LANGAGES FORMELS

POLYCOPIE DE COURS

Hervé Dicky

Les deux livres qui ont été principalement utilisés pour rédiger ce cours sont :

- J.-M. Autebert :
  Langages algébriques 1987 Masson
- Patrick Dehornoy :
  Mathématiques de l'informatique, Cours et exercices corrigés 2000 Dunod

Vous pouvez toujours me contacter par e-mail à l'adresse dicky@lirmm.fr

Table des matières

Pourquoi ce cours ?

L'objectif de ce cours est de vous familiariser avec des objets très simples des mathématiques discrètes : mots et langages, automates (déterministes ou indéterministes), expressions rationnelles, grammaires. Ces objets mathématiques sont utiles en informatique à deux titres :

– ils sont utiles pour modéliser de nombreux problèmes simples susceptibles d'une résolution informatique,
– ils permettent de modéliser de nombreuses notions informatiques (par exemple toutes celles liées à la compilation).

Modéliser c'est représenter partiellement des situations, événements, objets ... du monde. Si l'on modélise en vue d'un traitement informatique ce sera nécessairement un modèle formel qui sera construit et manipulé, il est donc nécessaire, pour un informaticien, de connaître un certain nombre de notions mathématiques utiles pour construire de tels modèles formels.

Comment travailler ?
Ce cours contient des définitions d'objets mathématiques, ainsi que des énoncés et des démonstrations de propriétés portant sur ces objets.
L'un des objectifs de ce cours est de vous aider à acquérir des notions abstraites et des techniques de démonstration.
Avant de venir en TD, lisez la partie du polycopié traite dans le dernier cours et sachez refaire les démonstrations.

Rédaction [1]
Il y a une grande différence entre compréhension et rédaction. Quand vous cherchez à comprendre une notion, à énoncer une propriété, ou à faire une démonstration vous procéderez généralement par essai-erreur-rectification-nouvel essai, etc ... (comme tout le monde). En bref, il n'y a pas beaucoup de règles pour trouver une démonstration. Après avoir manipulé de nombreux exemples et contre-exemples et être convaincu de la justesse de ce qu'on veut prouver, on peut essayer des techniques générales (par exemple : démonstrations comme celles exposées ci dessous ou en cherchant à utiliser comme lemmes des propriétés vues en cours ...). Surtout ne croyez pas qu'à force de regarder l'énoncé du problème la solution viendra toute seule sous la forme d'une démonstration

1. texte emprunté sans vergogne à M. Chein

rédigée ...
Une fois que vous avez compris vous pouvez attaquer la phase de rédaction qui doit respecter d'autres règles. Il faut une certaine quantité de travail avant de pouvoir rédiger une démonstration telle que vous pouvez la lire dans un cours ou dans un livre. Une rédaction suit des règles spécifiques de rhétorique prenant en compte les lecteurs potentiels. Pour vous aider vous pouvez penser que, dans votre cas, les lecteurs attendus de vos démonstrations sont vos camarades d'étude. Faites leur lire vos démonstrations, et, s'ils ne comprennent pas ... recommencez ! Les "démonstrations" de ces notes de cours ne seront généralement, pour vous, que des indications de démonstration, vous aurez à les compléter afin d'avoir des démonstrations telles que nous les souhaitons.
N'hésitez pas à rédiger les solutions des exercices étudiés en TD.

Différentes méthodes de démonstration

1. *par contraposée*
   démontrer $A \Rightarrow B$ et démontrer $\neg B \Rightarrow \neg A$ sont équivalents.
2. *par l'absurde*
   pour démontrer $A \Rightarrow B$, on prend comme hypothèses A et $\neg B$ et on montre qu'on obtient une contradiction.
3. *par "glissement d'hypothèses"*
   pour démontrer $(A \Rightarrow B) \ \Rightarrow \ (C \Rightarrow D)$, il suffit de démontrer
   $(A \Rightarrow B \text{ et } C) \ \Rightarrow D$
4. *par cas*
   Pour démontrer $(A_1 \textit{ ou } A_2) \Rightarrow B$ il suffit de démontrer
   $(A_1 \Rightarrow B) \textit{ et } (A_2 \Rightarrow B)$.
5. *par récurrence*
   Soit P une proposition dépendant d'un entier naturel n
   $P \ : \ \forall n \ P(n)$.
   Si on peut démontrer $P(0)$ et si on peut démontrer que pour tout n de $\mathbb{N}$, $P(n) \Rightarrow P(n+1)$ alors pour tout n de $\mathbb{N}$ on a $P(n)$, c'est à dire qu'on a démontré P.

Un exemple de jeu nécessitant un raisonnement mathématique.
Alice choisit une cible et *Bob* une première valeur entre 1 et 9. Ensuite chaque joueur doit chacun à son tour choisir un nombre, supérieur au nombre précédemment choisis par son adversaire et au plus égal à deux fois ce nombre.
Le premier joueur qui dépasse la cible a gagné.

Chapitre 1

Mots et Langages

1.1 Notions de base

- Alphabet : un ensemble fini, par exemple $\Sigma = \{a,\ b\}$
- Mot sur Σ : suite **finie non bornée** d'éléments de Σ : $m\ =\ aabba$
- longueur d'un mot (et notations) : $|m|$ et $m[i]$: $|m|\ =\ 5$, $m[3]\ =\ b$
- le mot vide ϵ : $|\epsilon|\ =\ 0$
- facteur
  $\{\epsilon,\ a,\ aa,\ aab,\ aabb,\ aabba,\ ab,\ abb,\ abba,\ b,\ bb,\ bba,\ ba\}$ forme l'ensemble des facteurs de $aabba$.
  Notation : $m[i \dots j]$ par exemple $m[2 \dots 4] = abb$
- occurrence d'une lettre dans un mot :
  $aabba$ possède 3 occurrences de a et 2 occurrences de b.
- notation : $|m|_l$ par exemple $|aabba|_a\ =\ 3$
- préfixe, suffixe :
  $\{\epsilon,\ a,\ aa,\ aab,\ aabb,\ aabba\}$, $\{\epsilon,\ a,\ ba,\ bba,\ abba,\ aabba\}$

1.2 Opérations sur les mots

1.2.1 Concaténation, factorisation, image miroir : exemples

concaténation

- m_1 =para, m_2 =chute, $m_1 m_2$ =parachute
- $\forall m\ m\epsilon = \epsilon m = m$
- $m^2 = mm$, $m^k = \underbrace{mm \dots m}_{k}$, $m^0 = \epsilon$

factorisation d'un mot écriture sous forme d'une concaténation de facteurs. Un mot de n lettres a $n + 1$ factorisations en deux facteurs. Par exemple pour $aabba$: $\{(\epsilon,\ aabba)\ ,\ (a,\ abba)\ ,\ (aa,\ bba)\ ,\ (aab,\ ba)\ ,\ (aabb,\ a)\ ,\ (aabba,\ \epsilon)\}$

image miroir *abbaa*

1.2.2 Définitions formelles

concaténation de mots
La concaténation de 2 mots u et v, noté $u.v$ (ou s'il n'y a pas d'ambiguïté uv), est un mot m de longueur $|u| + |v|$ défini par :
- $\forall i \in [1, |u|],\ m[i] = u[i]$
- $\forall i \in [1, |v|],\ m[|u| + i] = v[i]$

le monoïde libre Σ^* : définition par schéma d'induction.
Σ^* est l'ensemble de tous les mots construits avec l'alphabet Σ. Cet ensemble peut être défini par le schéma inductif suivant (dans la concaténation ci-dessous, a étant le *mot* composé de la seule *lettre* a [1]) :

Base : $\epsilon \in \Sigma^*$
Règle : si $m \in \Sigma^*$ et $a \in \Sigma$ alors $a.m \in \Sigma^*$

On notera $\Sigma^+ = \Sigma^* - \{\epsilon\}$

Définition formelle des facteurs
Soit m un mot sur Σ.
- le mot u est un **préfixe** de m si et seulement si il existe un mot x tel que $m = u.x$
- le mot u est un **suffixe** de m si et seulement si il existe un mot x tel que $m = x.u$
- le mot u est un **facteur** de m s'il existe 2 mots x et y tels que $m = x.u.y$.
- u est un **facteur** (respect. **préfixe, suffixe**) **propre** de m si u est un facteur (respect. préfixe, suffixe) de m différent de m et de ϵ.
- $u = a_1a_2 \ldots a_n$ est un **sous-mot** de m s'il existe $n+1$ mots $w_0, w_1, \ldots w_n$ tels que $m = w_0.a_1.w_1.a_2.\ldots.a_n.w_n$.

1.2.3 Modélisation de vérification de trajectoires

Sur une grille, on code une trajectoire par son origine, et par ses déplacements successifs ($a,\ a'$ pour les déplacements verticaux, $b,\ b'$ pour les latéraux), comme dans la figure 1.1.

Le problème est de pouvoir vérifier si deux trajectoires données ne créent pas d'accidents (comme les trajectoires (3, 4) $aaba'ba'b'$ et (2, 3) $babb'$) ou en génèrent un (comme les trajectoires (4, 3) $aaba'ba'b'b'$ et (0, 1) $aaabbb$.
Soient deux trajectoires $T_1 = (x_1, y_1)m_1$ et $T_2 = (x_2, y_2)m_2$, si il existe un préfixe p_1 de m_1 et un préfixe de même longueur p_2 de m_2 tels que
$y_1 + |p_1|_a - |p_1|_{a'} = y_2 + |p_2|_a - |p_2|_{a'}$ et $x_1 + |p_1|_b - |p_1|_{b'} = x_2 + |p_2|_b - |p_2|_{b'}$, alors T_1 et T_2 génèrent un accident (il y a d'autres cas d'accident).

1. car la concaténation d'une lettre devant un mot n'est pas définie.

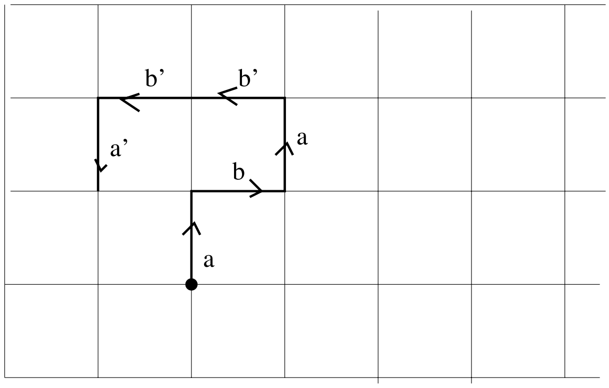


FIGURE 1.1 – Trajectoire issue du point (2, 1) et effectuant les déplacements $abab'b'a'$.

1.3 Ordres sur les mots

ordre sur un alphabet Σ $a <_\Sigma b$ donné à priori.

ordre préfixe $<_P$
$a <_P aa <_P aab <_P aabb <_P aabba$ $ab \not<_P ba$ et $ba \not<_P ab$
$m_1 <_P m_2$ si et seulement si m_1 est préfixe de m_2.

ordre lexicographique $<_L$
$loir <_L loire <_L loirs$

Soit $<_\Sigma$ un ordre total arbitraire sur Σ.
L'ordre lexicographique $<_L$ sur Σ^* est défini par :

$$\forall u, v \in \Sigma^*, u <_L v \text{ si et seulement si} \left| \begin{array}{l} \exists\, w, u' \in \Sigma^*, \exists\, v' \in \Sigma^+ \text{ tels que} \\ u = w.u', v = w.v' \text{ et } (u' = \epsilon \text{ ou } u'[1] <_\Sigma v'[1]) \end{array} \right.$$

Inconvénient : $a <_L aa <_L aaa <_L \ldots <_L aaaa \ldots$ et on n'atteint jamais b

ordre longueur lexicographique (hiérarchique) $<_H$
$a <_H b <_H aa <_H ab <_H ba <_H bb <_H aaa \ldots$
L'ordre hiérarchique sur Σ^*, noté $<_H$, est défini par :

$$\forall u, v \in \Sigma^*, u <_H v \text{ si et seulement si} \quad \begin{array}{l} |u| < |v| \text{ ou} \\ |u| = |v| \text{ et } u <_L v \end{array}$$

1.4 Manipuler les langages

1.4.1 Définition d'un langage sur un alphabet Σ

Un **langage** sur un alphabet Σ est un sous-ensemble de Σ^*.
Un langage peut être fini ou infini.

1.4.2 opérations sur les langages

Présentation intuitive

- union : $\{aa,\ abb,\ bba\} \cup \{ab,\ abb,\ bbb\} = \{aa,\ ab,\ abb,\ bba,\ bbb\}$
- intersection : $\{aa,\ abb,\ bba\} \cap \{ab,\ abb,\ bbb\} = \{abb\}$
- complémentation
  $L = \{m \in \{a,b\}^* \mid |m|_a = |m|_b\}$, $\bar{L} = \{m \in \{a,b\}^* \mid |m|_a \neq |m|_b\}$,
- concaténation, puissance
  $\{aa,\ bb\}\{ab,\ ba\} \ = \ \{aaab,\ aaba,\ bbab,\ bbba\}$
  $\{aa,\ bb\}^3 \ = \ \{aaaaaa,\ aaaabb,\ aabbaa,\ aabbbb,\ bbaaaa,\ bbaabb,\ bbbbaa,\ bbbbbb\}$

Définitions formelles Les opérations sur les langages sont :

- les opérations ensemblistes : l'union ($\cup$), intersection ($\cap$), la différence ensembliste ($\setminus$), le complémentaire (dans Σ^*).
- le *produit* ou concaténation de deux langages L_1 et L_2 définis sur un même alphabet Σ est le langage sur Σ noté $L_1.L_2$ défini par :
  $L_1.L_2 = \{m_1.m_2 \mid m_1 \in L_1 \text{ et } m_2 \in L_2\}$
- la *fermeture de Kleene* (ou *étoile*) d'un langage L sur un alphabet Σ est le langage sur Σ noté L^* et défini par

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

  où la suite (L_i) est une suite de langages sur Σ définie par
  - $L^0 = \{\epsilon\}$
  - $L^{n+1} = L.L^n \quad (\forall n \in \mathbb{N})$

  On utilisera également la notation L^+ pour noter $\displaystyle\bigcup_{i \in \mathbb{N}\setminus\{0\}} L^i$

1.5 Spécifier un langage sur un alphabet donné

1.5.1 Les différentes façons (en extension, par une propriété caractéristique des mots du langage, par un schéma de construction) :

- **en extension** : $L \ = \ \{ab,\ abb,\ bbb\}$
- **par une propriété caractéristique des mots du langage**
  $L \ = \ \{m \in \{0,1\}^* \mid m$ commence par un 1 et finit par un 0 $\}$
  ou autrement dit par un algorithme d'acceptation des mots du langage :
  si $m \in \{a,\ b\}^*$ et $|m|_a$ est premier et $|m|_b$ est un carré alors m est accepté.

