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 :

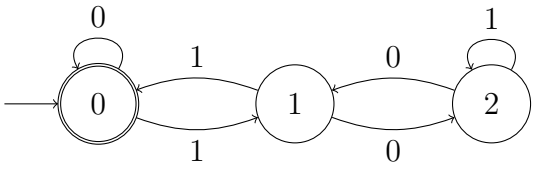
$\langle x \rangle_2 \mod 3$	0	1	2
$\langle x0 \rangle_2 \mod 3$	0	2	1
$\langle x1 \rangle_2 \mod 3$	1	0	2

- Algorithme :

Algorithm 1:

```
x ← 0;
for chaque bit b pris dans l'odre do
    if b = 0 then
        x ← 2x  mod 3;
    else
        x ← 2x + 1  mod 3
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 Kleine** : Soit Σ un alphabet, et L un langage sur Σ.
L'*étoile de* Kleine 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,

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

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 Kleine. 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 $ab^*|b$.