Université Paris 7 - L2 "Sciences et Applications"
Mention "Mathématiques, Informatique"
Parcours "Informatique" et "Mathématiques et Informatique"
Automates finis [AF3] - 2005/2006 - Partie 1
Introduction générale : notions de mots et langages.
Terminologie et notations utilisées

Informations générales

- Page du cours :
  `http://www.pps.jussieu.fr/~rifflet/enseignements/AF3`
- Horaires des TD et TP :
  - ◇ Groupe MAIF :
    TD : Me 8h30 [P42-N1] TP : Je 10h30 [IFN]
  - ◇ Groupe INF1 :
    TD : Je 14h30 [IJ4] TP : Je 16h30 [IFN]
  - ◇ Groupe INF2 :
    TD : Ma 8h30 [P14-J4] TP : Je 14h30 [IFN]
  - ◇ Groupe INF3 :
    TD : Je 14h30 [33-34. RdC 9] TP : Ma 14h30 [IFM]
  - ◇ Groupe INF4 :
    TD : Je 8h30 [P14-J3] TP : Je 12h30 [IFE]
- Contrôle des connaissances :
  - Pa : partiel le samedi 26 novembre 2005 à 8h30
  - Ej : examen obligatoire (absence éliminatoire) début janvier 2006
  - Td : contrôle en TD obligatoire (absence = 0)
  - Tp : contrôle en TP obligatoire (absence = 0)
  - Ec : note d'écrit = max(Ej, (Pa+Ej)/2)
  - N1 : note de la 1-ère session = (6Ec+2Td+2Tp)/10

1 Mots

$\hookrightarrow$ Un **alphabet** est un ensemble (supposé fini) dont les éléments sont appelés **lettres**. Un alphabet sera par exemple noté X ou Σ.

$\hookrightarrow$ Un **mot** (fini) w sur l'alphabet X est une suite (finie) de lettres et est noté par simple juxtaposition : $w = x_1x_2...x_n$ où $\forall i \in \{1, ..., n\}$, $x_i \in X$

Un mot w consiste donc en la donnée d'une fonction de l'ensemble $\mathcal{N}^*$ des entiers strictement positifs dans X, définie uniquement sur un segment initial $[1 : n]$ (en abrégé $[n]$) de $\mathcal{N}^*$.

Cet entier n est appelé **longueur** du mot w et est noté $\mid w \mid$.

$\hookrightarrow$ Un mot w associe donc à un entier i, compris entre 1 et $\mid w \mid$, une lettre de l'alphabet X, qui est notée w_i. Si w_i est la lettre x, on parle d'**occurrence** de la lettre x dans w (en position ou au rang i).

On note $\mid w \mid_x$ le nombre d'occurrences de la lettre x dans le mot w.

$\hookrightarrow$ Le **mot vide** est le mot (unique) de longueur 0. Des notations possibles en sont, suivant le contexte : 1_X ou 1 ou e ou λ ou ϵ .

$\hookrightarrow$ *Des exemples*

- les entiers naturels 10 sont des mots sur l'alphabet `{0,1, ...,9}`
- l'alphabet hexadécimal contient les 10 chiffres plus `A,B,C,D,E,F`
- les digicodes sont des mots formés sur un alphabet contenant les chiffres et un certain nombre de lettres
- les identificateurs d'un langage de programmation tel que Java sont des mots sur l'alphabet constitués des lettres, des chiffres décimaux et du caractère `_`
- les gènes sont des mots sur l'alphabet `{A,C,G,T}` (correspondant aux 4 bases de l'ADN, l'Adénine, la Cytorine, la Guamine et la Thymine).

$\hookrightarrow$ Étant donnés deux mots u et v, on note $u.v$ ou, de manière abrégée uv, le mot w de longueur $\mid u \mid + \mid v \mid$ obtenu en mettant bout à bout les lettres de u et les lettres de v.