– **par un schéma de construction** des mots du langage :
$10 \in L$ et $m \in L \;\Rightarrow\; m0 \in L$ et $m10 \in L$ et $10m \in L$ et $1m \in L$.

1.5.2 Dessin d'un mot sur un alphabet à deux lettres $\{a, b\}$

On monte en diagonal d'un cran pour la lettre a et on descend pour b comme on le voit dans la figure 1.2.

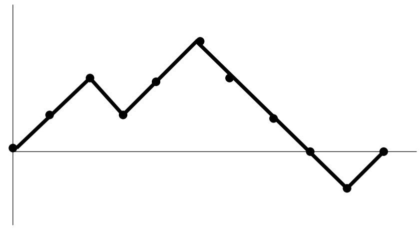


FIGURE 1.2 – Dessin du mot *aabaabbbba*

1.5.3 Langage fermé pour la concaténation et mots premiers d'un langage

On dit qu'un langage L est fermé pour la concaténation si et seulement si $\forall m_1, m_2 \in L \; : \; m_1 m_2 \in L$.
Si un langage L est fermé pour la concaténation, on dit qu'un mot $m \in L$ est premier si et seulement si m n'est pas la concaténation de deux mots non vides de L (le mot *aabaabbbba* du dessin de la figure 1.2 est la concaténation de deux mots premiers du langage des mots qui ont autant de a que de b).

exemple soit $L = \{m \in \{a, b\}^* \mid |m|_a = |m|_b\}$
L est fermé pour la concaténation et un mot m de L est premier si et seulement si m n'a pas de préfixe propre qui aie exactement autant de a que de b.

1.6 Combinatoire des mots

Théorème 1

$\forall x, y \in \Sigma^*,\ xy = yx \iff \exists z \in \Sigma^*\ \exists l, m \in \mathbb{N}$ tel que $x = z^l$ et $y = z^m$.
Admis [2]

Théorème à prouver

$$\forall u,\ v,\ w \in \Sigma^*,\ uvw = wuv \text{ et } uv \neq \epsilon \iff$$

$$\exists w_1, w_2 \in \Sigma^*\ \exists p, q, r \in \mathbb{N} \text{ tel que}$$

$$u = (w_1w_2)^p w_1,\ v = w_2(w_1w_2)^q \text{ et } w = (w_1w_2)^r \text{ et } w_1w_2 \neq \epsilon.$$

Attention ce théorème serait faux dans le seul cas (interdit) où $u = v = \epsilon \neq w$.

Cas triviaux

Si u (respectivement v) est le mot vide, nous nous trouvons dans le cas du théorème 1 et en prenant $p = 0$, $\omega_2 = \omega,\ \omega_1 = \epsilon$ (respectivement $q = 0$, $\omega_1 = \omega,\ \omega_2 = \epsilon$) nous déduisons dans ce cas le théorème 2 du théorème 1.

Principe de la démonstration

On va faire une démonstration par récurrence sur $l = |uvw|$.
On s'intéresse donc à la propriété :

$$\Pi(l)\ :\ \forall u, v, w \in \Sigma^* \text{ tel que } |uvw| \leq l \text{ et } uv \neq \epsilon :$$

$$uvw = wuv \iff \exists w_1, w_2 \in \Sigma^*\ \exists p, q, r \in \mathbb{N} \text{ tel que}$$

$$u = (w_1w_2)^p w_1,\ v = w_2(w_1w_2)^q \text{ et } w = (w_1w_2)^r \text{ et } w_1w_2 \neq \epsilon.$$

$\Pi(0)$ est vrai par vacuité, le problème sera de prouver que
$\forall l \in \mathbb{N} : \Pi(l) \Rightarrow \Pi(l+1)$.
Pour cela on prendra un mot uvw de longueur $l+1$ et on aura a étudier **8** cas :

2 cas spéciaux

1. $|u| = |w|$
2. $|v| = |w|$

3 cas simples

1. $|u| > |w|$ et $|v| > |w|$
2. $|u| > |w| > |v|$

---

2. sera démontré en TD.

3. $|u| < |w| < |v|$

1 cas complexe $|u| < |w|$ et $|v| < |w|$ qui se décompose en **3** sous-cas :

1. $|w| = |uv|$
2. $|w| < |uv|$
3. $|w| > |uv|$

Le premier point est évidement de se persuader que l'ensemble des situations est ainsi couvert. Le premier cas simple et le cas complexe couvrent les situations où la longueur de w est plus grande ou plus petite que la longueur des autres mots. Les deux autres cas simples couvrent les deux cas où $|w|$ est la longueur intermédiaire (les cas où il y a égalité sont les cas spéciaux).
Il faudra ensuite examiner chaque cas.

Cas $|u| > |w|$ **et** $|v| > |w|$ (figure 1.3)

Hypothèses	**Conclusions**
$\Pi(l)$	$\exists w_1, w_2 \in \Sigma^* \ \exists p, q, r \in \mathbb{N}$ tel que
$\lvert uvw\rvert = l + 1$	$u = (w_1 w_2)^p w_1, \ w = (w_1 w_2)^r.$
$uvw = wuv$	$v = w_2(w_1 w_2)^q$
$\lvert u\rvert > \lvert w\rvert$ et $\lvert v\rvert > \lvert w\rvert$.	

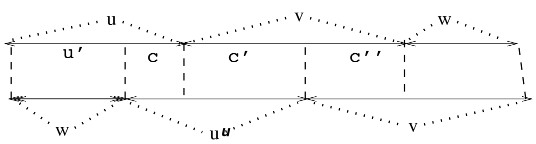


FIGURE 1.3 – $|u| > |w|$ et $|v| > |w|$

On voit, en faisant sauter c' sur la figure 1.3 que $uc''w = wcv = wuc''$ et que $|uc''w| < |uvw|$ donc $|uc''w| \leq l$

or d'après
$\Pi(l)$: $\forall u, c'', w \in \Sigma^*$ tel que $|uc''w| \leq l$:

$$uc''w = wuc'' \iff \exists w_1, w_2 \in \Sigma^* \ \exists p, q', r \in \mathbb{N} \text{ tel que}$$

$$u = (w_1 w_2)^p w_1, \ c'' = w_2(w_1 w_2)^{q'} \text{ et } w = (w_1 w_2)^r.$$

donc $v = c''w = w_2(w_1 w_2)^{q'}(w_1 w_2)^r = w_2(w_1 w_2)^{q'+r}$ et dans ce cas la conclusion est démontrée avec $q = q' + r$.

Cas $|u| < |w|, \ |v| < |w|, \ |uv| > |w|$: figure 1.4

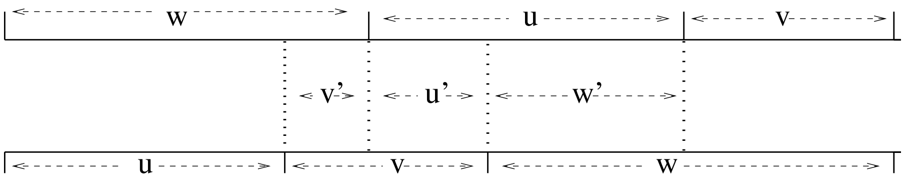


FIGURE 1.4 – cas $|u| < |w|,\ |v| < |w|$ et $|uv| > |w|$

1.7 Homomorphisme et code

Homomorphisme

L_1 et L_2 étant deux langages fermés pour la concaténation , ϕ est un homomorphisme de Ł_1^* dans Ł_2^* si et seulement si
- ϕ est une application totale de L_1 dans L_2
- $\forall m', m'' \in \text{Ł}_1\ :\ \phi(m'm'') = \phi(m')\phi(m'')$

Remarque

si ϕ est un homomorphisme de Ł_1^* dans Ł_2^* et si $\epsilon \in L_1$, alors $\phi(\epsilon) = \epsilon$: en effet
$\phi(m\epsilon) = \phi(m)\phi(\epsilon) = \phi(m)$.

spécification d'un homomorphisme

Pour connaître ϕ, il suffit de connaître l'image par ϕ de tous les mots premiers de L_1.

Code

Une partie A de Σ^* est un *code* si tout mot de A^* admet une décomposition unique en éléments de A. On appelle alors A^* le *monoïde libre* engendré par A. Tout alphabet est évidemment un code.
Par exemple $\{abb,\ bba,\ aa,\ ab,\ ba\}$ n'est pas un code (le mot *abbaabba* admet deux décompositions), $\{ab,\ abb\}$ est évidement un code (par récurrence sur la longueur du mot).

Chapitre 2

Grammaires.

2.1 Définitions relatives aux grammaires

2.1.1 Règle de réécriture

Définition

Sur un alphabet Σ, une règle de réécriture est un couple ordonné de deux mots de Σ^*.
On séparera les deux éléments du couple par le symbole $\rightarrow$.

Exemple : voici un exemple d'un ensemble de règles de réécritures
{Debout → Vautré , dans → devant , cuisine → télé , faire → regarder ,
la → le , vaisselle → match }

Utilisation d'une règle de réécriture

Quand on dispose d'une règle de réécriture $\omega_0 \rightarrow \omega_1$ et d'un mot contenant ω_0, $m'\omega_0 m''$ on peut réécrire ce mot, réécriture qui se note $m'\omega_0 m'' \rightarrow m'\omega_1 m''$.

suite de l'exemple
En utilisant la règle : faire → regarder
Debout_dans_la_cuisine_à_faire_la_vaisselle → Debout_dans_la_cuisine_à_regarder_la_vaisselle

Utilisation d'un ensemble de règles de réécriture

En partant d'un mot initial, on peut appliquer successivement plusieurs règles de réécriture (ou plusieurs fois la même règle). On notera $\xrightarrow{*}$ cette opération.

fin de l'exemple
Debout_dans_la_cuisine_à_faire_la_vaisselle $\xrightarrow{*}$ Vautré_devant_la_télé_à_regarder_le_match

2.1.2 Production

Une production est une règle de réécriture dont la partie gauche est une lettre (formellement une production est un élément du produit cartésien $\Sigma \times \Sigma^*$).

Exemple de productions :
$S \to aSb \quad S \to SS \quad S \to \epsilon$

Exemple et contre exemple de productions :

exemples :
$S \to ABCS \quad S \to \epsilon$
$A \to a$
$B \to b$
$C \to c$
Jusque là, ça va et on peut générer tous les mots de $(abc)^*$.

Contre exemple :
les mêmes règles et
$AB \to BA$, $BA \to AB$, $AC \to CA$, $CA \to AC$, $BC \to CB$, $CB \to BC$
Ce ne sont plus des règles de production, mais on peut maintenant générer tous les mots de $\{m \in \{a,\ b,\ c\}^* \mid |m|_a = |m|_b = |m|_c\}$.

2.1.3 Grammaires algébriques

Les grammaires algébriques sont des objets permettant de générer des langages en utilisant des règles de production.

Définition

Formellement une grammaire algébrique est un quadruplet $< \Sigma, X, P, S >$ où :
- Σ est l'alphabet d'entrée
- X est un alphabet auxiliaire ($X \cap \Sigma = \emptyset$)
- $S \in X$ est une lettre de X appelée *axiome*
- P est un ensemble de productions $\alpha \to \beta$ avec $\alpha \in X,\ \beta \in (\Sigma \cup X)^*$

Exemple : $G\ = < \Sigma,\ X,\ P,\ S >$
$\Sigma = \{a,\ b,\ c,\ d\}, X = \{S,\ T,\ U\}, S$ est l'axiome et l'ensemble P de règles est :
$\{S \to TU,\ T \to aTb,\ T \to \epsilon,\ U \to cUd,\ U \to \epsilon\}$ que l'on dénote :
$\{S \to TU,\ T \to aTb|\epsilon,\ U \to cUd|\epsilon\}$.

Utilisation : les productions permettent de réécrire des mots de $(X \cup \Sigma)^*$. Formellement l'application d'une production $\to$ est définie par : si $\alpha \to \beta$ est une règle de la grammaire alors $\forall \gamma_1, \gamma_2 \in (X \cup \Sigma)^*$ on a $\gamma_1 \alpha \gamma_2 \to \gamma_1 \beta \gamma_2$.
On dira que $\gamma_1 \beta \gamma_2$ est une dérivation autorisée de $\gamma_1 \alpha \gamma_2$.

Pour l'exemple ci dessus : $aTbU$ est une dérivation autorisée de TU.

Définition d'une chaîne de dérivations et de sa longueur

Pour une grammaire $G = < \Sigma,\ X,\ P,\ S >$, une chaîne de dérivations de ω_0 en ω_n est une suite de mots $(\omega_0\ ,\ \omega_1\ , \dots \omega_n)$ telle que
$\forall i \in [0 \dots n-1]\ :\ \omega_i \rightarrow \omega_{i+1}$ soit une dérivation autorisée.
n est alors la longueur de la chaîne de dérivations et l'on note $\omega_0 \xrightarrow{n} \omega_n$ le fait que ω_0 dérive en ω_n par une chaîne de dérivations de longueur n.

