Chapitre 14 : Langages formels

Table des matières

1 Langages réguliers

1.1 Motivation

1.1.1 introduction

On a souvent besoin de mettre en place une analyse de texte, même dans le cadre d'applications qui ne relèvent pas uniquement du traitement de texte.

Par exemple :

- La recherche d'un mot dans un texte (*cf* chap 11) ;
- analyser un document structuré afin de traiter de manière appropriée son contenu (exemple : compiler un programme, récupérer des données sérialisées (*cf* chap 11) dans un format particulier (ex : données brutes en CSV, fichiers de configuration en JSON ou en XML)) ;
- Reconnaître un encodage et le déchiffrer (exemple : QR-code).

Quelle que soit l'application, on a besoin d'un formalisme pour décrire la structure du texte et d'algorithmes efficaces capables d'analyser cette structure et d'extraire les données associées.

1.1.2 Exemple

Étant donné un fichier binaire, déterminer s'il contient la représentation binaire d'un entier non signé multiple de 3.

- Remarque : on n'utilise pas les types d'entiers natifs de C ou OCaml car ils ont une taille fixée qui peut être dépassée par le fichier.

Idée : on lit les bits un à un en effectuant les opérations associées modulo 3, en remarquant que

$$\begin{cases} \langle x0 \rangle_2 = 2 \langle x \rangle_2 \\ \langle x1 \rangle_2 = 2 \langle x \rangle_2 + 1 \end{cases}$$

On utilise cette table :

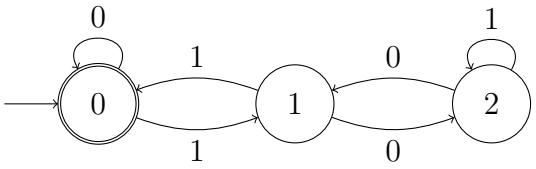
$\langle x \rangle_2 \mod 3$	0	1	2
$\langle x0 \rangle_2 \mod 3$	0	2	1
$\langle x1 \rangle_2 \mod 3$	1	0	2

- Algorithme :

Algorithm 1:

```
1 x ← 0;
2 for chaque bit b pris dans l'odre do
3     if b = 0 then
4         x ← 2x  mod 3;
5     else
6         x ← 2x + 1  mod 3
7 return x = 0;
```

• Représentation graphique : x ne peut prendre que trois valeurs différentes, appelées états, et on peut représenter les changements de valeur de x dans un graphe orienté dont les sommets sont les états, et les arcs sont étiquetés par le bit qui produit le changement de valeur de l'état source vers l'état cible.



On distingue de plus la valeur initiale par une flèche, et la valeur finale atteinte par un double cercle.

Cette représentation correspond au formalisme des *automates*, que nous verrons en **??** (page **??**).

1.2 Langages

1.2.1 Définition (*alphabet*)

Un *alphabet* est un ensemble fini non vide, dont les éléments sont appelés lettres ou symboles.

Notation usuelle : Σ.

1.2.2 Définition (*mot*)

Soit Σ un alphabet.

Un *mot* sur Σ est une suite finie de symboles $u = u_1 \cdots u_n$, potentiellement vide.

Si $n = 0$, on note $u = \varepsilon$.

On note $|u| = n$ la *longueur* du mot.

1.2.3 Définition (*concaténation*)

Soit Σ un alphabet, et u, v deux mots sur Σ.

On appelle *concaténation* de u et v le mot

$$uv = \begin{cases} v & \text{si } u = \varepsilon \\ u & \text{si } v = \varepsilon \\ u_1 \cdots u_n v_1 \cdots v_p & \text{si } \begin{cases} u = u_1 \cdots u_n \\ v = v_1 \cdots v_p \end{cases} \end{cases}$$

Exo

• $|uv| = |u| + |v|$

• La concaténation est une loi de composition interne associative et d'élément neutre ε sur l'ensemble des mots sur Σ.