Ce mot w, appelé (**produit de**) **concaténation** de u et v, correspond donc à la fonction de $\mathcal{N}^*$ dans X définie sur $[\mid u \mid + \mid v \mid]$ telle que :

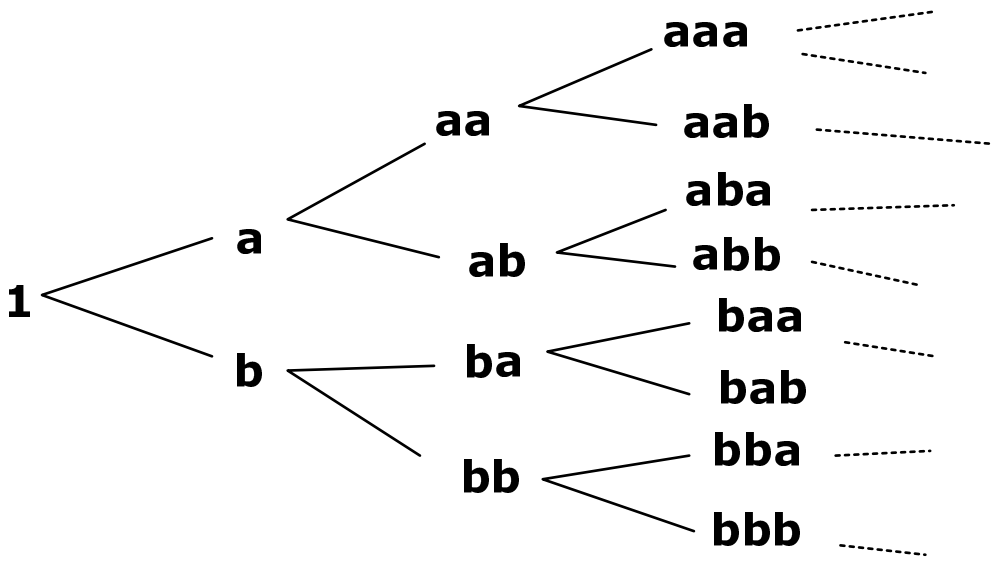
- $\forall i, 1 \leq i \leq \mid u \mid$, $w_i = u_i$
- $\forall i, 1 \leq i \leq \mid v \mid$, $w_{i+\mid u \mid} = v_i$

$\hookrightarrow$ Muni de la concaténation comme loi de composition interne, l'ensemble de tous les mots sur un alphabet X, qui est noté X^*, possède une structure algébrique appelée **monoïde** qui correspond au fait que la loi de composition interne possède les deux propriétés suivantes :

- elle est associative, c'est-à-dire que
  $$\forall u, v, w \in X^*, (u.v).w = u.(v.w)$$
- elle possède un élément neutre à gauche et à droite (le mot vide) :
  $$\forall u \in X^*, 1_X.u = u.1_X = u$$

L'ensemble X^* est appelé **monoïde libre** engendré par X.

↪ Il est commode de représenter X^* comme une arborescence infinie, en particulier pour numéroter les mots (ci-après l'arbre pour l'alphabet `{a,b}` :



↪ **Facteurs et sous-mots**

- Un mot f est dit **facteur** d'un mot w s'il existe des mots u et v tels $w = ufv$ (abréviation pour $w = (u.f).v$)
  Ainsi le mot $aabab$ a 12 facteurs : ϵ, a, aa, aab, $aaba$, $aabab$, ab, aba, $abab$, b, ba et bab.
  De plus :
  - ◇ si $uv \neq \epsilon$, f est un **facteur propre** ;
  - ◇ si $u = \epsilon$, f est dit **facteur gauche** ou **préfixe** de u ;
  - ◇ si $v = \epsilon$, f est dit **facteur droit** ou **suffixe** de u.
- Un mot v est un **sous-mot** d'un mot u s'il est obtenu par effacement de certaines occurrences de lettres dans u (contrairement à ce qui se passe pour un facteur, les lettres de v n'ont pas nécessairement des occurrences consécutives dans u). D'un point de vue de la définition fonctionnelle des mots, cela signifie que le mot v est la fonction composée d'une certaine application strictement croissante de $[\mid v \mid]$ dans $[\mid u \mid]$ ($\mid v \mid \leq \mid u \mid$) et de la fonction définissant le mot u.
  Ainsi, les mots aaa et bb sont deux sous-mots de $aabab$ sans en être des facteurs.

↪ **Lemme de Levy**

Soient u, v, w, z des mots sur un alphabet X tels que $uv = wz$.
Il existe alors un mot y tel que :

- ou bien $uy = w$ et $v = yz$
- ou bien $u = wy$ et $yv = z$

↪ **Ordre lexicographique**

On suppose tout d'abord que l'alphabet X (de n lettres) est totalement ordonné : $x_1 < x_2 < \ldots < x_n$.