Pour l'exemple ci dessus : $TU \rightarrow aTbU \rightarrow a\epsilon bU = abU \rightarrow abcUd \rightarrow abc\epsilon d = abcd$ est une chaîne de dérivations de longueur 4.

Définition de $\xrightarrow{*}$

$\forall \omega', \omega'' \in (X \cup \Sigma)^*\ :\ \omega' \xrightarrow{*} \omega'' \iff$ il existe une chaîne de dérivations de ω' en ω'' (peu importe la longueur de cette chaîne [1]).

Pour l'exemple ci dessus : $S \xrightarrow{*} aabbcccddd$.

langage L_G engendré par une grammaire G

L_G est l'ensemble des mots $m \in \Sigma^*$ tels que $S \xrightarrow{*} m$ (on dit qu'un tel mot m est généré par la grammaire G).
On dit qu'un langage L est **algébrique** si et seulement si il existe une grammaire algébrique G telle que $L = L_G$.

Pour l'exemple ci dessus : $L_G = \{a^n b^n\ c^p d^p |\ n, p \in \mathbb{N}\}$

langage élargi $\widehat{L_G}$ engendré par une grammaire G

$\widehat{L_G}$ est l'ensemble des mots $m \in (\Sigma \cup X)^*$ tels que $S \xrightarrow{*} m$.

Pour l'exemple ci dessus : $aSbcTd \in \widehat{L_G}$

2.2 Lemme fondamental sur les dérivations

2.2.1 Enoncé

si $u_1 u_2 \xrightarrow{k} v$, alors $\exists v_1, k_1, v_2, k_2$ tels que

$$v = v_1 v_2,\ k = k_1 + k_2 \text{ et } u_1 \xrightarrow{k_1} v_1 \text{ et } u_2 \xrightarrow{k_2} v_2$$

1. en particulier $\omega \xrightarrow{*} \omega$ car $\omega \xrightarrow{0} \omega$.

2.2.2 Démonstration

par récurrence sur k

base ordre 0 : $k = 0$ trivial

base ordre 1 : k=1

- u_1u_2 dérive signifie que u_1u_2 contient (au moins) un non terminal, celui qui est la partie gauche de la règle de production appliquée ;
- notons N ce non terminal, $\exists u',\ u$” tel que $u_1u_2 = u'Nu''$ (éventuellement u' ou u'' peut être vide).
- soit $N \rightarrow m$ la règle de la grammaire utilisée, on a $v = u'mu''$

Deux cas sont à envisager suivant que N est dans u_2 ou dans u_1 :

- $|u'| \geq |u_1|$ (N est dans u_2) : fig.2.1
  On peut alors écrire : $u' = u_1t$ et $u_2 = tNu''$.

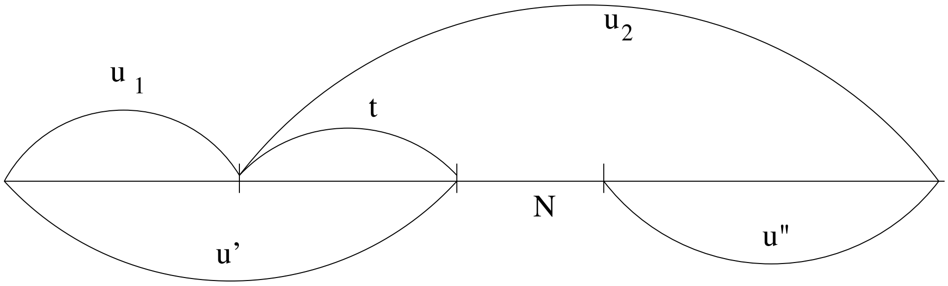


Figure 2.1 – $|u'| \geq |u_1|$

En posant $v_1 = u_1$ et $v_2 = tmu''$ on a bien $u_1 \xrightarrow{0} v_1$, $u_2 \xrightarrow{1} v_2$ et $v = v_1v_2$ [2].

- $|u'| < |u_1|$ (N est dans u_1) : fig.2.2.
  On a symétriquement $u_1 = u'Nt$ et $u'' = tu_2$.

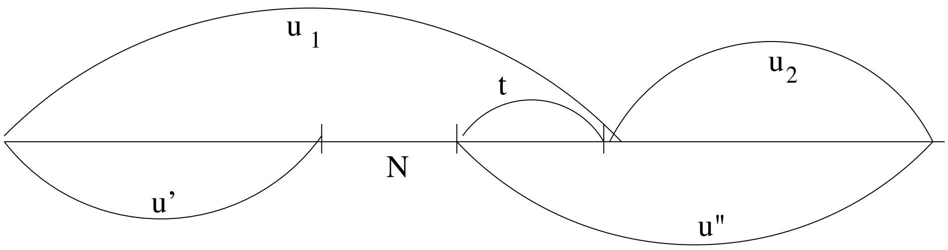


Figure 2.2 – $|u'| \geq |u_1|$

En posant $v_1 = u'mt$ et $v_2 = u_2$ on a bien $u_1 \xrightarrow{1} v_1$, $u_2 \xrightarrow{0} v_2$ et $v = v_1v_2$.

2. et $1 = 1 + 0$.

Récurrence Supposons le lemme vrai pour toute chaîne de dérivations de longueur inférieure ou égale à k ($k \geq 1$) et montrons qu'il est encore vrai pour une chaîne de dérivations de longueur $k+1$:
$u_1u_2 \overset{k+1}{\to} v$ autrement dit $\exists w$ tel que $u_1u_2 \overset{k}{\to} w \overset{1}{\to} v$
D'après l'hypothèse de récurrence,
$w = w_1w_2$ avec $u_1 \overset{k_1}{\to} w_1$, $u_2 \overset{k_2}{\to} w_2$, $k_1 + k_2 = k$ et de plus $w_1w_2 \to v$.
D'après la démonstration faite pour l'ordre 1, $v = v_1v_2$ avec $w_1 \overset{l_1}{\to} v_1$, $w_2 \overset{l_2}{\to} v_2$ et $l_1 + l_2 = 1$.
Donc $u_1 \overset{k_1+l_1}{\to} v_1$, $u_2 \overset{k_2+l_2}{\to} v_2$ et $k_1 + l_1 + k_2 + l_2 = k+1$.

2.2.3 Réciproque

si $u_1 \overset{k_1}{\to} v_1$ et $u_2 \overset{k_2}{\to} v_2$ alors $u_1u_2 \overset{k_1+k_2}{\to} v_1v_2$ (définition même d'une dérivation).

2.2.4 Utilisation : démonstration classique sur les langages engendrés par une grammaire

Enoncé

Soit L_G le langage engendré par la grammaire
$G =< \{a,b\},\ \{S\},\ S,\ \{S \to aSb \mid \epsilon\} >$ et $L_P = \{a^nb^n\ ;\ n \in \mathbb{N}\}$.
On veut démontrer que $L_P = L_G$.
Pour démontrer l'égalité de deux langages, il faut démontrer une **double inclusion** : que tout mot de L_G est un mot de L_P et que tout mot de L_P est un mot de L_G.

$L_G \subseteq L_P$

Pour montrer qu'un mot d'un langage (ici L_G) défini par une grammaire vérifie une propriété (ici $\in L_P$), on procédera toujours par récurrence sur **la longueur de la chaîne de dérivations**.
Soit $\Pi(d)$ la propriété : tout mot m de L_G obtenu en d dérivations est dans L_P.

- pour $d = 0$ cette propriété est vraie *par vacuité* : aucun mot de L_G ne peut être obtenu en 0 dérivation, donc on peut dire n'importe quoi de tout mot m de L_G obtenu en 0 dérivation.
- pour $d = 1$, la propriété est triviale, car ϵ est le seul mot obtenu en une dérivation.
- Prouvons $\forall d \in \mathbb{N}\ : \Pi(d) \Rightarrow \Pi(d+1)$.
  Soit m un mot obtenu en $d+1$ dérivations ($d > 0$) : $S \overset{d+1}{\to} m$, isolons la première dérivation : $S \to aSb \overset{d}{\to} m = am'b$ [3] avec $S \overset{d}{\to} m'$. On peut alors appliquer l'hypothèse de récurrence à m' donc il existe p tel que $m' = a^pb^p$ donc $m = a^{p+1}b^{p+1}$ donc $m \in L_P$.

3. d'après le lemme fondamental appliqué deux fois : une première fois avec $u'_1 = a$, $u'_2 = Sb$, $aSb = u'_1u'_2 \overset{*}{\to} m = m'_1m'_2$ avec $a \overset{*}{\to} m'_1 = a$ et $Sb \overset{*}{\to} m'_2$ et une deuxième fois avec $u''_1 = S$, $u''_2 = b$, $u''_1u''_2 \overset{*}{\to} m'_2 = m'm''_2$ avec $S \overset{*}{\to} m'$ et $b \overset{*}{\to} m''_2 = b$ donc $m = am'b$.

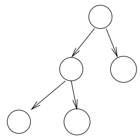


FIGURE 2.3 – Arbre ordonné

$L_P \subseteq L_G$

On procède par récurrence sur la longueur du mot.
Soit $\Pi(n)$ la propriété $a^n b^n \in L_G$.

- $\Pi(0)$ est vrai car la dérivation $S \rightarrow \epsilon$ prouve que $\epsilon \in L_G$.
- Il faut prouver que $\Pi(n) \Rightarrow \Pi(n+1)$, autrement dit que $(S \xrightarrow{*} a^n b^n) \Rightarrow (S \xrightarrow{*} a^{n+1} b^{n+1})$, ce qui est évident : $S \rightarrow aSb \xrightarrow{*} aa^n b^n b$.

Conclusion

On a vu au chapitre précédent que le langage ci dessus n'est pas rgulier. Donc il existe des langages algébriques qui ne sont pas réguliers. Autrement dit les automates à pile sont **plus puissants** que les automates d'états.

2.3 Arbres de dérivations

2.3.1 Définitions préalables

suite sans doublon sur un ensemble fini X

C'est une suite $f_1,\ f_2 \dots f_p$ où $\forall i \in [1 \dots p]\ f_i \in X$ (suite sur X) et où $\forall i, j \in [1 \dots p]\ (i \neq j) \Rightarrow (f_i \neq f_j)$ (suite sans doublon).

Dessin et définition d'un arbre ordonné

Le dessin de la figure 2.3 est celui d'un arbre ordonné.
Un arbre sur un ensemble fini X (appelé ensemble de *sommets*) est un couple $(X,\ F)$ où F est une fonction de X dans l'ensemble des suites sans doublon sur X, fonction telle que

- $\forall x \in X\ x \notin F(x)$
- $\exists!$ [4] $r \in X$ tel que $\forall y \in X - \{r\}\ \exists! z \in X$ tel que $y \in F(z)$

Pour $x \in X,\ F(x)$ est appelé la suite des *fils* de X.
r est appelé la *racine*.
Pour $x \neq r$ l'unique z tel que $x \in F(z)$ est appelé le *père* de x.
Le dessin de la figure 2.4 n'est pas celui d'un arbre car le sommet en bas a deux pères.
On appelle *feuille* de l'arbre les sommets f tels que $F(f) = \emptyset$ et *interne* les sommets qui ne sont pas des feuilles.

4. $\exists!$: il existe un et un seul.

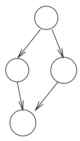


FIGURE 2.4 – pas un arbre

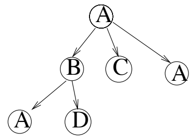


FIGURE 2.5 – Arbre étiqueté sur l'ensemble $\{A,\ B,\ C,\ D\}$

Étiquetage d'un ensemble X sur un ensemble d'étiquettes E

C'est une application (fonction totale) de X sur E.

Dessin et définition d'un arbre ordonné étiqueté

Un arbre ordonné étiqueté est un arbre ordonné dont l'ensemble des sommets est étiqueté. Le dessin de la figure 2.5 est celui d'un arbre étiqueté sur $\{A,\ B,\ C,\ D\}$.
Dans l'arbre de la figure 2.5, la racine est le sommet du haut étiqueté A, et la suite ordonnée des étiquettes des fils de cette racine est $(B,\ C,\ A)$.

2.3.2 Définition d'un arbre de dérivations

On appelle arbre de dérivations d'une grammaire $G =< \Sigma, X, P, S >$ tout arbre ordonné étiqueté sur $\Sigma \cup X \cup \{\epsilon\}$ tel que
- l'étiquette de la racine de l'arbre est l'axiome S de la grammaire
- les feuilles sont étiquetées sur $\Sigma \cup \{\epsilon\}$
- les noeuds internes sont étiquetées sur X
- $(X_1,\ X_2, \dots X_n)$ est la liste ordonnée des étiquettes des sommets de la suite des fils d'un noeud étiqueté X seulement si $X \to X_1\ X_2 \dots X_n$ est une production de P.

La lecture "de gauche à droite" des étiquettes des feuilles de l'arbre reconstitue le mot que l'arbre représente.

Exemple

Soit la grammaire ayant S pour axiome et pour règles de production
$S \to aSbS \mid \epsilon$, la figure 2.6 donne un arbre de dérivations de $aa\epsilon b\epsilon ba\epsilon b\epsilon = aabbab$

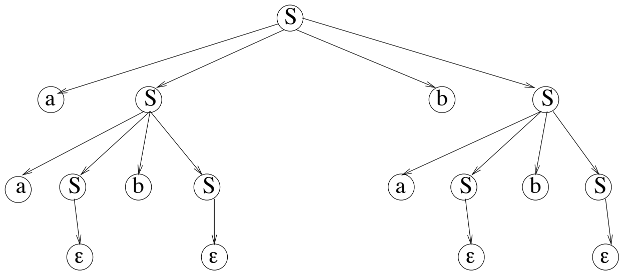


FIGURE 2.6 – Arbre de dérivation de *aabbab*

2.3.3 Chaîne de dérivations et arbre de dérivations

A toute chaîne de dérivations correspond un unique arbre de dérivations, mais à un arbre de dérivations peuvent correspondre plusieurs chaînes de dérivations.

