonction de recherche d'un motif dans un texte s'écrit :

```
let recherche motif texte =
    let resultat = ref [] in
    let n = String.length motif and p = String.length texte in
    for i = 0 to p - n do
        let j = ref 0 in
        while !j < n && motif.[!j] = texte.[i + !j] do
            incr j
        done;
        if !j = n then resultat := i::(!resultat)
    done;
    !resultat;;
```

Remarque : L'automate représentant la lecture d'un motif peut être plus compliqué. Compléter par exemple celui correspondant à la recherche du motif "papi" dans un texte.



Nous allons, dans la suite de ce chapitre, définir des ensembles de mots dont nous verrons dans le chapitre suivant qu'ils peuvent être reconnus par des automates.

3 Langages

3.1 Définition et opérations sur les langages

Définition 4. *On appelle* langage *sur un alphabet* Σ *toute partie de* Σ^*.

Exemples : Un langage peut être écrit en énumérant ses éléments ou à l'aide d'une propriété caractérisant ses éléments. Par exemple sur l'alphabet $\Sigma = \{a, b\}$,

- $L_1 = \{a, ab, aba, abab\}$ peut également être décrit comme l'ensemble des préfixes non vides du mot $m = abab$.
- $L_2 = \{a^n b^n;\ n \in \mathbb{N}\}$ peut être décrit comme l'ensemble des mots commençant par un certain nombre de a suivis du même nombre de b.

— $L_3 = \{mba;\ m \in \Sigma^*\}$ est l'ensemble des mots se terminant par ab.

On peut également décrire un langage par des règles de construction, qu'on appelle dans ce cas une *grammaire formelle*. Par exemple le langage L sur $\Sigma = \{a, b\}$ défini par :

$$\varepsilon \in L \text{ et } (r, s) \in L^2 \Rightarrow arbs \in L$$

est appelé langage des mots de DYCK.

On peut montrer par exemple que tout mot de DYCK est de longueur paire par récurrence sur la longueur n de ce mot.

En effet, la propriété est vraie à l'ordre 0 car 0 est pair. Soit $n \in \mathbb{N}$ tel que tout mot de DYCK de longueur $\leqslant n$ soit de longueur paire et soit m un mot de DYCK de longueur $n+1$. Alors il existe deux mots de DYCK r et s tels que $m = arbs$ et $|m| = 2+|r|+|s|$ est bien paire puisque r et s sont de longueur paire d'après l'hypothèse de récurrence.

L'ensemble des mots de DYCK de longueur inférieure ou égale à 6 est :

Opérations sur les langages

Comme les langages sont des ensembles, toutes les opérations ensemblistes leur sont applicables : réunion (parfois notée + dans ce contexte), intersection, passage au complémentaire dans Σ^*. D'autres opérations sont toutefois plus intéressantes, comme la concaténation de deux langages.

Définition 5. *Soit L et L' deux langages sur l'alphabet Σ*

— *La* concaténation *de L et L' est le langage, noté LL' défini par*

$$LL' = \{uv,\ u \in L \text{ et } v \in L'\}$$

— *On définit, par récurrence, la suite $(L^n)_{n\in\mathbb{N}}$ de langages par*
- $L^0 = \{\varepsilon\}$
- $\forall\, n \in \mathbb{N},\ L^{n+1} = L^n L = L^n L$

— *L'*étoile (ou fermeture de KLEENE) *du langage L est le langage L^* défini par $L^* = \bigcup_{n\in\mathbb{N}} L^n$ et on note $L^+ = \bigcup_{n\in\mathbb{N}^*} L^n$.*

Remarques : – Attention à ne pas confondre L^2 avec le langage plus petit constitué des carrés des mots de L : $\{u^2,\ u \in L\}$. Par exemple si $L = \{ab, ba\}$, $L^2 = \{abab, baba, abba, baab\}$ tandis que $\{u^2,\ u \in L\} = \{abab, baba\}$. Même remarque pour L^n.

– L^* est l'ensemble des mots obtenus en concaténant un nombre fini (éventuellement réduit à 0) de mots de L.

– on a $L^+ = LL^*$.

– Contrairement à L^* qui contient toujours le mot vide, L^+ ne contient le mot vide que si L le contient.