On en déduit l'ordre lexicographique sur les mots (ordre total), défini par $u < v$ (on note u_i et v_i les lettres successives du u et v) si et seulement si

- ◇ *ou bien* u est facteur gauche de v (c'est-à-dire qu'il existe un mot α tel que $v = u.\alpha$) ;
- ◇ *ou bien* $\exists j \in \mathcal{N}$ tel que $\forall i < j, u_i = v_i$ et $u_j < v_j$

2 Langages

↪ On appelle **langage** (formel) tout sous-ensemble L de X^*, c'est-à-dire $L \subset X^*$. Parmi les langages il convient de distinguer $\emptyset$ (l'ensemble vide qui ne contient aucun mot) et le langage $\{\epsilon\}$ (langage qui contient comme seul élément le mot vide).

↪ **Opérations sur les langages** :

Un certain nombre d'opérations peuvent être réalisées sur les langages (X désigne l'alphabet) :

- les opérations ensemblistes classiques :
  - ◇ l'union : $A \cup B = \{w \in X^* \mid w \in A \text{ ou } w \in B\}$
  - ◇ l'intersection : $A \cap B = \{w \in X^* \mid w \in A \text{ et } w \in B\}$
  - ◇ le complément par rapport à X^* : $A^c = \{w \in X^* \mid w \notin A\}$
  - ◇ la différence : $A \backslash B = A - B = A \cap B^c$
    $= \{w \mid w \in X^* \mid w \in A \text{ et } w \notin B\}\}$
- le produit de concaténation : $A.B = \{\ u.v / u \in A, v \in B\}$
- puissance d'un langage : $L^2 = L.L$ et $\forall n \in \mathcal{N}, L^{n+1} = L^n.L$
  (avec $L^0 = \{\epsilon\}$)
- (le passage à) l'etoile de Kleene : le langage L^* est défini par :
  $L^* = L^0 \cup L \cup L^2 \cup \ldots \cup L^n \ldots$
  $= \{u \mid \exists n \in \mathcal{N}, u_1, \ldots, u_n \in L \text{ tels que } u = u_1 \ldots u_n\}$
- l'opération *plus* : $L^+ = L.L^* = L \cup L^2 \cup \ldots \cup L^n \ldots$

↪ **Description de langages**

On s'intéresse ici à différentes manières de décrire des langages, c'est-à-dire de décrire des ensembles de mots. Cela est plus ou moins simple selon la nature du langage considéré. Ainsi, décrire un langage fini peut se faire par exemple par simple énumération des mots qui le composent. Par contre, la description du français vu comme un langage au sens dont nous venons de les définir, c'est-à-dire comme un ensemble de mots (des phrases correctes) construites sur un alphabet (en se limitant aux 26 lettres minuscules et à l'espace, et en omettant donc les signes de ponctuation, les accents, la distinction minuscule/majuscule, les chiffres) est autrement plus complexe.

• On peut donner des langages une *description en langue naturelle* (en quelque sorte littéraire) comme par exemple :

- ◇ langage L_1 sur l'alphabet `{0,1}` : ensemble des mots dont l'interprétation comme un entier écrit en base 2 est un multiple du nombre trois. Il contient par exemple le mot `101111101` (dont l'expression en base 10 est `381`) mais pas `101111100` (le nombre `380`) ;
- ◇ langage L_2 sur l'alphabet `{a,b}` formés de tous les mots palindromes, c'est-à-dire pouvant se lire indifféremment de gauche à droite ou de droite à gauche. Ainsi le langage contient `abbaaabbaaabba` mais pas `abbabaabba` ;
- ◇ le langage L_3 consistant en l'ensemble des entiers naturels définissables en français (avec l'alphabet de 27 lettres défini précédemment) par un mot contenant au plus dix sept espaces.
  Ce langage est intéresssant car il conduit à un paradoxe. Soit en effet le plus petit nombre entier naturel n n'appartenant pas à L_3. Ce nombre est définissable par le mot (donc, phrase sans accent ni signe de ponctuation où l'espace permet de séparer les mots) :

  le plus petit entier naturel non definissable en francais par un mot contenant au plus dix sept espaces

  Cette définition est un mot contenant exactement 17 espaces (le caractère de fin de ligne est assimilé à un espace) : le nombre n qu'il définit devrait de ce fait appartenir au langage L_3.
  La seule conclusion semble être que ce langage contient effectivement tous les nombres. Tout nombre serait donc définissable par un mot contenant au plus dix sept espaces, par exemple

  `13524567893457812935678123`1 !!!