Exemple : l'arbre de la figure 2.6 est l'unique arbre correspondant aux chaînes [5] :

1. $\mathbf{S} \to a\mathbf{S}bS \to aa\mathbf{S}bSbS \to aa\epsilon bSbS = aab\mathbf{S}bS \to aab\epsilon bS = aabb\mathbf{S} \to aabba\mathbf{S}bS \to aabba\epsilon bS = aabbab\mathbf{S} \to aabbab\epsilon = aabbab$
2. $\mathbf{S} \to aSb\mathbf{S} \to aa\mathbf{S}bSbS \to aa\epsilon bSbS = aab\mathbf{S}bS \to aab\epsilon bS = aabb\mathbf{S} \to aabbaSb\mathbf{S} \to aabbaSb\epsilon = aabba\mathbf{S}b \to aabba\epsilon b = aabbab$
3. $\mathbf{S} \to a\mathbf{S}bS \to aa\mathbf{S}bSbS \to aa\epsilon bSbS = aabSb\mathbf{S} \to aab\mathbf{S}baSbS \to aab\epsilon baSbS = aabba\mathbf{S}bS \to aabba\epsilon bS = aabbab\mathbf{S} \to aabbab\epsilon = aabbab$

• • •

80. $\mathbf{S} \to aSb\mathbf{S} \to aSbaSb\mathbf{S} \to aSbaSb\epsilon = aSba\mathbf{S}b \to aSba\epsilon b = a\mathbf{S}bab \to aaSb\mathbf{S}bab \to aaSb\epsilon bab = aa\mathbf{S}bbab \to aa\epsilon bbab = aabbab$

Conclusion

Une fois une grammaire G donnée, on peut représenter la dérivation d'un mot de $L(G)$ à partir de l'axiome à l'aide d'un arbre de dérivation. Cette représentation fait abstraction de l'ordre d'application des règles de la grammaire, contrairement à la représentation par chaînes de dérivation.

2.3.4 Dérivation gauche

Définition

Une dérivation gauche est une chaîne de dérivations ($S = \omega_0 \to \omega_1 \to \ldots \to \omega_n$) telle que $\forall i \in [0 \ldots n-1]$ le non terminal dérivé dans la dérivation $\omega_i \to \omega_{i+1}$ est le non terminal le plus à gauche de ω_i.

5. à chaque dérivation, le symbole non terminal dérivé est indiqué en gras.

Théorème

A toute chaîne de dérivations, donc à tout arbre de dérivation correspond une unique dérivation à gauche.
Il suffit de parcourir cet arbre de dérivation en suivant les branches le plus loin possible vers le bas, ensuite seulement de la gauche vers la droite ; pendant ce parcours, en repérant le premier passage à chaque sommet de l'arbre de dérivation, on obtient une suite de dérivations à gauche.
Cette façon de parcourir un arbre s'appelle "en profondeur d'abord" : on descend toujours autant que possible, en commençant à gauche ; on ne remonte que le minimum nécessaire pour pouvoir redescendre sur une branche située plus à droite.

2.4 Ambigüité

2.4.1 Définition d'une grammaire ambigüe

Une grammaire est dite ambigüe s'il existe au moins un mot de L(G) ayant plusieurs arbres de dérivations différents.

exemple : la grammaire $G =< \{a, b\},\ \{S\},\ S,\ \{S \rightarrow aSbS \mid aSb \mid SS \mid \epsilon\} >$ est ambiguë car on peut par exemple obtenir le mot *abab* par deux arbres de dérivations différents, comme le montre la figure 2.7

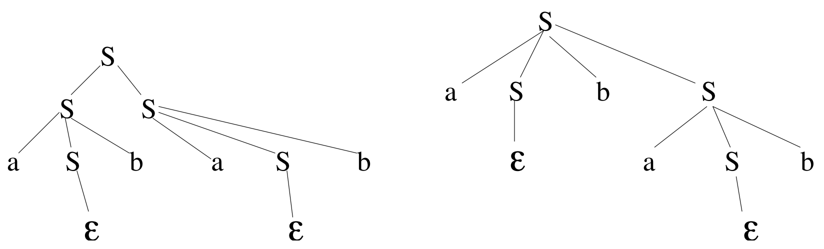


FIGURE 2.7 – deux arbres de dérivations donnant le même mot *abab*.

2.4.2 Remarques

Le terme "ambigüe" est appliqué à la grammaire et non au langage : il est souvent possible de transformer une grammaire ambigüe en une grammaire non ambigüe générant le même langage. Cependant, il existe des langages pour lesquels il n'existe pas de grammaire non ambigüe ; de tels langages sont dits intrinsèquement ambigüs. La propriété d'ambigüité est indécidable : cela signifie qu'il n'existe pas - et ne peut pas exister - d'algorithme général qui, étant donnée une grammaire algébrique, puisse toujours déterminer en un temps fini si la grammaire est ambigüe ou non. Seules peuvent être déterminées des conditions suffisantes assurant la non-ambigüité.

Chapitre 3

Automates

3.1 Automates déterministes

3.1.1 L'objet lui-même

Définition

Un **automate déterministe d'états fini** (AFD) $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ est un quintuplet où :

- Σ est l'alphabet d'entrée
- E est un ensemble fini dont les éléments sont appelés *états*
- $i \in E$ est l'état de *départ*
- $F \subseteq E$ est l'ensemble des états *d'arrivée*
- δ est la *fonction de transition* de $E \times \Sigma$ dans E.

02AAExemple.odp

Représentation

On représente un AFD $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ par un *graphe* étiqueté tel que :

- les sommets sont les états de E
- à chaque résultat $e_2 = \delta(e_1, a)$ de la fonction de transition δ correspond un arc du sommet e_1 vers le sommet e_2, arc étiqueté par a ; pour éviter de dessiner trop d'arcs (et d'avoir un multi-graphe), on s'autorise à étiqueter les arcs par des ensembles de symboles[1].
- le sommet associé à i est repéré par une flèche entrante et ceux associés aux états de F sont marqués d'un double cercle.

Exemples

premier exemple : un seul état de sortie

1. l'arc e_1e_2 est étiqueté par a et par b si et seulement si $\delta(e_1, a) = \delta(e_1, b) = e_2$

représentation : figure 3.1

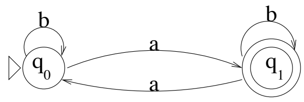


FIGURE 3.1 – Reconnaît les mots ayant un nombre impair de a

Définition formelle $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$ avec
- $\Sigma = \{a,\ b\}$
- $E = \{q_0,\ q_1\}$
- $i = q_0$
- $F = \{q_1\}$
- $\delta(q_0, b) = q_0,\ \delta(q_0, a) = q_1\ ,\ \delta(q_1, b) = q_1,\ \delta(q_1, a) = q_0$

deuxième exemple : plusieurs états de sortie : figure 3.2

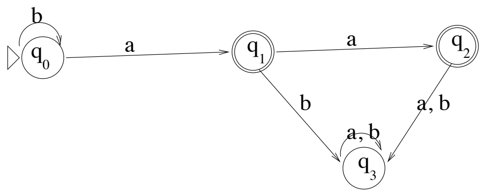


FIGURE 3.2 – reconnaît a et aa

troisième exemple : plusieurs transitions entre deux mêmes état

représentation : figure 3.3

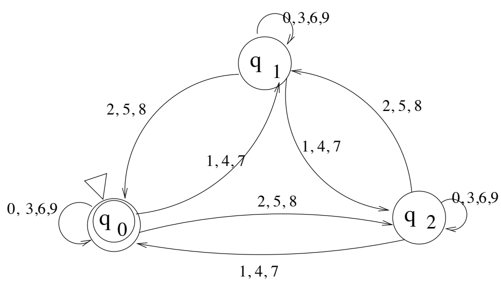


FIGURE 3.3 – Reconnaît les nombres divisibles par 3 écrits en base 10

Définition formelle $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$ avec
- $\Sigma = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9\}$
- $E = \{q_0,\ q_1,\ q_2\}$
- $i = q_0$
- $F = \{q_0\}$
- $\delta(q_0, 0) = \delta(q_0, 3) = \delta(q_0, 6) = \delta(q_0, 9) = \delta(q_1, 2) = \delta(q_1, 5) = \delta(q_1, 8) = \delta(q_2, 1) = \delta(q_2, 4) = \delta(q_2, 7) = q_0$
  $\delta(q_1, 0) = \delta(q_1, 3) = \delta(q_1, 6) = \delta(q_1, 9) = \delta(q_2, 2) = \delta(q_2, 5) = \delta(q_2, 8) = \delta(q_3, 1) = \delta(q_3, 4) = \delta(q_3, 7) = q_1$
  $\delta(q_2, 0) = \delta(q_2, 3) = \delta(q_2, 6) = \delta(q_2, 9) = \delta(q_0, 2) = \delta(q_0, 5) = \delta(q_0, 8) = \delta(q_1, 1) = \delta(q_1, 4) = \delta(q_1, 7) = q_2$

Chemin et trace

Soit $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ un automate déterministe d'états fini.

Un **chemin** est une séquence de la forme $(e_0, a_0, e_1, a_1, \ldots, e_{k-1}, a_{k-1}, e_k)$ avec $k \geq 0$, où les e_i sont des états, les a_i des lettres de Σ, et

$$\forall i \in [0, k-1]\ ,\ \delta(e_i, a_i) = e_{i+1}.$$

On dit que $(e_0, a_0, e_1, a_1, \ldots, e_{k-1}, a_{k-1}, e_k)$ est un chemin de longueur k entre les états e_0 et e_k, qui a pour **trace** le mot de Σ^* : $a_0 a_1 \ldots a_{k-1}$.
De plus pour tout état e, (e) est un chemin de longueur 0 entre e et e dont la trace est ϵ.

Justification de la légende de la figure 3.1 : La trace d'un chemin issu de q_0 et finissant en q_0 contient un nombre pair de a, celle d'un chemin issu de q_0 et finissant en q_1 contient un nombre impair de a (et réciproquement un chemin issu de q_0 et dont la trace contient un nombre pair *(resp. impair)*de a finit en q_0 *(resp. q_1)*).

Justification de la légende de la figure 3.3
Le numéro de l'état extrémité de n'importe quel chemin issu de q_0 indique le reste modulo 3 de la trace de ce chemin considérée comme l'écriture en base 10 d'un entier

Mots acceptés et langages reconnus

Un mot m est **accepté** par un automate $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ si et seulement si il existe dans cet automate un chemin de trace m de i à un état $f \in F$.
Le langage **reconnu** par un automate déterministe (complet ou non) $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$, langage noté $L(\mathcal{A})$, est l'ensemble des mots de Σ^* qui sont acceptés par $\mathcal{A}$.

Automates déterministes complets et non complets : définitions et exemples

Définition Un automate déterministe $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$ est complet si et seulement si δ est une application (fonction définie partout) de $E \times \Sigma$ dans E. Il n'est pas complet si et seulement si il existe au moins un couple $(e,\ \alpha)$ tel que $\delta(e,\ \alpha)$ ne soit pas défini.

exemple d'automate non complet : figure 3.4

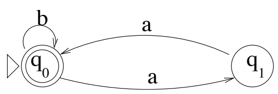


FIGURE 3.4 – Il n'y a pas de transition étiquetée b issue de q_1

exemple d'automate complet : figure 3.5

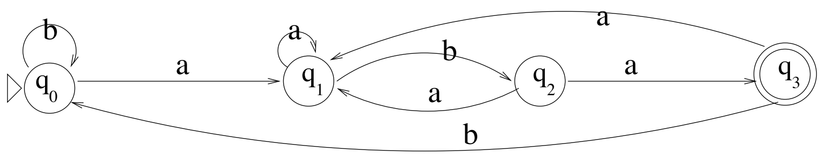


FIGURE 3.5 – Automate complet

3.1.2 Automates déterministes complets

fonction de transition itérée d'un automate déterministe complet

COMPLET

définition intuitive : Soit un automate déterministe complet $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$, sa fonction de transition itérée est une **application** δ^* de $E \times \Sigma^* \longrightarrow E$. Intuitivement, $\delta^*(e, m)$ sera l'état accessible par un chemin d'origine e et de trace m, ce que l'on définit formellement par récurrence sur $|m|$.

définition formelle :
$\forall e \in E\ \delta^*(e, \epsilon)\ = \{e\}$
$\forall e \in E,\ \forall \alpha \in \Sigma,\ \forall m \in \Sigma^*\ :\ \delta^*(e, \alpha m) = \delta^*(\delta(e, \alpha), m)$

exemple Par exemple, pour l'automate de la figure 3.6
$\delta^*(q_0, ab) = q_2$, $\delta^*(q_0, ba) = q_3$, $\delta^*(q_1, bab) = q_2$ et $\delta^*(q_1, bba) = q_3$.