Donnons encore deux opérations sur les langages.

Définition 6. *Soit K, L deux langages sur un alphabet Σ.*

— *On appelle* racine carrée *du langage L, notée $\sqrt{L}$ le langage défini par*

$$\sqrt{L} = \{u \in \Sigma^* \,|\, u^2 \in L\}$$

— *On appelle* quotient (gauche) *des langages K et L, noté $K^{-1}L$ le langage*

$$K^{-1}L = \{v \in \Sigma^* \,|\, \exists u \in K \text{ tel que } uv \in L\}$$

Exemple : Si $L = \{a^n b^n\, n \in \mathbb{N}\}$ et $K = \{am,\ m \in \Sigma^*\}$, déterminer $\sqrt{L}$ et $K^{-1}L$

3.2 Expressions et langages rationnels

On va s'intéresser ici à des langages qui peuvent être décrits à l'aide d'une formule, appelée expression rationnelle, et on verra dans les chapitre suivant que ces langages sont très directement liés aux automates.

Définition 7. *Soit Σ un alphabet. Les expressions rationnelles (ou regulières) sur Σ sont définies inductivement par les propriétés suivantes :*
- *$\emptyset$ et ε sont des expressions rationnelles,*
- *$\forall\, a \in \Sigma$, a est une expression rationnelle,*
- *Si e_1 et e_2 sont des expressions rationnelles, alors $e_1 + e_2$, $e_1 e_2$ et e_1^* sont des expressions rationnelles.*

Remarque : $e_1 + e_2$ est parfois notée $e_1|e_2$ et e^* est également notée $e*$.

Expliquons maintenant comment on associe à toute expression rationnelle un langage.

Définition 8. L'interprétation *d'une expression rationnelle est définie par les règles inductives suivantes :*
- *$\emptyset$ dénote le langage vide et ε dénote le langage $\{\varepsilon\}$,*
- *$\forall\, a \in \Sigma$, a dénote le langage $\{a\}$,*
- *$e_1 + e_2$ dénote la réunion des langages dénotés par e_1 et e_2,*
- *$e_1 e_2$ dénote la concaténation des langages dénotés par e_1 et e_2*
- *e_1^* dénote l'étoile de* Kleene *du langage dénoté par e_1.*

Si e est une expression rationnelle sur l'alphabet Σ, on note $L[e]$ le langage dénoté par e.

Exemples : Sur l'alphabet $\Sigma = \{a, b\}$,
– a^*b^* dénote le langage des mots commençant par un certain nombre (éventuellement nul) de a suivi d'un certain nombre (éventuellement nul) de b.
– $(ab)^*$ dénote le langage constitué du mot vide et des mots commençant par a, alternant les a et les b et terminant par b. Notons que $\varepsilon + a(ba)^*b$ dénote le même langage.
– $(a+b)^*aa(a+b)^*$ dénote le langage des mots dont aa est un facteur.
– $(a+ba)^*$ dénote le langage des mots dans lesquels chaque b et suivi obligatoirement d'un a.
– $(a^*b)^*$ dénote le langage formé du mot vide et des mots se terminant par b. Notons que $\varepsilon + (a+b)^*b$ dénote le même langage.

Définition 9.
Un langage est dit rationnel *s'il est dénoté par une expression rationnelle.*
L'ensemble des langages rationnels sur l'alphabet Σ est noté $\mathrm{Rat}(\Sigma)$

Exemples : – Σ est rationnel car il dénoté par l'expression rationnelle $\sum_{a\in\Sigma} a$.
– Plus généralement, tout langage fini est rationnel.
– Comme Σ est rationnel, Σ^* est rationnel ainsi que $\Sigma^+ = \Sigma\Sigma^*$.
– L'ensemble des mots de longueur impaire est rationnel car il est égal à $\Sigma(\Sigma^2)^*$

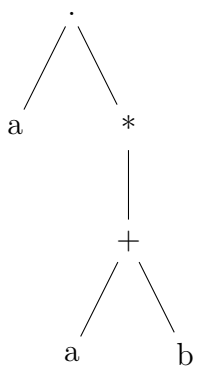
On a clairement la propriété suivante :

Théorème : L'ensemble des langages rationnels sur Σ est la plus petite partie de $\mathscr{P}(\Sigma^*)$ (au sens de l'inclusion) contenant $\emptyset$, $\{\varepsilon\}$, $\{a\}$ pour tout $a \in \Sigma$ et qui est stable par réunion, concaténation et passage à l'étoile de Kleene.