  Cet exemple illustre assez clairement le danger de ce type de définitions et l'intérêt de disposer de méthodes de description plus formelles définissant de manière inattaquable (sans ambiguité) les langages.

• *Les descriptions énumératives* : elles sont clairement utilisables pour les langages finis, mais également pour certains langages infinis tels que le langage $L_4 = \{a^n b^n / n \geq 1\}$ contenant tous les mots formés exactement d'une suite de n occurrences de la lettre a suivie d'une suite contenant le même nombre n d'occurrences de la lettre b.

• *Les définitions par expression* : c'est ce que fait par exemple la commande `grep` d'*Unix* avec un type d'expressions particulier. L'expression `ab*cccab` représente alors le langage L_5 dont les mots commencent par une occurrence de la lettre a, suivie par un nombre quelconque (éventuellement nul) d'occurrences de la lettre b, suivi du facteur $cccab$ qui facteur droit du mot.

• *Les mécanismes génératifs*, appelés **grammaires** ou **systèmes de réécriture** : ils définissent un mécanisme de génération des mots sous la forme de règles de construction inductives. C'est de cette manière qu'on décrit la syntaxe des langages de programmation et qu'on définit un certain nombre de phrases correctes dans les langues naturelles.

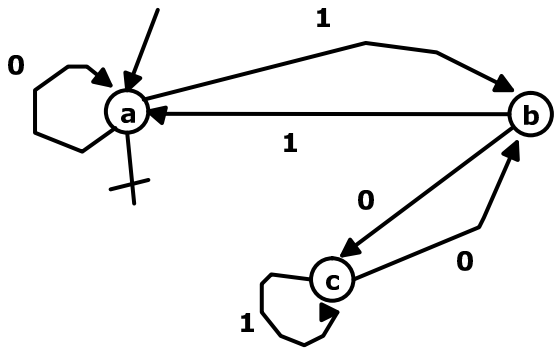
• *Les mécanismes de reconnaissance*, appelés **automates** ou **machines** : ils permettent de dire si un mot appartient (et éventuellement n'appartient pas) au langage considéré. Un analyseur syntaxique d'un langage de programmation est typiquement une telle machine : elle permet de dire si un programme écrit dans le langage correspondant est ou non syntaxiquement correct.

↪ **Hiérarchie de langages** :
Nous allons nous intéresser dans la suite de ce cours à la définition formelle des langages et donc aux aspects automates, machines et expressions. Selon la complexité des automates ou des grammaires utilisées, différents types de langages peuvent être définis. Cette complexité définit une classification des familles de langages (dans ce cours nous nous concentrerons sur la famille du premier type). Le tableau ci-dessous résume cette classification :

Famille	Type d'automate	Type de grammaire
reconnaissables ou **rationnels** ou **réguliers**	**automates finis**	grammaires linéaires d'un côté (à droite (ou à gauche)
algébriques ou *context-free*	**automates à pile**	grammaires algébriques ou *context-free*
context-sensitive ou **contextuels**	**automates à mémoire linéairement bornée**	grammaires *context-sensitive* ou contextuelles
contextuels **récursivement énumérables**	**machine de Türing**	systèmes de réécriture généraux

Dans cette dernière famille (celle des langages récursivement énumérables), la sous-famille des langages dont le complémentaire est également récursivement énumérable est particulièrement importante : il s'agit de la famille des **langages récursifs**. Elle a un rôle essentiel dans la mesure où elle correspond à une formalisation de ce qui est effectivement calculable par une machine.
Les différentes familles sont strictement incluses les unes dans les autres (la plus petite est celle des reconnaissables et les récursifs sont placés entre les contextuels et les récursivement énumérables).
Exemples de langages des différentes familles :

- les langages L_1 et L_5 sont reconnaissables :
  - ◇ le langage L_1 est l'ensemble des mots qui, quand il les lit complètement, font passer l'automate fini représenté par le schéma suivant de l'état a à l'état a :



- ◇ le langage L_5 est l'ensemble des mots sur l'alphabet `{a,b,c}` produits par les règles de grammaire suivantes en partant initialement de S : $S \to aT \quad T \to bT \quad T \to U \quad U \to cccab$