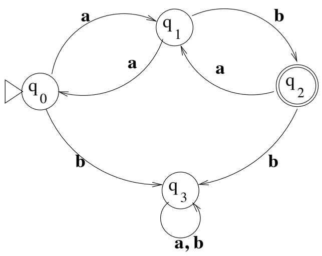


FIGURE 3.6 – automate déterministe complet dont le langage reconnu est moins évident

Lemme : $\delta^*(e,\alpha) = \delta(e,\alpha)$
$\delta(e,\alpha) = \delta^*(\delta(e,\alpha),\epsilon)$ par définition de $\delta^*(/,\epsilon)$
$\delta^*(\delta(e,\alpha),\epsilon) = \delta^*(e,\alpha\epsilon)$ par définition de $\delta^*(/,\alpha m)$
$\delta^*(e,\alpha\epsilon) = \delta^*(e,\alpha)$ C.Q.F.D.

Théorème : $\delta^*(e,m\alpha) = \delta(\delta^*(e,m),\alpha)$
se démontre par récurrence sur la longueur de m.

propriété $\Pi(l)$
$\forall m \in \Sigma^* \;:\; \{[|m| \leq l] \Rightarrow [\forall e \in E \;,\; \forall \alpha \in \Sigma \;:\; \delta^*(e,m\alpha) = \delta(\delta^*(e,m),\alpha)]\}$

$\Pi(0)$ Il faut prouver que $\forall e \in E \;,\; \forall \alpha \in \Sigma \;:\; \delta^*(e,\alpha) = \delta(\delta^*(e,\epsilon),\alpha)]\}$
Mais $\delta^*(e,\epsilon) = e$ et on a vu dans le lème que $\delta^*(e,\alpha) = \delta(e,\alpha)$. C.Q.F.D.

$\forall l \in \mathbb{N} \;:\; \Pi(l) \Rightarrow \Pi(l+1)$
$\delta^*(e,\beta m\alpha) \overset{def.\ \delta^*}{=} \delta^*(\delta(e,\beta),m\alpha) \overset{hyp.\ rec.}{=} \delta(\delta^*(\delta(e,\beta),m),\alpha)$
mais $\delta^*(\delta(e,\beta),m) \overset{def.\ \delta^*}{=} \delta^*(e,\beta m)$
donc $\delta^*(e,\beta m\alpha) = \delta(\delta^*(e,\beta m),\alpha)$ C.Q.F.D.

définition formelle d'un mot accepté :
un mot m est accepté par $\mathcal{A}$ si et seulement si $\delta^*(i,m) \in F$.

Algorithme d'acceptation d'un mot par un automate déterministe complet

algorithme

Entrées : un automate $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$ déterministe complet
un mot m_0
Sorties : un booléen indiquant si $\mathcal{A}$ accepte ou non m

$e \leftarrow i$; $m \leftarrow m_0$; $p \leftarrow \epsilon$;
tant que $m \neq \epsilon$ **faire**
  $\alpha \leftarrow$ première lettre de m ; $m \leftarrow$ suffixe sauf première lettre de m;
  $e \leftarrow \delta(e, \alpha)$; $p \leftarrow p.\alpha$;
fin
retourner $e \in F$

complexité
proportionnelle à $|m_0|$ (les opérateurs élémentaires sont en temps constant).

invariant : p est le préfixe *déjà lu*, c'est à dire $m_0 = p.m$ et $e = \delta^*(i, p)$.

3.1.3 Automates déterministes quelconques

fonction de transition itérée : définition récursive et théorème

définition intuitive
Soit un automate déterministe (pas obligatoirement complet) $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$, sa fonction de transition itérée est une **fonction** δ^* de $E \times \Sigma^* \longrightarrow E$.
$\delta^*(e, m)$ sera éventuellement (c'est à dire quand le résultat est défini) l'état accessible par un chemin d'origine e et de trace m.

définition formelle récursive
$\forall e \in E\ \delta^*(e, \epsilon)\ = \{e\}$
$\forall e \in E,\ \forall \alpha \in \Sigma,\ \forall m \in \Sigma^*$ si $\delta(e, \alpha)$ est défini et alors si $\delta^*(\delta(e, \alpha), m)$ est défini, alors $\delta^*(e, \alpha m) = \delta^*(\delta(e, \alpha), m)$, sinon $\delta^*(e, \alpha m)$ n'est pas défini.

Exemple : pour l'automate de la figure 3.7
$\delta^*(q_0, ab) = q_2$ et $\delta^*(q_1, bab) = q_2$, mais ni $\delta^*(q_0, ba)$ ni $\delta^*(q_1, bba)$ ne sont définis.

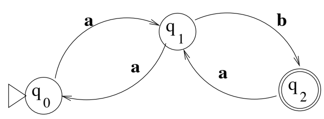


FIGURE 3.7 – automate déterministe non complet

Le théorème devient : si $\delta^*(e, m)$ n'est pas défini, alors $\delta^*(e, m\alpha)$ n'est pas défini, sinon $\delta^*(e, m\alpha) = \delta(\delta^*(e, m), \alpha)$ (éventuellement pas défini).
La démonstration est laissée en T.D.

Définition d'un mot accepté

Un mot m est accepté par $\mathcal{A}$ si et seulement si $\delta^*(i, m) \in F$ (donc si $\delta^*(i, m)$ n'est pas défini, m n'est pas accepté par $\mathcal{A}$).

Algorithme d'acceptation d'un mot par un automate déterministe quelconque

algorithme :

Entrées : un automate $\mathcal{A} = (\Sigma,\ E,\ i,\ F, \delta)$ déterministe quelconque
un mot m_0
Sorties : un booléen indiquant si $\mathcal{A}$ accepte ou non m_0

```
e ← i ; p ← ε ; m ← m0;
tant que m ≠ ε faire
|  α ← première lettre de m ; m ← suffixe sauf première lettre de m;
|  si δ(e, α) est défini alors
|   |  e ← δ(e, α) ; p ← p.α ;
|  sinon
|   |  retourner faux
|  fin
fin
retourner e ∈ F
```

complexité en $\mathcal{O}(|m_0|)$

Invariant : $e = \delta^*(i, p)$ (en particulier, $\delta^*(i, p)$ est défini).

3.2 Automates indéterministes

3.2.1 L'objet lui-même

Définition

Un **automate d'états fini** (AF) $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ est un quintuplet où :

- Σ est l'alphabet d'entrée
- E est l'ensemble des états
- $I \subseteq E$ est l'ensemble des états de départ
- $F \subseteq E$ est l'ensemble des états d'arrivée
- $\delta \subseteq E \times \Sigma \times E$ est l'ensemble des transitions.
  δ est une fonction de $E \times \Sigma$ dans $\mathcal{P}(E)$.

Représentation

Cette fois ci, tous les sommets de I sont repérés par une flèche entrante.

Exemples

un premier exemple : figure 3.8

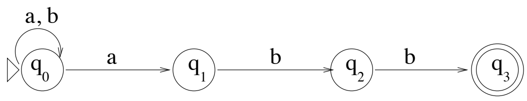


FIGURE 3.8 – Il y a deux transitions étiquetée a issues de q_0

encore un exemple : figure 3.9

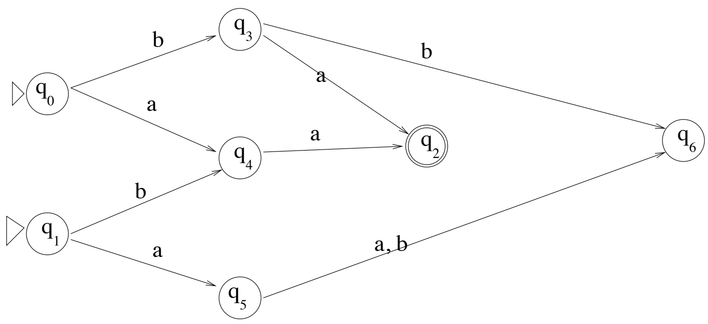


FIGURE 3.9 – Il y a deux états d'entrée (et le mot ba est accepté de deux façons)

un dernier exemple : figure 3.10

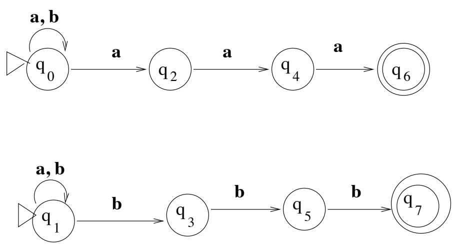


FIGURE 3.10 – Reconnaît tous les mots finissant par aaa ou par bbb.

Justification de la légende :

quand on arrive en	on vient de lire un mot qui a comme suffixe
q_2	a
q_3	b
q_4	aa
q_5	bb
q_6	aaa
q_7	bbb

Définition formelle de l'automate de la figure 3.10 $\mathcal{A} = (\Sigma,\ E,\ I,\ F, \delta)$ avec

- $\Sigma = \{a,\ b\}$
- $E = \{q_0,\ q_1,\ q_2,\ q_3,\ q_4,\ q_5,\ q_6,\ q_7\}$
- $I = \{q_0,\ q_1\}$
- $F = \{q_6,\ q_7\}$
- $\delta = \{(q_0, a, q_0)\ ,\ (q_0, b, q_0)\ ,\ (q_0, a, q_2)\ ,\ (q_2, a, q_4)\ ,\ (q_4, a, q_6)\ ,\ (q_1, a, q_1)\ ,\ (q_1, b, q_1),$
  $(q_1, b, q_3)\ ,\ (q_3, b, q_5)\ ,\ (q_5, b, q_7)\}$
- Autrement dit $\delta(q_0, a) = \{q_0, q_2\},\ \delta(q_1, b) = \{q_1, q_3\},\ \delta(q_2, a) = \{q_4\},$
  $\delta(q_2, b)$ n'est pas défini ....

Extension de δ

On étend δ sans problème en une fonction de $\mathcal{P}(E) \times \Sigma$ dans $\mathcal{P}(E)$:

$$\delta(Q, \alpha) = \bigcup_{q \in Q} \delta(q, \alpha)$$

Définition formelle chemin et trace

Les définitions de chemin et de trace restent valides pour un automate indéterministe, mais plusieurs chemins différents de même origine peuvent avoir la même trace.

Mots acceptés et langages reconnus

Un mot m est accepté par un automate indéterministe $\mathcal{A} = (\Sigma,\ E,\ I,\ F, \delta)$ si et seulement si il existe un chemin de trace m d'un état $i \in I$ jusqu'à un état $f \in F$.
Le langage $L(\mathcal{A})$ reconnu par un automate indéterministe $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Automates indéterministes complets et non complets

L'automate indéterministe $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ est complet si et seulement si $\forall e \in E,\ \forall \alpha \in \Sigma\ \exists e' \in E$ tel que $(e, \alpha, e') \in \delta$.

3.2.2 Utilisation des automates indéterministes

le problème de l'indéterminisme

Par exemple soit $\mathcal{A} = (\Sigma, E, I, F, \delta)$ un automate d'états fini avec $\Sigma = \{a\}$, $E = \{e_0, e_1, e_2\}$, $I = \{e_0\}$, $F = \{e_1\}$ et $\delta = \{(e_0, a, e_1),\ (e_0, a, e_2)\}$.
$\mathcal{A}$ accepte le mot a, même si le chemin e_0ae_2 n'est pas le chemin qui permette d'accepter le mot.

fonction de transition itérée

Soit un automate indéterministe $\mathcal{A} = (\Sigma,\ E,\ I,\ F, \delta)$, sa fonction de transition itérée δ^* est une fonction de $E \times \Sigma^* \longrightarrow \mathcal{P}(E)$.

définition intuitive
$\delta^*(e, m)$ est l'ensemble des états extrémités d'un chemin d'origine e et de trace m

définition récursive
$\forall e \in E\ :\ \delta^*(e, \epsilon)\ = \{e\}$

$$\forall e \in E,\ \forall \alpha \in \Sigma,\ \forall m \in \Sigma^*\ :\ \delta^*(e, \alpha.m) = \bigcup_{(e,\alpha,e') \in \delta} \delta^*(e', m)$$

Remarquons que si $\delta(e, \alpha) = \emptyset$, alors

$$\forall m \in \Sigma^*\ :\ \bigcup_{(e,\alpha,e') \in \delta} \delta^*(e', m) = \emptyset$$

extension de δ^* On étend cette fonction δ^* de $\mathcal{P}(E) \times \Sigma^* \longrightarrow \mathcal{P}(E)$:

$$\forall E' \subseteq E\ :\ \delta^*(E', m) = \bigcup_{e' \in E'} \delta^*(e', m)$$

exemple : figure 3.11
$\delta^*(q_0, abb) = \delta^*(\{q_1,\ q_2\}, bb) = \delta^*(\{q_3,\ q_1,\ q_4\}, b) = \{q_3\}$

le théorème reste valide (et sera démontré en T.D.) :
$\forall E' \subseteq E,\ \forall \alpha \in \Sigma,\ \forall m \in \Sigma^*\ :\ \delta^*(E', m\alpha) = \delta(\delta^*(E', m), \alpha)$

Mots acceptés et langages reconnus

un mot m est accepté par $\mathcal{A}$ si et seulement si $\delta^*(I, m) \cap F \neq \emptyset$.
par exemple, l'automate de la figure 3.12 reconnaît tout mot qui contient un facteur `ab`.

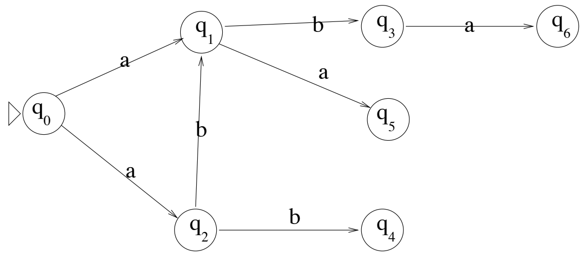


FIGURE 3.11 –

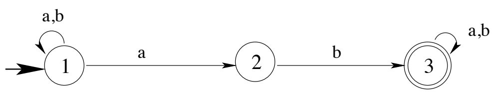


FIGURE 3.12 – automate indéterministe

le langage reconnu par un automate indéterministe $\mathcal{A} = (\Sigma,\ E,\ I,\ A, \delta)$, langage noté $L(\mathcal{A})$, est l'ensemble des mots de Σ^* qui sont acceptés par $\mathcal{A}$.