Arbre binaire associé à une expression rationnelle
On peut naturellement associer à toute expression rationnelle un arbre binaire où les feuilles sont éléments de $\{\emptyset, \varepsilon\} \cup \Sigma$ et les nœuds les opérateurs $\{+, ., *\}$. Ceci permet

de prouver certains propriétés relatives aux expressions rationnelles par induction sur la hauteur de l'arbre en question, appelée également *profondeur* de l'expression rationnelle considérée.

Par exemple l'arbre associé à l'expression rationnelle $a(a+b)^*$ est



Expressions équivalentes
La correspondance entre expressions et langages n'est pas bijective : chaque expression dénote un unique langage, mais un langage donné peut être dénoté par plusieurs expressions différentes, comme on l'a remarqué sur certains exemples.

Définition 10. *Deux expressions rationnelles sont dites* équivalentes *si elles dénotent le même langage.*

Remarques : – Déterminer si deux expressions rationnelles sont équivalentes ou chercher pour un langage rationnel donné l'expression rationnelle la plus courte le dénotant, sont des problèmes difficiles à résoudre de façon algorithmique. L'approche sans doute la plus efficace consiste à passer par les automates, comme on le verra dans la chapitre suivant.
– Nous allons déjà apporter une simplification, en montrant que tout langage rationnel non vide peut être dénoté par une expression ne comportant pas le symbole $\emptyset$, puis qu'on peut s'arranger pour que ε intervienne au plus une fois dans l'expression dénotant le langage.

Proposition Si le langage dénoté par l'expression rationnelle e est non vide, il existe une expression rationnelle e' équivalente à e ne contenant pas le symbole $\emptyset$.

Démonstration : Notons L le langage, supposé non vide, dénoté par l'expression e. Montrons le résultat souhaité par induction sur e.

- Si $e = \varepsilon$ ou $e \in \Sigma$, on peut poser $e' = e$.
- Si $e = e_1 + e_2$, alors $L = L_1 \cup L_2$, où L_i est le langage dénoté par e_i. Trois cas sont alors possibles
  - Si $L_1 = \emptyset$, $L = L_2 \neq \emptyset$ et par hypothèse d'induction, il existe une expression régulière e'_2 sans symbole $\emptyset$ dénotant $L_2 = L$ donc telle que $e'_2 \equiv e$.
  - Le cas où $L_2 = \emptyset$ se traite de manière analogue
  - Sinon, L_1 et L_2 sont non vides donc, par hypothèse d'induction, il existe deux expressions rationnelles e'_1 et e'_2 ne comportant pas le symbole $\emptyset$ telles que $e'_1 \equiv e_1$ et $e'_2 \equiv e_2$ et alors $e' = e'_1 + e'_2$ est une expression sans $\emptyset$ équivalente à e.
- Si $e = e_1 e_2$ aucune des deux expressions rationnelles e_1 et e_2 ne peut dénoter le langage vide (car la concaténation de deux langages dont l'un est l'ensemble vide est l'ensemble vide), donc par hypothèse d'induction, il existe $e'_1 \equiv e_1$ et $e'_2 \equiv e_2$ telles que e'_1 et e'_2 ne comportent pas le symbole $\emptyset$ et alors $e' = e'_1 e'_2$ est une expression équivalente à e ne comportant pas le symbole $\emptyset$.
- Si $e = e_1^*$ alors ou bien e_1 dénote le langage vide auquel cas $e \equiv \varepsilon$ ou bien par hypothèse d'induction, il existe une expression e'_1 ne comportant pas le symbole $\emptyset$ telle que $e'_1 \equiv e_1$ et alors $e \equiv (e'_1)^*$ expression rationnelle ne comportant pas le symbole $\emptyset$ ♯

Proposition Si le langage dénoté par l'expression rationnelle e est non vide, il existe une expression rationnelle e' ne contenant ni le symbole $\emptyset$, ni le symbole ε telle que e soit équivalente à ε, e' ou $\varepsilon + e'$.