1.2.4 Définition (*préfixe, suffixe, facteur, sous-mot*)

Soit Σ un alphabet, et u, v deux mots sur Σ

• v est un *préfixe* de u ssi $\exists w$ mot tel que $u = vw$

• v est un *suffixe* de u ssi $\exists w$ mot tel que $u = wv$

• v est un *facteur* de u ssi $\exists x, y$ mots tels que $u = xvy$

• v est un *sous-mot* de u ssi $\exists i_1 < i_2 < \cdots < i_k$ tels que si $u = u_1 \cdots u_n$, alors $v = u_{i_1} \cdots u_{i_k}$

Exemple : si $u = abc$, $v = ac$ est un sous-mot de u, mais pas un facteur.

1.2.5 Langages

• Définition (*langage*) : un *langage* sur un alphabet Σ est un ensemble de mots sur Σ.

• Exemples :
  – l'ensemble de tous les mots, noté Σ^* (*cf* 1.3, page 5) ;
  – l'ensemble des écritures binaires de multiples des 3 ($\Sigma = \{0, 1\}$) ;
  – Σ (si on voit les lettres comme des mots de longueur 1) ;
  – l'ensemble des code sources OCaml de programmes qui ne terminent pas.

• Problème : étant donné un langage L sur un alphabet Σ, on veut disposer d'une représentation formelle de L pour pouvoir étudier la question suivante : étant donné un mot u, a-t-on $u \in L$?

C'est une question importante car souvent, comme dans les exemples en 1.1.1 (page 2), il faut pouvoir vérifier la structure d'un élément avant d'en extraire des données.

Malheureusement, on ne peut pas toujours répondre algorithmiquement à cette question. (*cf* chap 16 et la notion de décidabilité et le problème de l'arêt), mais on peut y répondre pour une classe restreinte de langages.

1.3 Langages réguliers

1.3.1 Opérations sur les langages

Outre les opérations ensemblistes usuelles (intersection, union, complémentaire), on définit certaines opérations plus spécifiques aux langages.

• **Concaténation** : Soit Σ un alphabet, et L, L' deux langages sur Σ.
La concaténation de L et L' est le langage

$$LL' = \{uv \mid (u,v) \in L \times L'\}$$

Remarque : $L\varnothing = \varnothing L = \varnothing$.

• **Puissance** : Soit Σ un alphabet, L un langage sur Σ, et $n \in \mathbb{N}$.
La puissance n-ème de L est le langage

$$L^n = \begin{cases} \{\varepsilon\} & \text{si } n = 0 \\ LL^{n-1} & \text{si } n > 0 \end{cases}$$

• **Étoile de Kleene** : Soit Σ un alphabet, et L un langage sur Σ.
L'*étoile de* Kleene de L est le langage

$$L^* = \bigcup_{n\in\mathbb{N}} L^n$$

• Remarques :
– Σ^* est bien l'ensemble de tous les mots : tout mot $u = u_1 \cdots u_n$ est la caractérisation de ses lettres ($\forall i \in [\![1\ ;\ n]\!]\,,\ u_i \in \Sigma$, donc $u = u_1 \cdots u_n \in \Sigma^n \subseteq \Sigma^*$).
– On peut aussi définir la puissance n-ème d'un mot :

$$u^n = \begin{cases} \varepsilon & \text{si } n = 0 \\ uu^{n-1} & \text{si } n > 0 \end{cases}$$

Attention : ne pas confondre L^n et $\{u^n \mid u \in L\}$.
– On note aussi

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

Exo $L^+ = L^* \ \Leftrightarrow\ \varepsilon \in L$.

1.3.2 Théorème (*Lemme d'*Arden) (H.P)

> Soit Σ un alphabet, et K, L deux langages sur Σ.

(1) K^*L est le minimum (pour l'ordre de l'inclusion) des solutions de l'équation

$$X = KX \cup L$$

d'inconnue un langage X.

(2) Si $\varepsilon \notin K$, alors K^*L est l'unique solution.

□

(1) On a :

$$\begin{aligned} K(K^*L) \cup L &= K\left(\bigcup_{n\in\mathbb{N}} K^n L\right) \cup L \\ &= K^+L \cup K^0 L \\ &= K^*L \end{aligned}$$

Donc K^*L est une solution.
Puis soit X une solution. Montrons que $K^*L \subset X$.
Soit $u \in K^*L$. Par définition,

$$\exists n \in \mathbb{N},\ \exists k_1, \dots, k_n \in K,\ \exists l \in L \mid u = k_1 \cdots k_n l$$

On montre par récurrence sur n que $u \in X$:
• $n = 0$: $u \in L \subset KX \cup L = X$, donc $u \in X$.