Algorithme d'acceptation d'un mot par un automate indéterministe quelconque

algorithme

Entrées : un automate $\mathcal{A} = (\Sigma,\ E,\ I,\ F, \delta)$ indéterministe quelconque
un mot m_0
Sorties : un booléen indiquant si $\mathcal{A}$ accepte ou non m_0

$E' \leftarrow I$; $m \leftarrow m_0$; $p \leftarrow \epsilon$;
tant que $m \neq \epsilon$ **faire**
  $E'' \leftarrow \emptyset$;
  $\alpha \leftarrow$ première lettre de m ; $p \leftarrow p.\alpha$;
  $m \leftarrow$ suffixe de m privé de sa première lettre;
  pour $e' \in E'$ **faire** $E'' \leftarrow E'' \cup \delta(e', \alpha)$;
  $E' \leftarrow E''$
fin
retourner $E' \cap F \neq \emptyset$

complexité : beaucoup plus compliquée !

justification Les invariants importants de la répétitive sont :
- $m_0 = pm$
- $E' = \delta^*(I, p)$

Le premier invariant est trivial le deuxième est valide car initialement $I = \delta^*(I, \epsilon)$, mais surtout à cause du théorème que nous allons démontrer ci dessous, et qui dira que $\delta^*(I, p\alpha) = \delta(\delta^*(I, p), \alpha)$.

théorème : $\delta^*(E', m\alpha) = \delta(\delta^*(E', m), \alpha)$
Par récurrence sur $l = |m|$, on pose
$\Pi(l) : |m| \leq l \Rightarrow \delta^*(E', m\alpha) = \delta(\delta^*(E', m), \alpha)$
La base ($m = \epsilon$) est triviale.
Prouvons $\Pi(l) \Rightarrow \Pi(l+1)$:
soit m un mot de longueur $l+1$ et β sa première lettre : $m = \beta m'$

$$\delta^*(E', m\alpha) = \delta^*(E', \beta m'\alpha) = \delta^*(\delta(E', \beta), m'\alpha) = \delta(\delta^*(\delta(E', \beta), m'), \alpha) =$$

$$\delta(\delta^*(E', \beta m'), \alpha) = \delta(\delta^*(E', m), \alpha)$$

3.3 Quelques suppléments utiles

3.3.1 langages rationnels

Un langage est **rationnel** (régulier, reconnaissable) si et seulement si il est reconnu par un automate d'états fini.
On note $Rec(\Sigma^*)$ l'ensemble des langages (sur Σ^*) rationnels.

3.3.2 états accessibles et co-accessibles, théorème de suppression

Un état e est **accessible** si et seulement si il existe un chemin depuis un état initial jusqu'à e.
Un état e est **co-accessible** si et seulement si il existe un chemin depuis e jusqu'à un état final.

exemples : figures 3.13 et 3.14

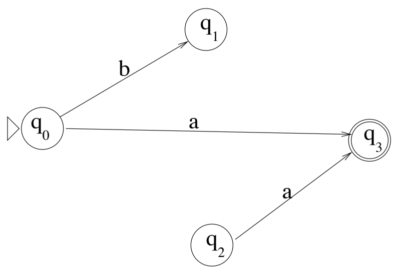


FIGURE 3.13 – q_2 n'est pas accessible et q_1 n'est pas co-accessible

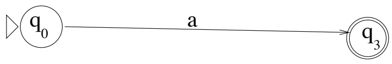


FIGURE 3.14 – q_0 et q_3 sont accessibles et co-accessibles

Théorème : suppression des états non accessibles et non co-accessibles

Soit $\mathcal{B}$ l'automate obtenu en supprimant dans l'automate $\mathcal{A}$ les états non accessibles ou non coaccessibles : $\mathcal{A}$ et $\mathcal{B}$ reconnaissent exactement le même langage.

démonstration : si m est accepté par $\mathcal{A}$ alors il existe dans $\mathcal{A}$ un chemin de trace m d'un état initial à un état final, mais alors tous les états de ce chemin sont accessibles et coaccessibles donc ce chemin est un chemin de $\mathcal{B}$ donc m est accepté par $\mathcal{B}$.
Réciproquement tout chemin de $\mathcal{B}$ est un chemin de $\mathcal{A}$, donc tout mot accepté par $\mathcal{B}$ est accepté par $\mathcal{A}$.

3.3.3 théorème de complétion de l'automate

Soit $\mathcal{B}$ l'automate obtenu en complétant l'automate (déterministe ou indéterministe) $\mathcal{A}$ (par un état *poubelle* p non coaccessible) : $\mathcal{A}$ et $\mathcal{B}$ reconnaissent exactement le même langage. La démonstration est laissée en exercice.

Entrée : un automate $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta_{\mathcal{A}})$

Sortie un automate $\mathcal{B} = (\Sigma,\ E \cup \{p\},\ I,\ F,\ \delta_{\mathcal{B}})$ où $p \notin E$ et où $\delta_{\mathcal{B}} = \delta_{\mathcal{A}} \cup \delta' \cup \delta''$ avec

$$\delta' = \bigcup_{e_1 \in E,\ \alpha \in \Sigma\ \mid\ \forall e' \in E\ :\ e_1 \alpha e' \notin \delta_{\mathcal{A}}} \{e_1 \alpha p\}$$

$$\delta'' = \bigcup_{\alpha \in \Sigma} \{p \alpha p\}$$

exemple : figure 3.15

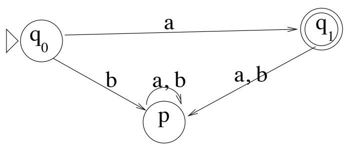


FIGURE 3.15 – Cet automate complet reconnaît le même langage que celui de la figure 3.14

Chapitre 4

ϵ-transitions

4.1 Définition des ϵ-transitions

4.1.1 Description informelle

Les transitions étiquetées ϵ permettent de changer d'états sans "consommer" de lettres du mot.

exemple : figures 4.1, 4.2 et 4.3

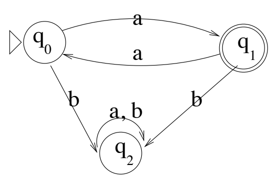


FIGURE 4.1 – reconnaît les mots sans b et avec un nombre impair de a

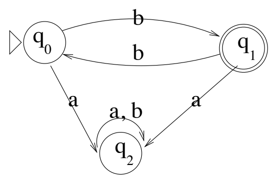


FIGURE 4.2 – reconnaît les mots sans a et avec un nombre impair de b

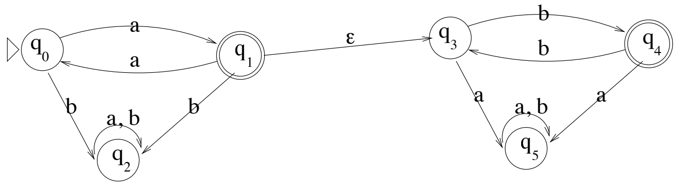


FIGURE 4.3 – reconnaît les mots composés d'un nombre impair de a suivis d'un nombre impair de b

4.1.2 Définitions formelles

Automate avec ϵ-transition

Un **automate fini** $\mathcal{A} = (\Sigma,\ E,\ I,\ F, \delta)$ reste un quintuplet où :
- Σ est l'alphabet d'entrée
- E est un ensemble fini
- $I \subseteq E$ est l'ensemble des états de départ
- $F \subseteq E$ est l'ensemble des états d'arrivée
- mais maintenant l'ensemble des transitions δ est inclus dans $E \times \Sigma \cup \{\epsilon\} \times E$

chemin et trace avec ϵ-transition

Un chemin est maintenant une séquence de la forme $(e_0, a_0, e_1, a_1, \ldots, e_{k-1}, a_{k-1}, e_k)$ avec $k \geq 0$ et où les e_i sont des états, les a_i des symboles de $\Sigma \cup \{\epsilon\}$ et $\forall i \in [0, k-1], (e_i a_i e_{i+1}) \in \delta$.
On dit que $(e_0, a_0, e_1, a_1, \ldots, e_{k-1}, a_{k-1}, e_k)$ est un chemin de longueur k entre les états e_0 et e_k, qui a pour trace le mot de Σ^* : $a_0 a_1 \ldots a_{k-1}$ (mais ϵ étant l'élément neutre de la concaténation, les occurences de ϵ disparaissent de cette trace, sauf si la dite trace est réduite à ϵ).

4.2 Automates standards

4.2.1 Définition

Un automate $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ est standard si et seulement si
- I est un singleton d'un seul état i
- F est un singleton d'un seul état f
- $\forall e \in E, \forall \alpha \in \Sigma \cup \{\epsilon\}\ :\ e\alpha i \notin \delta$ et $f\alpha e \notin \delta$.

4.2.2 Théorème d'existence

A tout automate $\mathcal{A}$, on peut faire correspondre un automate standard $\mathcal{A}_{\mathcal{S}}$ qui reconnaisse le même langage.

exemple : l'automate de la figure 4.4 et celui de la figure 4.5 reconnaissent le même langage.

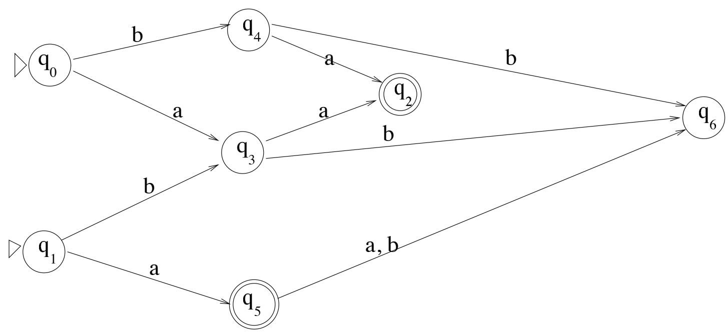


FIGURE 4.4 – reconnaît $\{ba,\ aa,\ a\}$

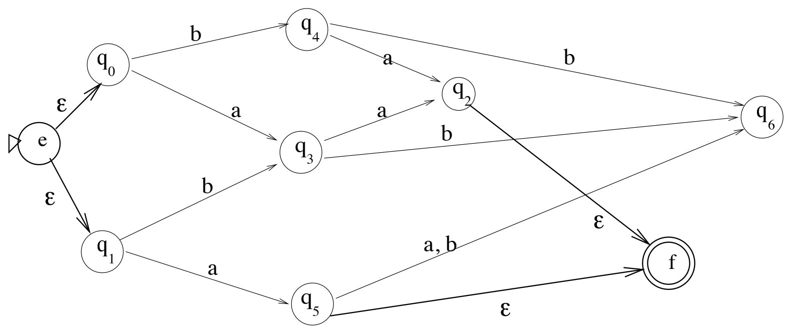


FIGURE 4.5 – reconnaît le même langage que celui de la figure 4.4

4.2.3 Transformation d'un automate quelconque en automate standard

Entrée

Un automate $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ avec δ une fonction de $E \times \Sigma \cup \{\epsilon\}$ dans $\mathcal{P}(E)$, de fonction de transition itérée δ^* de $E \times \Sigma^*$ dans $\mathcal{P}(E)$, fonction étendue en une fonction δ^* de $\mathcal{P}(E) \times \Sigma^*$ dans $\mathcal{P}(E)$.

Sortie

Un automate standard $\mathcal{A}_\mathcal{S} = (\Sigma,\ E_\mathcal{S},\ I_\mathcal{S} = \{i_f\},\ F_\mathcal{S} = \{f_f\},\ \delta_\mathcal{S})$ avec $\delta_\mathcal{S}$ une fonction de $E_S \times \Sigma \cup \{\epsilon\}$ dans $\mathcal{P}(E_S)$, de fonction de transition itérée $\delta_\mathcal{S}^*$ de $E \times \Sigma^*$ dans $\mathcal{P}(E)$, fonction étendue en une fonction $\delta_\mathcal{S}^*$ de $\mathcal{P}(E_S) \times \Sigma^*$ dans $\mathcal{P}(E_S)$.

Propriété voulue

Les langages reconnus par les automates $\mathcal{A}$ et $\mathcal{A}_\mathcal{S}$ sont les mêmes, autrement dit $\forall m \in \Sigma^*\ :\ f_\mathcal{S} \in \delta^*_\mathcal{S}(i_\mathcal{S}, m) \iff \delta^*(I, m) \cap F \neq \emptyset$.

Définition de $\mathcal{A}_\mathcal{S}$

$E_\mathcal{S} = E + \{i_\mathcal{S}, f_\mathcal{S}\}$ avec $E \cap \{i_\mathcal{S}, f_\mathcal{S}\} = \emptyset$.
$I_\mathcal{S} = \{i_\mathcal{S}\}$ et $F_\mathcal{S} = \{f_\mathcal{S}\}$
$\delta_\mathcal{S} = \delta \cup \{(i_\mathcal{S}, \epsilon, i) \mid i \in I\} \cup \{(f, \epsilon, f_\mathcal{S}) \mid f \in F\}$
$\mathcal{A}_\mathcal{S}$ est standard, d'après la définition de $\delta_\mathcal{S}$.

preuve que $L_\mathcal{A} = L_{\mathcal{A}_\mathcal{S}}$

Il est facile de voir que $\epsilon \in L_\mathcal{A} \iff \epsilon \in L_{\mathcal{A}_\mathcal{S}}$
Regardons maintenant un mot m non vide, donc tel que $m = l_1 l_2 \ldots l_n$ (avec $\forall i \in [1 \ldots n]\ :\ l_i \in \Sigma$) :

$$m \in L_\mathcal{A}$$

$\iff$ $\exists$ un chemin $e_1\alpha_1 e_2\alpha_2 \ldots e_p\alpha_p e_{p+1}$ dans $\mathcal{A}$ avec

1. $e_1 \in I$
2. $e_{p+1} \in F$ (éventuellement $p = 0$)
3. $\alpha_1\alpha_2 \ldots \alpha_p = l_1 l_2 \ldots l_n$ (mais certains α_i peuvent être ϵ).