Démonstration : D'après la proposition précédente, on peut supposer que e ne comporte pas le symbole $\emptyset$. On raisonne par induction sur e.

— Si $e = \varepsilon$ ou $e \in \Sigma$, on a le résultat souhaité.

— Si $e = e_1 + e_2$, on applique l'hypothèse d'induction à e_1 et e_2 : il existe e'_1 et e'_2 expressions régulières sans $\emptyset$ ni ε telles que e_i soit équivalente à ε, e'_i ou $e'_1 + \varepsilon$. Cela nous donne neuf cas possibles résumés dans le tableau suivant où on a indiqué dans chaque cas une expression équivalente à e du type souhaité :

$+$	ε	e'_2	$\varepsilon + e'_2$
ε	ε	$\varepsilon + e'_2$	$\varepsilon + e'_2$
e'_1	$\varepsilon + e'_1$	$e'_1 + e'_2$	$\varepsilon + (e'_1 + e'_2)$
$\varepsilon + e'_1$	$\varepsilon + e'_1$	$\varepsilon + (e'_1 + e'_2)$	$\varepsilon + (e'_1 + e'_2)$

— Si $e = e_1 e_2$, on a de même :

$\cdot$	ε	e'_2	$\varepsilon + e'_2$
ε	ε	e'_2	$\varepsilon + e'_2$
e'_1	e'_1	$e'_1 e'_2$	$e'_1 + e'_1 e'_2$
$\varepsilon + e'_1$	$\varepsilon + e'_1$	$e'_2 + e'_1 e'_2$	$\varepsilon + (e'_1 + e'_2 + e'_1 e'_2)$

— Si $e = e_1^*$ et qu'on note e'_1 une expression sans $\emptyset$ ni ε telle que e_1 soit équivalente à ε, e'_1 ou $\varepsilon + e'_1$ on a selon les cas e qui est équivalente à ε, ${e'_1}^*$ ou à $\varepsilon + {e'_1}^*$ ♯

Implémentation en OCaml des expressions rationnelles

On peut représenter en machine les expressions régulières sans symbole $\emptyset$ avec le type somme suivant :

```
type regexp =
    | Epsilon
    | Const of string
    | Somme of regexp * regexp
    | Concat of regexp * regexp
    | Etoile of regexp ;;
```

Par exemple l'expression rationnelle $a(a+b)^*$ sera représentée avec ce type par :

```
let e =
```

3.3 Langages locaux

Nous étudions dans ce paragraphe des langages caractérisés par une lecture locale des lettres (d'où leur nom), le principe étant de ne lire les mots (et de tester leurs lettres) qu'avec une fenêtre de lecture de longueur 2.

Définition 11. *Un langage L sur l'alphabet Σ est dit* local *s'il existe $I \subset \Sigma$, $F \subset \Sigma$ et $B \subset \Sigma^2$ tels que*
pour tout mot $m \in \Sigma^* \setminus \{\varepsilon\}$ s'écrivant $m = u_1 u_2 \dots u_n$, on a l'équivalence :

$$m \in L \iff \big(u_1 \in I, u_n \in F, \text{ et } \forall\, i \in [\![1, n-1]\!],\, u_i u_{i+1} \in B\big)$$

Remarque : On constate qu'un langage local peut contenir ou non le mot vide.
Notation : Si L est un langage local décrit par les ensembles I, F et B de définition précédente, on notera

$$L = L_{I,B,F,bo}$$

où bo est le booléen égal à `true` si ε appartient à L et à `false` sinon.

Exemples : – $\emptyset$ et $\{\varepsilon\}$ sont des langages locaux, définis par $I = B = F = \emptyset$,
– $L = \{a\}$ est local, défini par $I = F = \{a\}$, $B = \emptyset$ et $\varepsilon \notin L$,
– $L = \{(ab)^n\,;\, n \in \mathbb{N}\}$ est local, défini par $I = \{a\}$, $B = \{ab, ba\}$, $F = \{b\}$ et $\varepsilon \in L$,
– $L' = \{(ab)^n\,;\, n \in \mathbb{N}^*\}$ est également local, défini par les mêmes ensembles I, B, F mais avec cette fois $\varepsilon \notin L'$.