• Héréditée : si $\forall k_1 \cdots k_n \in K,\ \forall l \in L,\ k_1 \cdots k_n l \in X$, considérons $u = k_1 \cdots k_{n+1} l$, avec $k_1 \cdots k_{n+1} \in K$ et $l \in L$.
$u = k_1(\underbrace{k_2 \cdots k_{n+1} l}_{\in X \text{ par H.R}}) \in KX \subseteq KX \cup L = X$, donc $u \in X$.
Finalement, $K^*L \subset X$, et K^*L est bien le minimum des solutions.

(2) On suppose $\varepsilon \notin K$. Soit X une solution.
On sait par (1) que $K^*L \subseteq X$.
Il suffit de montrer que $X \subseteq K^*L$.
Soit $u \in X$ dont on note n la longueur.
On montre par récurrence que

$$\forall k \in [\![0\ ;\ n]\!],\ X = \bigcup_{j=0}^{k} K^j L \cup K^{k+1} X$$

• $k = 0$:

$$\bigcup_{j=0}^{0} K^j L \cup K^{0+1} X = L \cup KX = X$$

• Héréditée : Soit $k \in [\![0\ ;\ n-1]\!] \mid X = \bigcup_{j=0}^{k} K^j L \cup K^{k+1} X$

$$\begin{aligned} X &= KX \cup L \\ &= K\left(\bigcup_{j=0}^{k} K^j L \cup K^{k+1} L\right) \cup L \\ &= \bigcup_{j=0}^{k} K^{j+1} L \cup K^{k+1} L \cup L \\ &= \bigcup_{j=0}^{k+1} K^J L \cup K^{k+2} L \end{aligned}$$

En particulier,

$$X = \bigcup_{j=0}^{n} K^J L \cup K^{n+1} X$$

Or $\forall v \in k^{n+1}X,\ |v| \geqslant n+1$

En effet, $\exists k_1 \cdots k_{n+1} \in K,\ \exists x \in X \mid v = k_1 \cdots k_{n+1} x$.

Or $\forall i \in [\![1\ ;\ n+1]\!], k_i \neq \varepsilon$, donc $|k_i| \geqslant 1$, donc

$$|v| = \sum_{i=1}^{n+1} |k_i| + |x| \geqslant n+1+|x| \geqslant n+1$$

Donc, comme $|u| = n < n+1$,

$$\bigcup_{j=0}^{n} K^j L \subseteq \cup_{j\in\mathbb{N}} K^J L = K^* L$$

Donc $u \in K^*L$, et $X \subseteq K^*L$ ■

• Contre-exemple au point (2) si $\varepsilon \in K$: on prend $K = \{\varepsilon\}$, et $L = \{a\}$. $K^*L = \{a\} = L$, et tout $X \mid a \in X$ est solution ($KX \cup L = X \cup \{a\}$). Par exemple : $\{a\} = K^*L$ ou $\{a, aa\}, \{a, aa, aaa\}$.

1.3.3 Langage régulier

La classe des *langages réguliers*, aussi appelés *langages rationnels*, est la famille des langages que l'on peut construire à partir de langages de base ($\varnothing$, $\{\varepsilon\}$, $\{a\}$ $\forall a \in \Sigma$) et des opérations dites *régulières* : union, concaténation et étoile de KLEENE. L'ensemble $\mathsf{Reg}(\Sigma)$ des langages réguliers sur l'alphabet Σ est définit inductivement par :

$$\frac{}{\varnothing \in \mathsf{Reg}(\Sigma)} \qquad \frac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \qquad \frac{L \in \mathsf{Reg}(\Sigma)}{L^* \in \mathsf{Reg}(\Sigma)}$$

$$\frac{L \in \mathsf{Reg}(\Sigma)\ L' \in \mathsf{Reg}(\Sigma)}{L \cup L' \in \mathsf{Reg}(\Sigma)} \qquad \frac{L \in \mathsf{Reg}(\Sigma)\ L' \in \mathsf{Reg}(\Sigma)}{LL' \in \mathsf{Reg}(\Sigma)}$$

• Remarque : $\{\varepsilon\} \in \mathsf{Reg}(\Sigma)$ car $\{\varepsilon\} = \varnothing^0 = \bigcup_{n\in\mathbb{N}} \varnothing^n = \varnothing^*$

$\Sigma \in \mathsf{Reg}(\Sigma)$ car, en notant $\Sigma = \{a_1 \cdots a_n\}$, on a :

$$\Sigma = \bigcup_{k=1}^{n} \{a_k\}$$

Donc une récurrence sur $n = |\Sigma|$ conclut.
De même, tout langage fini est régulier (Exo).