- les langages L_2 et L_4 sont algébriques (mais pas reconnaissables) :
  - ◇ le langage L_4 est l'ensemble des mots qui, quand il les a lus complètement, en partant de l'état q_0 et avec le symbole V sur la pile, se retrouve à l'état q_1 avec une pile vide, en appliquant les règles de fonctionnement suivantes (les configurations rencontrées et non définies bloquent la machine et entrainent le rejet du mot lu) :

Etat courant	Sommet de pile	Lettre lue	Nouvel état	Ecriture sur la pile
q_0	V	a	q_0	B
q_0	B	a	q_0	BB
q_0	B	b	q_1	ϵ
q_1	B	b	q_1	ϵ

  - ◇ le langage L_2 est l'ensemble des mots sur l'alphabet `{a,b}` produits par les règles de grammaire suivantes en partant initialement de S : $S \to aSa \quad S \to bSb \quad S \to a \quad S \to b \quad S \to \epsilon$

  Cette grammaire a la propriété que son membre gauche est réduit à un caractère n'appartenant pas à l'alphabet sur lequel le langage est défini.

- le langage $L_6 = \{a^n b^n c^n \mid n \geq 1\}$ est un langage contextuel (mais pas algébrique, et donc pas non plus reconnaissable). Il correspond à l'ensemble des mots sur l'alphabet `{a,b,c}` produits par les règles de grammaire suivantes en partant initialement de S :

  $S \to aSBC \quad S \to aBC$
  $CB \to BC \quad aB \to ab$
  $bB \to bb \quad bC \to bc \quad cC \to cc$

  Cette grammaire a la propriété de non décroissance des longueurs (la longueur de la partie droite d'une règle est au moins égale à celle de sa partie gauche).

  Un autre exemple de langage contextuel qui n'est pas algébrique est le langage contenant les mots carrés, c'est-à-dire le langage

  $$L_7 = \{w \mid \exists u \text{ tel que } w = u.u\}$$

- exhiber un langage qui soit récursif et non contextuel est un peu plus compliqué. Cela nécessite l'utilisation de l'argument de diagonalisation de Cantor :

  1. on peut numéroter les mots de X^* (en parcourant l'arbre qui lui est associé par niveau) : il est ainsi possible de parler du i-ème mot m_i de X^* ;
  2. on peut par ailleurs numéroter les grammaires textuelles en leur associant un mot de X^* de telle sorte :
     - à des grammaires différentes sont associés des mots différents (il s'agit donc d'une opération de codage) ;
     - étant donné un mot de X^*, il est facile de savoir s'il code une grammaire contextuelle (tous les mots de X^* ne sont pas associés à une grammaire textuelle) ;
     - il est ainsi possible de parler de la i-ème grammaire textuelle G_i (on note par ailleurs $L(G_i)$ le langage engrendré par G_i) ;
  3. soit le langage $L = \{m_i \mid m_i \notin L(G_i)\}$. C'est un langage récursif : il est facile de retrouver i à partir de m_i, puis de retrouver G_i et enfin il existe un algorithme pour tester si un mot appartient à un langage contextuel.
     Supposons que L soit contextuel : $\exists j$ tel que $L = L(G_j)$.
     - si $m_j \in L$, $m_j \in L(G_j)$ (car $L = L(G_j)$), ce qui est contradictoire avec la définition de L ;
     - si $m_j \notin L$, $m_j \notin L(G_j)$. Par définition de L, $m_j \in L$, d'où nouvelle contradiction.
     Le langage L n'est donc pas contextuel.

- l'existence d'un langage récursivement énumérable qui ne soit pas récursif se montre par un argument diagonal identique en numérotant les machines de Türing (ce qui permet de parler de la i-ème machine). Si on considère alors le langage
  $$L = \{m_i \mid m_i \text{ n'est pas accepté par la machine } T_i\}$$
  on peut montrer qu'il n'est pas récursivement énumérable et donc que son complémentaire n'est pas récursif.

$\hookrightarrow$ La notion de **code**.

Une application τ de X dans Y^* est un codage ou $\{\tau(x) \mid x \in X\}$ est un code si τ s'étend en une injection τ^* de X^* dans Y^* :

- s'étend veut dire : $\forall x_1, x_2, \ldots, x_n \in X$ $(u = x_1 x_2 \ldots x_n \in X^*)$,
  $\tau^*(u) = \tau^*(x_1 x_2 ... x_n) = \tau(x_1).\tau(x_2)...\tau(x_n)]$.
- injectif veut dire : si u et v sont des mots différents de X^*
  $\tau^*(u) \neq \tau^*(v)$