Remarque : Si $L = L_{I,B,F,b}$ est local on peut affirmer que :
– l'ensemble des préfixes de taille 1 des mots de L est inclus dans I
– l'ensemble des suffixes de taille 1 des mots de L est inclus dans F
– l'ensemble des facteurs de taille 2 des mots de L est inclus dans B
mais ces inclusions ne sont pas forcément des égalités car, par exemple si $\Sigma = \{a, b\}$, et $L = \{a^n\,;\, n \in \mathbb{N}\}$, L est local, défini par $I = F = \{a\}$, $B = \{aa\}$ et $\varepsilon \in L$ mais il est également défini par $I = \{a, b\}$, $B = \{aa, ab\}$, $F = \{a\}$ et $\varepsilon \in L$.

Exercice : Pour les langages suivants sur l'alphabet $\Sigma = \{a, b\}$ indiquer s'ils sont locaux ou non, en précisant lorsqu'ils sont locaux des ensembles I, B, F et un booléen b tels que $L = L_{I,B,F,b}$.

– $L = \{a^n b^p\,;\, (n, p) \in \mathbb{N}^2\}$

– $L' = \{a^n b^n\,;\, n \in \mathbb{N}\}$

Remarque : En introduisant $N = \Sigma^2 \setminus B$, on peut donner une définition plus synthétique des langages locaux

Proposition : Un langage L sur l'alphabet Σ est local si et seulement s'il existe $I \subset \Sigma$, $F \subset \Sigma$ et $N \subset \Sigma^2$ tels que :

$$L \setminus \{\varepsilon\} = \big(I.\Sigma^* \cap \Sigma^*.F\big) \setminus \big(\Sigma^*.N.\Sigma^*\big)$$

Démonstration : • Si $L \setminus \{\varepsilon\} = \big(I.\Sigma^* \cap \Sigma^*.F\big) \setminus \big(\Sigma^*.N.\Sigma^*\big)$ et qu'on pose $B = \Sigma^2 \setminus N$, les mots de L différents du mot vide sont les mots commençant par une lettre de I, terminant par une lettre de F et dont aucun facteur de taille deux n'appartient à N c'est-à-dire dont tous les facteurs de taille deux appartiennent à B, donc L est local associé à I, F et B.
• Si $L = L_{I,B,F,bo}$ est local et que l'on pose $N = \Sigma^2 \setminus B$, les mots de $L \setminus \{\varepsilon\}$ sont les mots commençant par une lettre de I, terminant par une lettre de F et dont aucun facteur de taille deux n'appartient à N donc $L \setminus \{\varepsilon\} = \big(I.\Sigma^* \cap \Sigma^*.F\big) \setminus \big(\Sigma^*.N.\Sigma^*\big)$ ♯

Remarque : Cette caractérisation montre qu'un langage local peut être construit à partir des trois langages rationnels $I.\Sigma^*$, $\Sigma^*.F$ et $\Sigma^*.N.\Sigma^*$ mais en utilisant des opérations non rationnelles : l'intersection et la différence. Il faudra donc attendre le chapitre suivant avant de pouvoir montrer que tout langage local est rationnel.
La réciproque n'est pas vraie comme le prouve le langage $L = \{ab\} \cup \{a^n\,;\, n \in \mathbb{N}\}$ qui est rationnel mais n'est pas local.

Opérations sur les langages locaux

Proposition L'intersection de deux langages locaux est un langage local

Démonstration : En effet si $L_1 = L_{I_1,B_1,F_1,b1}$ et $L_2 = L_{I_2,B_2,F_2,b2}$ sont deux langages locaux, on a clairement : $L_1 \cap L_2 = L_{I_1 \cap I_2, B_1 \cap B_2, F_1 \cap F_2, b_1 \&\& b_2}$ ♯

Remarque : La réunion ou la concaténation de deux langages locaux n'est pas forcément un langage local comme on le constate avec $L_1 = \{ab\}$ et $L_2 = \{a^n\,;\, n \in \mathbb{N}\}$. On a toutefois les résultats suivants :