Il manque encore un formalisme pour décrire les langages réguliers.

1.3.4 Expressions régulières

Soit Σ un alphabet, et $S = \{(,), |, *, \varnothing, \varepsilon\}$ un ensemble de symboles supposé disjoint de Σ.
L'ensemble $\mathsf{Regexp}(\Sigma)$ des *expressions régulières* sur Σ, aussi appelées *expressions rationnelles*, est l'ensemble des mots sur $\Sigma \cup S$ défini inductivement par

$$\frac{}{\varnothing \in \mathsf{Regexp}(\Sigma)} \qquad \frac{}{\Sigma \in \mathsf{Regexp}(\Sigma)} \qquad \frac{a \in \Sigma}{a \in \mathsf{Regexp}(\Sigma)}$$

$$\frac{e \in \mathsf{Regexp}(\Sigma)\ f \in \mathsf{Regexp}(\Sigma)}{(e|f) \in \mathsf{Regexp}(\Sigma)} \qquad \frac{e \in \mathsf{Regexp}(\Sigma)\ f \in \mathsf{Regexp}(\Sigma)}{(ef) \in \mathsf{Regexp}(\Sigma)}$$

$$\frac{e \in \mathsf{Regexp}(\Sigma)}{(e^*) \in \mathsf{Regexp}(\Sigma)}$$

Remarques :
- $|$ est parfois noté $+$.
- On se passe de certaines parenthèses avec les règles de priorité :

$$* > \text{concaténation} > |$$

Par exemple, $((a\,(b^*))\,|b)$ s'écrit $ab^*|b$.

1.3.5 Définition (*langage dénoté par une expression régulière*)

Soit Σ un alphabet.
Le *langage dénoté par une expression régulière* $e \in \mathsf{Regexp}(\Sigma)$ est le langage $L(e)$ défini inductivement par :

$$L(\varnothing) = \varnothing \qquad L(\varepsilon) = \{\varepsilon\} \qquad \forall a \in \Sigma,\ L(a) = \{a\}$$

$$L(e|f) = L(e) \cup L(f) \qquad L(ef) = L(e)L(f) \qquad L(e^*) = L(e)^*$$

Exemple :
$L_1 = \{\varepsilon\} = L(\varnothing^*)$
$L_2 = \Sigma = L(a_1|a_2|\cdots|a_n)$ en notant $\Sigma = \{a_k \mid k \in [\![1\ ;\ n]\!]\}$

$$\begin{aligned} L_3 &= \{u \in \Sigma^* \mid |u| \equiv 1\ [2]\} \\ &= \bigcup_{n\in\mathbb{N}} \{u \in \Sigma^* \mid |u| = 2n+1\} \\ &= \left(\bigcup_{n\in\mathbb{N}} \{u \in \Sigma^* \mid |u| = 2n\}\right)\Sigma \\ &= L((\Sigma\Sigma)^*\Sigma) \end{aligned}$$

$L_4 = \{u \in \Sigma^* \mid a \text{ préfixe de } u \text{ et } b \text{ suffixe de } u\} = L(a\Sigma^*b)$
$L_5 = \{u \in \Sigma^* \mid ab \text{ est facteur de } u\} = L(\Sigma^*ab\Sigma^*)$

1.3.6 Proposition

> Soit Σ un alphabet, et L un langage sur Σ.
> Alors :
> $$L \text{ est régulier} \quad \Leftrightarrow \quad \exists e \in \mathsf{Regexp}(\Sigma) \mid L = L(e)$$

□

$\boxed{\Leftarrow}$ $\boxed{\text{Exo}}$ (par induction sur e)

$\boxed{\Rightarrow}$ par induction sur une dérivation de $L \in \mathsf{Reg}(\Sigma)$, en faisant une disjonction de cas selon la dernière règle utilisée.