$\iff$ $\exists$ un chemin $i_\mathcal{S}\epsilon e_1\alpha_1 e_2\alpha_2 \ldots e_p\alpha_p e_{p+1}\epsilon f_\mathcal{S}$ dans $\mathcal{A}_\mathcal{S}$

$\iff$ $m \in L_{\mathcal{A}_\mathcal{S}}$

4.3 Concaténation de langages

Soit deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$ qui reconnaissent les langages $L_{\mathcal{A}_1}$ et $L_{\mathcal{A}_2}$, et $\mathcal{A}_{1_\mathcal{S}} = (\Sigma,\ E_{1_\mathcal{S}},\ \{i_{1_\mathcal{S}}\},\ \{f_{1_\mathcal{S}}\},\ \delta_{1_\mathcal{S}})$ et $\mathcal{A}_{2_\mathcal{S}} = (\Sigma,\ E_{2_\mathcal{S}},\ \{i_{2_\mathcal{S}}\},\ \{f_{2_\mathcal{S}}\},\ \delta_{2_\mathcal{S}})$ les automates standards construit à partir de $\mathcal{A}_1$ et $\mathcal{A}_2$.
Soit $\mathcal{A} = (\Sigma,\ E_{1_\mathcal{S}} \cup E_{2_\mathcal{S}},\ \{i_{1_\mathcal{S}}\},\ \{f_{2_\mathcal{S}}\},\ \delta_{1_\mathcal{S}} \cup \delta_{2_\mathcal{S}} \cup \{f_{1_\mathcal{S}}\epsilon i_{2_\mathcal{S}}\})$ et $L_\mathcal{A}$ le langage reconnu par $\mathcal{A}$. Alors $L_\mathcal{A} = L_{\mathcal{A}_1} L_{\mathcal{A}_2}$.

4.4 Fermeture de Kleene

Soit un automate $\mathcal{A}$ qui reconnaît le langage $L_\mathcal{A}$, et $\mathcal{A}_\mathcal{S} = (\Sigma,\ E_\mathcal{S},\ \{i_\mathcal{S}\},\ \{f_\mathcal{S}\},\ \delta_\mathcal{S})$ l'automate standard construit à partir de $\mathcal{A}$.
Soit $\mathcal{A}_* = (\Sigma,\ E_\mathcal{S},\ \{i_\mathcal{S}\},\ \{i_\mathcal{S}\},\ \delta_\mathcal{S} + (f_\mathcal{S}\epsilon i_\mathcal{S}))$ et $L_{\mathcal{A}_*}$ le langage reconnu par $\mathcal{A}_*$.

propriétés de $\mathcal{A}_*$

1. $\{i_\mathcal{S}\}$ est le seul état final
2. $(f_\mathcal{S}\ \epsilon i_\mathcal{S})$ est la seule transition qui sorte de $f_\mathcal{S}$
3. c'est aussi la seule qui rentre dans $i_\mathcal{S}$

Théorème :
$L_{\mathcal{A}_*}$ est l'ensemble des mots obtenus par concaténation de 0, 1 ou plusieurs mots de $L_\mathcal{A}$.

tout mot de $L_{\mathcal{A}_*}$ est une concaténation de mots de $L_\mathcal{A}$
Soit $\mathcal{C}_m$ un chemin dans $\mathcal{A}_*$ qui prouve que $m \in L_{\mathcal{A}_*}$, donc un chemin d'origine et d'extremité $i_\mathcal{S}$.

- si $\mathcal{C}_m$ ne contient aucune occurrence de $f_\mathcal{S}$, alors $\mathcal{C}_m$ est réduit à $(i_\mathcal{S})$ (d'après la propriété 3 ci-dessus), donc $m = \epsilon$.
- si $\mathcal{C}_m$ contient une seule occurrence de $f_\mathcal{S}$, alors $\mathcal{C}_m$ privé de la dernière transition $f_\mathcal{S}\epsilon i_\mathcal{S}$ est un chemin dans $\mathcal{A}$ de $i_\mathcal{S}$ à $f_\mathcal{S}$ de trace m, ce qui prouve que $m \in L_{\mathcal{A}_*}$
- sinon, on procède par récurrence sur le nombre d'occurences de $f_\mathcal{S}$ dans $\mathcal{C}_m$:
  si $\mathcal{C}_1$ est le début du chemin jusqu'à la première occurence de $f_\mathcal{S}$ et $\mathcal{C}_2$ tel que $\mathcal{C}_m = \mathcal{C}_1\epsilon\mathcal{C}_2$ (la seule transition de $f_\mathcal{S}$ à $i_\mathcal{S}$ est étiquetée ϵ), $\mathcal{C}_2$ est un chemin de $i_\mathcal{S}$ à $i_\mathcal{S}$ et si m_1 et m_2 sont les traces de $\mathcal{C}_1$ et $\mathcal{C}_2$ $m = m_1m_2$ et par hypothèse de récurrence m_2 est une concaténation de mots de $L_\mathcal{A}$ et $m_1 \in L_\mathcal{A}$.

toute concaténation de mots de $L_\mathcal{A}$ est un mot de $L_{\mathcal{A}_*}$
soit $m = m_1m_2\ldots m_n$ avec $\forall i \in [1\ldots n]\ m_i \in L_\mathcal{A} = L_{\mathcal{A}_\mathcal{S}}$, il existe alors dans $\mathcal{A}_\mathcal{S}$ des chemins $\mathcal{C}_1,\ \mathcal{C}_2\ldots\mathcal{C}_n$ de trace $m_1, m_2\ldots m_n$ de $i_\mathcal{S}$ à $f_\mathcal{S}$ donc $\mathcal{C}_1\epsilon\mathcal{C}_2\ldots\mathcal{C}_n\epsilon i_\mathcal{S}$ prouve que $m \in L_{\mathcal{A}_*}$.

4.5 Transformation d'automate : suppression des ϵ-transitions

4.5.1 Spécifications de la transformation

Entrée

Un automate $\mathcal{A}_\epsilon = (\Sigma,\ E,\ I,\ F_\epsilon,\ \delta_\epsilon)$ avec δ_ϵ une fonction de $E \times \Sigma \cup \{\epsilon\}$ dans $\mathcal{P}(E)$, de fonction de transition itérée δ_ϵ^* de $E \times \Sigma^*$ dans $\mathcal{P}(E)$, fonction étendue en une fonction δ_ϵ^* de $\mathcal{P}(E) \times \Sigma^*$ dans $\mathcal{P}(E)$.

Sortie

Un automate $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ avec δ une fonction de $E \times \Sigma$ dans $\mathcal{P}(E)$, de fonction de transition itérée δ^* de $E \times \Sigma^*$ dans $\mathcal{P}(E)$, fonction étendue en une fonction δ^* de $\mathcal{P}(E) \times \Sigma^*$ dans $\mathcal{P}(E)$.

Propriété voulue

Les langages reconnus par les automates $\mathcal{A}_\epsilon$ et $\mathcal{A}$ sont les mêmes, autrement dit $\forall m \in \Sigma^*\ :\ \delta^*_\epsilon(I, m) \cap\ F_\epsilon = \emptyset \iff \delta^*(I, m) \cap F = \emptyset$.

4.5.2 Exemple d'application

Le problème : $\mathcal{L} = (\mathcal{L}_1 \cup \mathcal{L}_2).\mathcal{L}_4$

Soient les langages suivants, tous sur l'alphabet $\Sigma = \{a,\ b\}$:
- $\mathcal{L}_1$ est l'ensemble de tous les mots finissant par *aaa*.
- $\mathcal{L}_2$ est l'ensemble de tous les mots finissant par *bbb*.
- $\mathcal{L}_3 = \{(ab)^n \mid n \in \mathbb{N}\}$ (en particulier $\epsilon \in \mathcal{L}_4$)
- $\mathcal{L}_4 = \mathcal{L}_1 \cup \mathcal{L}_2$
- $\mathcal{L} = \mathcal{L}_4\mathcal{L}_3$

On veut construire l'automate sans ϵ-transition qui reconnaît $\mathcal{L}$.

La démarche

On va construit d'abord les automates des figures 4.6, 4.7, 4.8, 4.9, 4.10 et 4.10 :

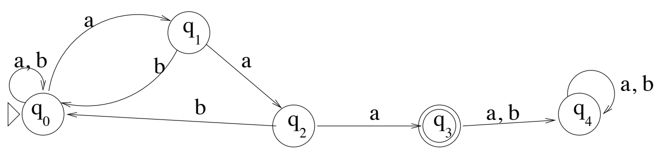


FIGURE 4.6 – Automate indéterministe complet qui reconnaît le langage $\mathcal{L}_1$ des mots finissant par *aaa*

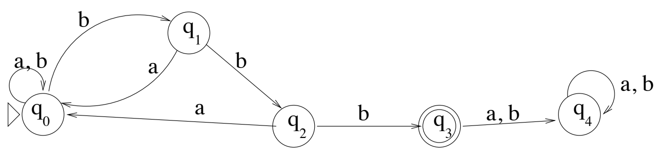


FIGURE 4.7 – Automate indéterministe complet qui reconnaît le langage $\mathcal{L}_2$ des mots finissant par *bbb*

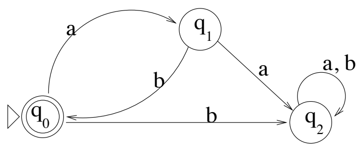


FIGURE 4.8 – Automate indéterministe qui reconnaît le langage $\mathcal{L}_3$ des mots composés de *ab* concaténés

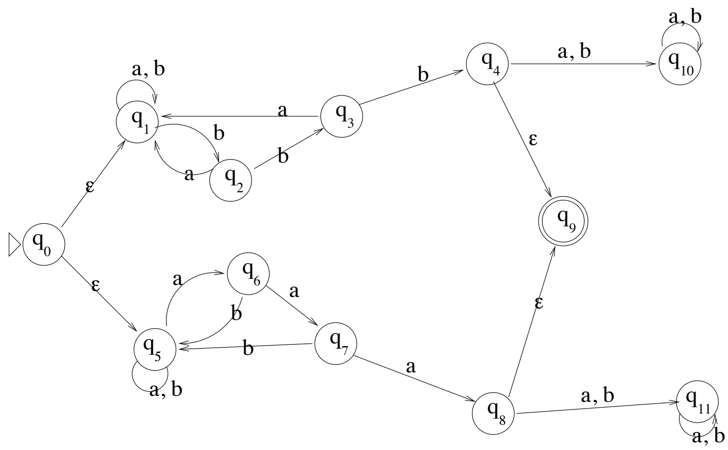


FIGURE 4.9 – Automate indéterministe qui reconnaît le langage $\mathcal{L}_4$ des mots finissant par *aaa* ou *bbb*

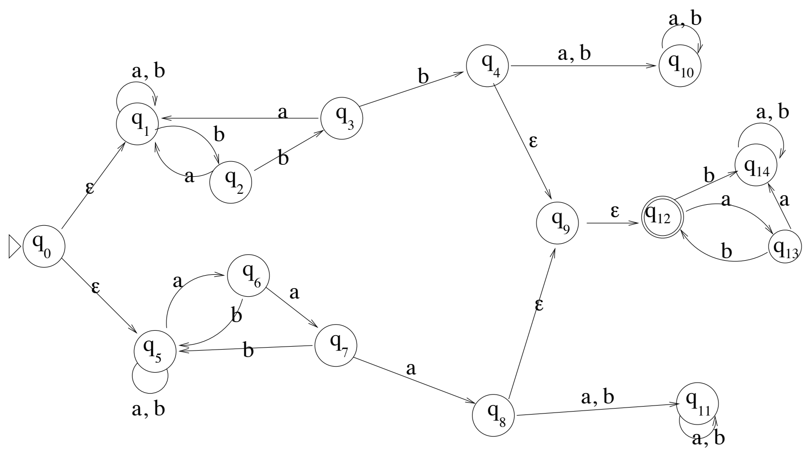


FIGURE 4.10 – Automate indéterministe qui reconnaît le langage $\mathcal{L}$

Reste à supprimer les ϵ-transitions de cet automate pour obtenir un automate sans ϵ-transition qui reconnaisse le même langage.

Nous allons apprendre à supprimer ces dites ϵ-transitions.

4.6 Définition de l'ϵ-fermeture d'un automate $\mathcal{A}_\epsilon$ à ϵ-transitions

4.6.1 Définition de $\hat{\epsilon}$: une fonction de E dans $\mathcal{P}(E)$

pour tout état $e \in E$, $\hat{\epsilon}(e)$ est l'ensemble des états $f \in E$ tels qu'il existe dans $\mathcal{A}_\epsilon$ un chemin de e à f de trace ϵ.
Dans la figure 4.10, $\hat{\epsilon}(q_8) = \{q_8,\ q_9,\ q_{12}\}$.
On appelle fermeture transitive des ϵ-transitions cette fonction $\hat{\epsilon}$.