Proposition : Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1 \cup L_2$ est encore un langage local

Démonstration : Notons Σ_1 et Σ_2 les deux alphabets sur lesquels sont respectivement

définis L_1 et L_2 avec $\Sigma_1 \cap \Sigma_2 = \emptyset$ et posons $\Sigma = \Sigma_1 \cup \Sigma_2$.
Par hypothèse L_1 et L_2 sont locaux donc s'écrivent $L_1 = L_{I_1,B_1,F_1,b_1}$ et $L_2 = L_{I_2,B_2,F_2,b_2}$ avec pour $i \in \{1,2\}$, $I_i \in \Sigma_i$, $B_i \in \Sigma_i^2$, $F_i \in \Sigma_i$.
On a clairement $L_1 \cup L_2 \subset L_{I_1 \cup I_2, B_1 \cup B_2, F_1 \cup F_2, b_1||b_2}$.
Montrons l'inclusion réciproque : soit $u \in L_{I_1 \cup I_2, B_1 \cup B_2, F_1 \cup F_2, b_1||b_2}$.
Si $u = \varepsilon$, le booléen $b_1||b_2$ est vrai donc b_1 ou b_2 est vrai donc ε appartient à L_1 ou à L_2 donc $u = \varepsilon \in L_1 \cup L_2$.
Sinon, u s'écrit $u = u_1 u_2 \dots u_p$.
On a $u_1 \in I_1 \cup I_2$: sans perte de généralité, on peut supposer que $u_1 \in I_1 \subset \Sigma_1$
Si $p \geqslant 2$, on a alors $u_1 u_2 \in B_1 \cup B_2$ mais comme $u_1 \in \Sigma_1$ et que $\Sigma_1 \cap \Sigma_2 = \emptyset$ et que $B_2 \subset \Sigma_2^2$, on a nécessairement $u_1 u_2 \in B_1$ donc $u_2 \in \Sigma_1$.
De proche en proche, on montre que pour tout $i \in [\![1, p-1]\!]$, $u_i u_{i+1} \in B_1$.
En particulier $u_p \in \Sigma_1$ et comme, $u_p \in F_1 \cup F_2$, on a nécessairement $u_p \in F_1$. On a donc $u \in L_1$ donc $u \in L_1 \cup L_2$ ce qui achève de montrer que $L_{I_1 \cup I_2, B_1 \cup B_2, F_1 \cup F_2, b_1||b_2} \subset L_1 \cup L_2$.
Donc $L_1 \cup L_2 = L_{I_1 \cup I_2, B_1 \cup B_2, F_1 \cup F_2, b_1||b_2}$ est local ♯

Proposition : Si L_1 et L_2 sont deux langages locaux définis sur des alphabets disjoints, alors $L_1.L_2$ est encore un langage local

Démonstration : On prend les mêmes notations que dans la démonstration de la proposition précédente.
Remarquons que si a est la première lettre d'un mot de $L_1.L_2$ alors a est la première lettre d'un mot de L_1 ou, dans la mesure où $\varepsilon \in L_1$, a est la première lettre d'un mot de L_2. Cette remarque nous conduit à poser $I = \begin{cases} I_1, & \text{si } \varepsilon \notin L_1 \,; \\ I_1 \cup I_2, & \text{sinon.} \end{cases}$
En raisonnant de même, on pose $F = \begin{cases} F_2, & \text{si } \varepsilon \notin L_2 \,; \\ F_1 \cup F_2, & \text{sinon.} \end{cases}$ et $B = B_1 \cup F_1.I_2 \cup B_2$.
Par ailleurs, le mot vide appartient à $L_1.L_2$ si et seulement si il appartient à L_1 et à L_2 ce qui nous conduit à poser $b = b_1 \&\& b_2$.
On a alors $L_1.L_2 \subset L_{I,B,F,b}$.
Montrons l'inclusion réciproque : soit $u \in L_{I,B,F,b}$.
Si $u = \varepsilon$, alors b est vrai donc b_1 et b_2 sont vrais donc $\varepsilon \in L_1 \cap L_2$ donc $\varepsilon = \varepsilon.\varepsilon \in L_1.L_2$.
Sinon, u s'écrit $u_1 u_2 \dots u_p$ avec les $u_i \in \Sigma \in \Sigma_1 \cup \Sigma_2$.