- $\dfrac{}{\varnothing \in \mathsf{Reg}(\Sigma)}$: alors $L = \varnothing = L(\varnothing)$
- $\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)}$: alors $L = \{a\} = L(a)$
- $\dfrac{L_1 \in \mathsf{Reg}(\Sigma)\ L_2 \in \mathsf{Reg}(\Sigma)}{L_1 \cup L_2 \in \mathsf{Reg}(\Sigma)}$: alors $L = L_1 \cup L_2$, et par H.I :

$$\exists e_1, e_2 \in \mathsf{Regexp}(\Sigma) \left| \begin{array}{l} L_1 = L(e_1) \\ L_2 = L(e_2) \end{array} \right.$$

D'où $L = L_1 \cup L_2 = L(e_1) \cup L(e_2)$

- $\dfrac{L_1 \in \mathsf{Reg}(\Sigma)\ L_2 \in \mathsf{Reg}(\Sigma)}{L_1L_2 \in \mathsf{Reg}(\Sigma)}$: alors $L = L_1L_2$, et par H.I :

$$\exists e_1, e_2 \in \mathsf{Regexp}(\Sigma) \left| \begin{array}{l} L_1 = L(e_1) \\ L_2 = L(e_2) \end{array} \right.$$

D'où $L = L_1L_2 = L(e_1)L(e_2)$

- $\dfrac{L_0 \in \mathsf{Reg}(\Sigma)}{L_0^* \in \mathsf{Reg}(\Sigma)}$: alors $L = L_0^*$, et par H.I :

$$\exists e \in \mathsf{Regexp}(\Sigma) \left| L_0 = L(e) \right.$$

D'où $L = L_0^* = L(e)$ ■

Exemple : $L = \{ab, ac\}$

$$\dfrac{\dfrac{\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{b \in \Sigma}{\{b\} \in \mathsf{Reg}(\Sigma)}}{\{ab\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{\dfrac{a \in \Sigma}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{c \in \Sigma}{\{c\} \in \mathsf{Reg}(\Sigma)}}{\{ac\} \in \mathsf{Reg}(\Sigma)}}{\{ab, ac\} \in \mathsf{Reg}(\Sigma)}$$

$$ab|ac$$

$$\dfrac{\dfrac{a \in \mathsf{Reg}(\Sigma)}{\{a\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{\dfrac{b \in \Sigma}{\{b\} \in \mathsf{Reg}(\Sigma)} \quad \dfrac{c \in \Sigma}{\{c\} \in \mathsf{Reg}(\Sigma)}}{\{b, c\} \in \mathsf{Reg}(\Sigma)}}{\{ab, ac\} \in \mathsf{Reg}(\Sigma)}$$

$$a(b|c)$$

1.3.7 Définition (*équivalence d'expressions régulières*)

Soit Σ un alphabet, et $e, f \in \mathsf{Regexp}(\Sigma)$.
Alors e, f sont dits équivalents, noté $e \equiv f$, si et seulement si $L(e) = L(f)$.

1.3.8 Proposition

> Soit Σ un alphabet, et $e \in \mathsf{Regexp}(\Sigma)$.
> (1) Si e ne contient pas $\varnothing$, alors $L(e) \neq \varnothing$
> (2) La réciproque est fausse
> (3) Si $L(e) \neq \varnothing$, alors $\exists f \in \mathsf{Regexp}(\Sigma) \mid e \equiv f$ et f ne contient pas $\varnothing$.