- Si $u_1 \in \Sigma_2$, alors $\varepsilon \in L_1$ et $u_1 \in I_2$. De proche en proche, on montre que les $u_i u_{i+1}$ sont tous dans B_2. En particulier, $a_p \in \Sigma_2$ donc $a_p \in F_2$. On a donc dans ce cas $u \in L_2$ et, puisque $\varepsilon \in L_1$, $u \in L_1.L_2$.
- Sinon, $u_1 \in \Sigma_1$. Notons alors $u_1 \dots u_k$ le plus grand préfixe de u qui soit dans Σ_1^*. Alors $u_1 \in I_1$ et pour tout $i \in [\![1, k-1]\!]$, $u_i u_{i+1} \in B_1$. Ensuite, de deux choses l'une :
  - Si $k = p$, alors $u_p \in F_1$ et $\varepsilon \in L_2$. On a alors $u \in L_1$ et puisque $\varepsilon \in L_2$, $u \in L_1.L_2$.
  - Si $k < p$ alors $u_k u_{k+1} \in \Sigma_1 \times \Sigma_2 \cap B = F_1.I_2$ donc $u_{k+1} \in I_2$ et on montre de proche en proche que pour $i \in [\![k+1, p-1]\!]$, $u_i u_{i+1} \in B_2$. En particulier $u_p \in \Sigma_2 \cap F = F_2$. On a alors $u_1 \dots u_k \in L_1$ et $u_{k+1} \dots u_p \in L_2$ donc $u \in L_1.L_2$ ♯

Proposition Si L est un langage local, L^* est un langage local.

Démonstration : Par hypothèse L est de la forme $L = L_{I,B,F,bo}$. Posons $I' = I$, $F' = F$, $B' = B \cup F.I$, $b' = true$ et montrons que $L^* = L_{I',B',F',b'}$.
Le mot vide ε appartient à L^* et à $L_{I',B',F',b'}$ (puisque $b' = true$).
Si $u \in L^* \setminus \{\varepsilon\}$, u s'écrit $u_1 u_2 \dots u_k$ avec tous les u_i dans L.
La première lettre de u est celle de u_1 et appartient donc à $I = I'$.

La dernière lettre de u est celle de u_k et appartient à $F = F'$.
Les facteurs de longueur 2 de u sont soit des facteurs de l'un des u_i soit un mot de deux lettres constitué de la dernière lettre d'un u_i et de la première lettre de u_{i+1} donc un mot de $F.I \setminus B'$.
On a donc $u \in L_{I',B',F',b'}$. Par conséquent $L^* \subset L_{I',B',F',b'}$.
Réciproquement si $u \in L_{I',B',F',b'} \setminus \{\varepsilon\}$.
Parmi les facteurs de longueur 2 de u considérons ceux appartenant à $F.I$: ils permettent de factoriser u en $u_1 u_2 \dots u_k$ avec pour tout i, $u_i \in L_{I,B,F,b} \setminus \{\varepsilon\} \subset L$ et donc $u \in L^*$. ♯

Expressions rationnelles linéaires

Une expression rationnelle générale ne dénote pas forcément un langage local : par exemple, le langage dénoté par $ab + a^*$ n'était pas local. Il existe toutefois des expressions rationnelles particulières pour lesquelles le langage dénoté est local : les expressions linéaires.

Définition 12. *Une expression rationnelle e sur l'alphabet Σ est dite* linéaire *si tout caractère de Σ apparaît au plus une fois dans e.*

Exemples : sur $\Sigma = \{a, b, c\}$, $ab + a^*$ n'est pas linéaire mais $ab + c^*$ l'est.
L'intérêt de cette notion découle de la proposition suivante

Proposition : Tout langage dénoté par une expression rationnelle linéaire est local.

Démonstration : Par induction structurelle sur l'expression rationnelle linéaire e.