□

(1) Exo par induction sur e
(2) $a|\varnothing$ avec $a \in \Sigma$
(3) Par induction sur e :

- $e = \varnothing$: impossible
- $e = \Sigma$ ou $e = a$ avec $a \in \Sigma$, $f = e$ convient
- $e = e_1|e_2$: $L(e_1) = \varnothing = L(e_2)$ est impossible car $L(e) = L(e_1) \cup L(e_2)$
  - Si $L(e_1) = \varnothing$ et $L(e_2) \neq \varnothing$ (l'autre cas est symétrique), $L(e) = L(e_2)$, et par H.I, $\exists f_2 \in \mathsf{Regexp}(\Sigma) \mid f_2$ ne contient pas $\varnothing$ et $f_2 \equiv e_2$.
    Alors $e \equiv f_2$.
  - Si $L(e_1) \neq \varnothing$ et $L(e_2) \neq \varnothing$, par H.I, $\exists f_1, f_2 \in \mathsf{Regexp}(\Sigma) \mid f_1, f_2$ ne

contiennent pas $\varnothing$, et

$$\begin{cases} f_1 \equiv e_1 \\ f_2 \equiv e_2 \end{cases}$$

Alors $L(e) = L(e_1) \cup L(e_2) = L(f_1) \cup L(f_2) = L(f_1|f_2)$ et $e \equiv f_1|f_2$ qui ne contient pas $\varnothing$.

- $e = e_1e_2$: $L(e_1) = \varnothing$ ou $L(e_2) = \varnothing$ est impossible car $L(e) = L(e_1)L(e_2)$.

De même, on applique l'H.I à e_1 et e_2 pour obtenir deux expressions régulières dont on prend la concaténation.

- $e = l_0^*$:
  - Si $L(e_0) = \varnothing$, alors $L(e) = \{\varepsilon\} = L(\varepsilon)$
  - Sinon, on applique l'H.I à e_0 pour obtenir une expression régulière dont on prend l'étoile de KLEENE

■

1.3.9 Proposition

Soit Σ un alphabet, et $e \in \mathsf{Regexp}(\Sigma)$.

On définit le terme constant de e inductivement par

$$c(\varnothing) = 0 \qquad c(\varepsilon) = 1 \qquad c(a) = 0$$
$$c(ef) = \min(c(e), c(f)) \qquad c(e|f) = \max(c(e), c(f)) \qquad c(e^*) = 1$$

Alors :

$$\varepsilon \in L(e) \Leftrightarrow c(e) = 1$$

□

Exo par induction sur e. ■

1.4 Expressions régulières étendues

1.4.1 Définition (*expressions régulières étendues*)

Soit Σ un alphabet.

On étend la définition des expressions régulières en ajoutant deux symboles $\cap$ et $\setminus$ à S et en ajoutant les règles d'inférence suivantes :

$$\frac{e \in \mathsf{Regexp}(\Sigma)\ f \in \mathsf{Regexp}(\Sigma)}{e \cap f \in \mathsf{Regexp}(\Sigma)} \qquad \frac{e \in \mathsf{Regexp}(\Sigma)\ f \in \mathsf{Regexp}(\Sigma)}{(e \setminus f) \in \mathsf{Regexp}(\Sigma)}$$

Les langages dénotés sont

$$L(e \cap f) = L(e) \cap L(f)$$
$$L(e \setminus f) = L(e) \setminus L(f)$$

Nous verrons que les expressions régulières étendues dénotent les mêmes langages que les expressions régulières.

1.4.2 Application

Les expressions régulières (étendues) sont liées aux expressions régulières de la norme POSIX, utilisées par exemple dans la commande `grep`.

Exemple :

```
find | grep -E '.*/d[ms][1/]*\.(pdf|tex)$' | grep 'corrigé'
```

trouve les fichiers dont le nom contient `dm` ou `ds`, d'extension `.pdf` ou `.tex`, et dont le nom contient `corrigé`.

1.4.3 Définition (*langages des préfixes / suffixes / facteurs*)

Soit L un langage sur un alphabet Σ.

On défini les langages des préfixes / suffixes de longueur 1 et des facteurs de longueur 2 des mots de L par

$$\begin{aligned} P_1(L) &= \{a \in \Sigma \mid \exists u \in \Sigma^* \mid au \in L\} \\ S_1(L) &= \{a \in \Sigma \mid \exists u \in \Sigma^* \mid ua \in L\} \\ F_2(L) &= \{u \in \Sigma^2 \mid \exists x, y \in \Sigma^* \mid xuy \in L\} \end{aligned}$$

1.4.4 Langages locaux

- Exo pour tout langage L sur un alphabet Σ,

$$L \setminus \{\varepsilon\} \subseteq (P_1(L)\Sigma^* \cap \Sigma^* S_1(L)) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2(L)\right) \Sigma^*\right)$$

mais cette inclusion peut être stricte.

- Définition : Soit Σ un alphabet, et L un langage sur Σ.

L est *local* si et seulement si

$$L \setminus \{\varepsilon\} = (P_1(L)\Sigma^* \cap \Sigma^* S_1(L)) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2(L)\right) \Sigma^*\right)$$

- Exemples :
  - $(ab)^*$ et $(ab)^+$ sont locaux
  - $P_1 = \{a\}$, $S_1 = \{b\}$, et $F_2 = \{a, b\}^2 \setminus \{ab, ba\}$ définissent les langages locaux ε et $\varnothing$

- Proposition :

> Tout langage local est dénoté par une expression régulière étendue

□

En utilisant la définition et le fait qu'un langage fini est régulier Exo ■

• Exo : Donner un algorithme permettant de déterminer $P_1(L)$, $S_1(L)$, $F_2(L)$ pour tout langage régulier.
Indication : induction sur une expression régulière dénotant L.

1.4.5 Langages linéaires

• Définition (*expression régulière linéaire*) : une expression régulière est dite *linéaire* si et seulement si chaque lettre qui la compose n'y apparaît qu'une seule fois.

• Exemple : $(ab)^*|cd^*e$ mais pas $(ab^*)|ca$

• Définition (*langage linéaire*) : un langage est *linéaire* si et seulement si il est dénoté par une expression régulière linéaire.

• Proposition :

> Tout langage linéaire est local.

□

– Lemme 1 :

> *Lemme 1* : Soient L_1, L_2 deux langages locaux sur les alphabets disjoints.
> Alors $L_1 \cup L_2$ est local

Exo

– Lemme 2 :

> *Lemme 2* : Soient L_1, L_2 deux langages locaux sur des alphabets disjoints.
> Alors $L_1 L_2$ est local.

Exo
Indication : distinguer quatre cas selon que $\varepsilon \in L_1$ ou $\varepsilon \in L_2$.

– Par induction sur une expression régulière linéaire dénotant le langage
  . $\varnothing$ ou ε : $P_1 = S_1 = F_2 = \varnothing$ conviennent
  . $a \in \Sigma$: $P_1 = S_1 = \{a\}$ et $F_2 = \varnothing$ convient
  . $e|f$: par H.I, $L(e)$ et $L(f)$ sont locaux.
  De plus, come $e|f$ est linéaire, e et f sont exprimés sur des alphabets disjoints.
  Donc le lemme 1 conclut.
– ef : le même raisonnement avec le lemme 2.

– e^* : par H.I, $L(e)$ est local, donc

$$L(e) \setminus \{\varepsilon\} = (P_1\Sigma^* \cap \Sigma^* S_1) \setminus \left(\Sigma^* \left(\Sigma^2 \setminus F_2\right) \Sigma^*\right)$$

$p_1(e^*) = P_1(e)$
$S_1(e^*) = S_1(e)$
$F_2(e^*) = F_2(e) \cup S_1(e)P_1(e)$ ■

• Contre-exemples :

– lemmes 1 et 2 : $L_1 = ab$ et $L_2 = a^*$

Si $L_1 \cup L_2$ était local, comme $P_1 = \{a\}$, $S_1 = \{a, b\}$, $F_2 = \{ab, aa\}$, on aurait $aab \in L_1 \cup L_2$: absurde

Si L_1L_2 était local, comme $P_1 = \{a\}$, $S_1 = \{a, b\}$, $\{ab, ba, aa\}$, on aurait $aab \in L_1L_2$: absurde

– $a(ba)^*$ est local mais non linéaire.

• Linéarisation : on peut transformer une expression régulière e en expression régulière linéaire $\tilde{e}$ en numérotant ses lettres (l'alphabet devient $\Sigma \times [\![1\ ;\ n]\!]$)

Exemple : $aa(a|ab)^*b$ devient $a_1a_2(a_3|a_4b_1)^*b_2$
C'est une étape importante pour faire le lien entre automates et langages réguliers.
Exo $\forall u,\ u \in L(e) \Leftrightarrow \exists$ numérotation des lettres de u donnant $\tilde{u} \in L(\tilde{e})$