- Si $e = \emptyset$ (resp. $e = \varepsilon$), $L[e] = \emptyset$ (resp. $L[e] = \{\varepsilon\}$) donc $L[e]$ est local (prendre $I = B = F = \emptyset$ et $b =$`false` (resp. $b =$`true`))
- Si $e = a \in \Sigma$, $L[e] = \{a\}$ qui est local (prendre $I = F = \{a\}$, $B = \emptyset$ et $b =$`false` donc $L[e]$).
- Si $e = e_1 + e_2$ (resp. $e = e_1 e_2$) est une expression rationnelle linéaire, e_1 et e_2 sont des expressions rationnelles régulières et on peut considérer qu'elles sont définies sur des alphabets disjoints : par hypothèse d'induction, $L[e_1]$ et $L[e_2]$ sont des langages locaux, et comme ils sont définis sur des alphabets disjoints $L[e] = L[e_1] \cup L[e_2]$ (resp. $L[e] = L[e_1]L[e_2]$) est un langage local.
- Si $e = e_1^*$ est une expression rationnelle linéaire, e_1 est linéaire donc par hypothèse d'induction $L[e_1]$ est local donc $L[e] = L[e_1]^*$ est également local. ♯

Remarque : La réciproque du résultat précédent est fausse : par exemple $L = \{a^n\ ,\ n \in \mathbb{N}^*\}$ est local (prendre $I = F = \{a\}$, $B = \{aa\}$ et $b =$`false`), il est rationnel car dénoté par l'expression aa^* mais il n'existe pas d'expression rationnelle linéaire le dénotant.

Algorithmes de détermination des ensembles I, B et F

Les différents résultats relatifs aux propriétés de stabilité des langages locaux nous permettent d'écrire des fonctions déterminant les ensembles I, B et F correspondant au langage local dénoté par une expression rationnelle linéaire, en procédant par induction structurelle.

Écrivons tout d'abord une fonction déterminant si le mot vide appartient au langage dénoté par une expression régulière :

```
let rec motvide e = match e with
    | Epsilon        -> true
    | Const _        -> false
    | Somme (e1,e2)  -> motvide e1 || motvide e2
    | Concat (e1,e2) -> motvide e1 && motvide e2
    | Etoile _       -> true;;
```

Nous aurons également besoin d'une fonction qui étant donné deux listes (chacune supposée sans doublon) retourne une liste sans doublon dont l'ensemble des éléments est la réunion des ensembles des éléments des deux listes.

```
let rec union l1 l2 = match l1 with
    | []                        -> l2
    | t :: q when List.mem t l2 -> union q l2
    | t :: q                    -> t :: (union q l2);;
```

Nous pouvons maintenant écrire les fonctions donnant les ensembles I et F (sous forme de listes)

```
let rec i e = match e with
    | Epsilon                         -> []
    | Const a                         -> [a]
    | Somme (e1, e2)                  -> union (i e1) (i e2)
    | Concat (e1,e2) when motvide e1 -> union (i e1) (i e2)
    | Concat (e1,e2)                  -> i e1
    | Etoile e1                       -> i e1;;

let rec f e = match e with
    | Epsilon                         -> []
    | Const a                         -> [a]
    | Somme (e1, e2)                  -> union (f e1) (f e2)
    | Concat (e1,e2) when motvide e2 -> union (f e1) (f e2)
    | Concat (e1,e2)                  -> f e2
    | Etoile e1                       -> f e1;;
```

Pour déterminer l'ensemble des facteurs d'ordre deux nous avons besoin d'une fonction déterminant pour deux parties A et B de Σ, l'ensemble AB.

```
let rec produit l1 l2 = match (l1,l2) with
    | [] , _    -> []
    | _ , []    -> []
    | t1::q1, _ -> union (List.map (function x -> t1^x) l2) (produit q1 l2);;
```

On en déduit la fonction calculant l'ensemble B de facteurs de longueur 2 de $L[e]$.

```
let rec b e = match e with
    | Epsilon        -> []
    | Const a        -> []
    | Somme (e1,e2)  -> union (b e1) (b e2)
    | Concat (e1,e2) -> let l = union (b e1) (b e2) in
                        union l (produit (f e1) (i e2))
    | Etoile e1      -> union (b e1) (produit (f e1) (i e1));;
```