4.6.2 Construction de $\hat{\epsilon}(e)$

Démarche

On va construire successivement les ensembles $\epsilon_i(e)$, définis en compréhension par : $\hat{\epsilon_i}(e)$ est l'ensemble des états f tel qu'il existe un chemin
- dans $\mathcal{A}$
- de e à f
- composé uniquement d'ϵ-transitions
- composé au plus de i ϵ-transitions

Définition récursive de $\epsilon_i(e)$

- $\epsilon_0(e) = \{e\}$
- $\forall i \in \mathbb{N}\ :\ \epsilon_{i+1}(e) = \epsilon_i(e) \cup \{g \in E \mid \exists f \in \epsilon_i(e) \text{ tel que } f\epsilon g \in \delta_\epsilon\}$

$(\epsilon_i(e))$est une suite finie : $\epsilon_{|E|-1}(e) = \hat{\epsilon}(e)$

L'inclusion $\epsilon_{|E|-1}(e) \subseteq \hat{\epsilon}(e)$ est évidente. Reste à prouver l'inclusion réciproque :

$$
\begin{array}{lll}
f \in \hat{\epsilon}(e) & \Rightarrow & \text{il existe un chemin d'}\epsilon\text{-transitions de } e \text{ à } f \\
 & \Rightarrow & \text{il existe un chemin élémentaire}^1 \text{ d'}\epsilon\text{-transitions de } e \text{ à } f \\
 & \Rightarrow & f \in \epsilon_{|E|-1}(e)
\end{array}
$$

Remarque :

$\forall i \in \mathbb{N}\ :\ [\epsilon_{i+1}(e) = \epsilon_i(e)] \Rightarrow [\forall k \in \mathbb{N}\ :\ \epsilon_{i+k}(e) = \epsilon_i(e)]$ et donc
$\forall i \in \mathbb{N}\ :\ [\epsilon_{i+1}(e) = \epsilon_i(e)] \Rightarrow [\epsilon_i(e) = \hat{\epsilon}(e)]$

Exemple :

figures 4.11, 4.12 et 4.13.

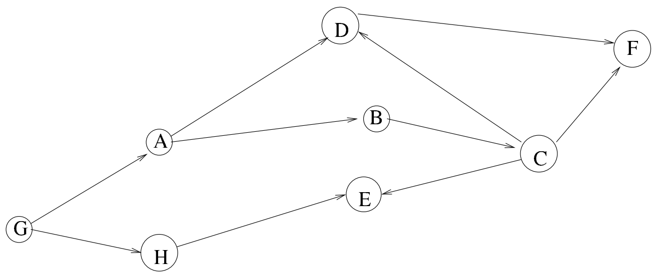


FIGURE 4.11 – Tous les arcs de cette figure sont des ϵ-transitions.

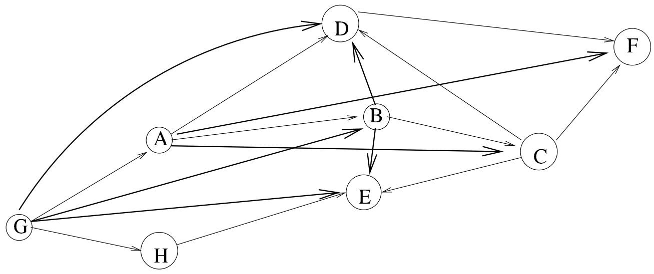


FIGURE 4.12 – on a rajouté les arcs qui apparaissent dans ϵ_2

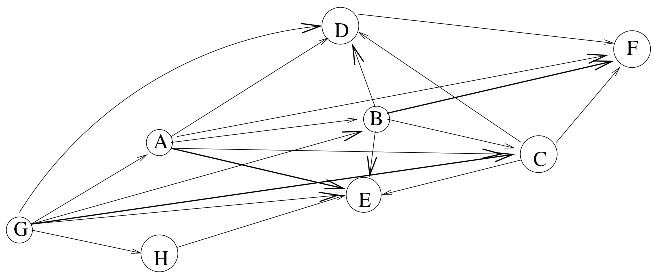


FIGURE 4.13 – on a rajouté les arcs qui apparaissent dans ϵ_3

4.6.3 $\hat{\epsilon}$ est transitive

$$\forall e \in E \ : \ \bigcup_{f\in\hat{\epsilon}(e)} \bigcup_{g\in\hat{\epsilon}(f)} g = \bigcup_{f\in\hat{\epsilon}(e)} f$$

4.7 Définition récursive de δ^* pour les automates qui ont des ϵ-transitions

4.7.1 Rappel de la définition intuitive

$\forall E' \subseteq E$, $\forall m \in \Sigma^*$, $\delta^*(E', m)$ est l'ensemble des états que l'on peut atteindre à partir d'un des états de E' par un chemin de trace m.

4.7.2 Définition récursive

- $\delta^*(E', \epsilon) = \hat{\epsilon}(E')$ (figure 4.14).
- $\delta^*(E', \alpha) = \hat{\epsilon}(\delta(\hat{\epsilon}(E'), \alpha))$ (figure 4.15).
- $\delta^*(E', \alpha m) = \delta^*(\delta^*(E', \alpha), m)$ (figure 4.16).

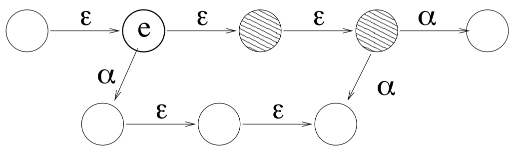


FIGURE 4.14 – $\delta^*(e, \epsilon)$

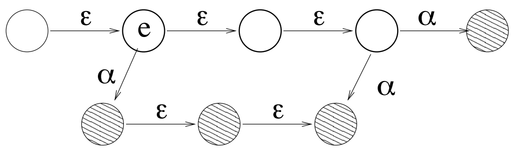


FIGURE 4.15 – $\delta^*(e, \alpha)$

Remarque :

- on étend sans problème la fonction $\hat{\epsilon}$ de E à $\mathcal{P}(E)$: $\hat{\epsilon}(E') = \bigcup_{e'\in E'} \hat{\epsilon}(e')$
- et la fonction δ de E à $\mathcal{P}(E)$: $\delta(E', \alpha) = \bigcup_{e'\in E'} \delta(e', \alpha)$

4.7.3 Théorème : $\delta^*(E', m\alpha) = \delta^*(\delta^*(E', m), \alpha)$

On admet ce théorème qui esr illustré dans la figure 4.17.
par récurrence sur $|m|$: soit $\Pi(l)$ la propriété :
$\forall m \in \Sigma^*, \ \forall \alpha \in \Sigma \ : \ [|m| \leq l] \Rightarrow [\delta^*(E', m\alpha) = \delta^*(\delta^*(E', m), \alpha)]$

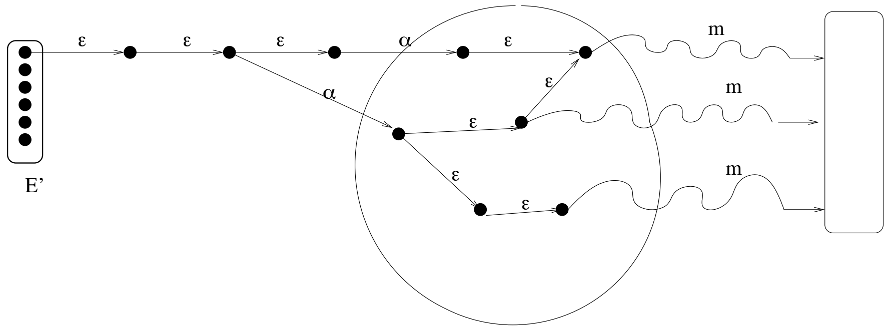


FIGURE 4.16 – $\delta^*(E', \alpha m) = \delta^*(\delta^*(E', \alpha), m)$

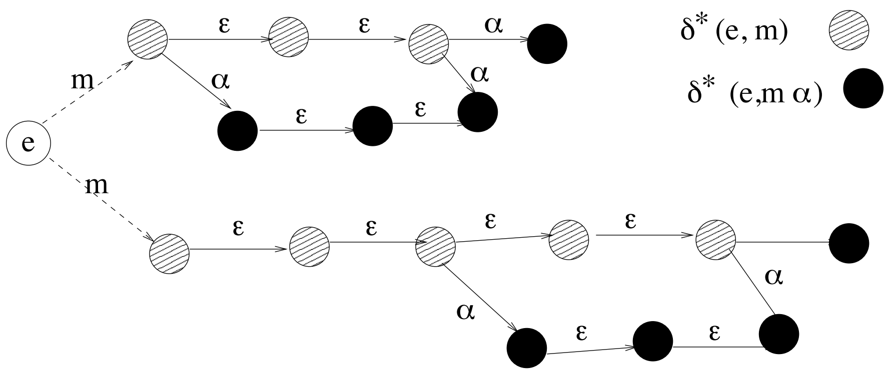


FIGURE 4.17 – $\delta^*(E', m\alpha) = \delta^*(\delta^*(E', m), \alpha)$

Base $\Pi(0)$ est trivial : $\forall\alpha \in \Sigma\ \ \delta^*(E'', \alpha) = \delta^*(\delta^*(E'', \epsilon), \alpha)$ car $\delta^*(E'', \epsilon) = \hat{\epsilon}(E'')$ donc $\delta^*(\delta^*(E'', \epsilon), \alpha) = \delta^*(\hat{\epsilon}(E''), \alpha)$ et vu la définition de $\delta^*(E', \alpha) = \hat{\epsilon}(\delta(\hat{\epsilon}(E'), \alpha))$ on a (en remplaçant E' par $\hat{\epsilon}(E'')$) : $\delta^*(\delta^*(E'', \epsilon), \alpha) = \hat{\epsilon}(\delta(\hat{\epsilon}(\hat{\epsilon}(E'')), \alpha))$ et en utilisant la transitivité de $\hat{\epsilon}$, $\delta^*(\delta^*(E'', \epsilon), \alpha) = \hat{\epsilon}(\delta(\hat{\epsilon}(E''), \alpha)) = \delta^*(E'', \alpha)$

$\Pi(l) \Rightarrow \Pi(l+1)$ Soit un mot m de longueur $l+1$, et soit β la première lettre de ce mot et m' le suffixe ($m = \beta m'$).

$$\delta^*(E', m\alpha) \overset{def\ m}{=} \delta^*(E', \beta m'\alpha) \overset{def\ \delta^*}{=} \delta^*(\delta^*(E', \beta), m'\alpha) \overset{H.R.}{=} \delta^*(\delta^*(\delta^*(E', \beta), m'), \alpha) \overset{def\ \delta^*}{=}$$

$$\delta^*(\delta^*(E', \beta m'), \alpha) \overset{def\ m}{=} \delta^*(\delta^*(E', m), \alpha)$$

4.8 Construction de l'automate sans ϵ-transitions

4.8.1 Résultat voulu qu'il faudra démontrer

L' automate $\mathcal{A}_\epsilon = (\Sigma,\ E,\ I,\ F_\epsilon,\ \delta_\epsilon)$ et l'automate sans ϵ-transitions $\mathcal{A} = (\Sigma,\ E,\ I,\ F,\ \delta)$ reconnaissent le même langage.
Donc, à partir d'un automate quelconque, on peut construire un automate sans ϵ-transitions qui reconnaisse le même langage.

4.8.2 Deux exemples de construction possible d'un automate sans ϵ-transitions

Si on veut obtenir un automate sans ϵ-transitions équivalent à celui de la figure 4.18,

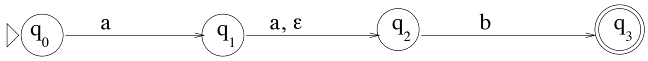


FIGURE 4.18 – automate initial

on obtient soit l'automate de la figure 4.19, soit l'automate de la figure 4.20

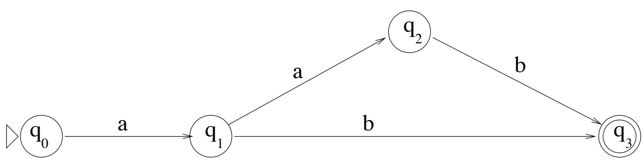


FIGURE 4.19 – automate obtenu en effectuant d'abord les ϵ-transitions

On va s'intéresser à la construction qui fournit la figure 4.19.

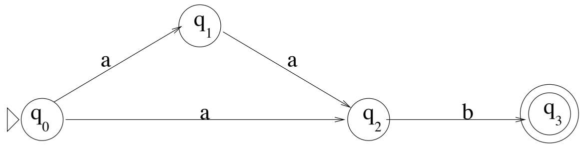


FIGURE 4.20 – autre construction possible

4.8.3 Construction en effectuant d'abord les ϵ-transitions

On définit F par $F = \{e \in E \mid \hat{\epsilon}(e) \cap F_\epsilon \neq \emptyset\}$ et δ par

$$\forall e \in E,\ \forall \alpha \in \Sigma\ :\ \delta(e,\alpha) = \bigcup_{f \in \hat{\epsilon}(e)} \delta_\epsilon(f,\alpha)$$

4.8.4 Démonstration de l'égalité des langages reconnus par les deux automates

Lemme : $\delta^*_\epsilon(e_0,\alpha) = \hat{\epsilon}(\delta(e_0,\alpha))$

$$\delta^*_\epsilon(e_0,\alpha) \overset{def\ \delta^*_\epsilon}{=} \hat{\epsilon}(\delta_\epsilon(\hat{\epsilon}(e_0),\alpha)) \overset{def\ \delta}{=} \hat{\epsilon}(\delta(e_0,\alpha))$$

La figure 4.21 montre la nécessité du $\hat{\epsilon}$.

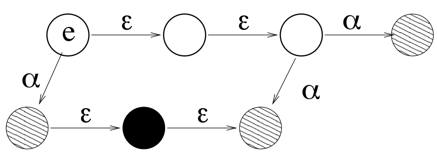


FIGURE 4.21 – L'état ! appartient à mais pas à

Théorème : $\forall m \in \Sigma^*,\ \forall e \in E\ :\ \delta^*_\epsilon(e,m) = \hat{\epsilon}(\delta^*(e,m))$

Se démontre par récurrence sur $l = |m|$. On pose

$$\Pi(l) \ = \ \forall m \in \Sigma^*,\ \forall e \in E\ :\ [|m| \leq l] \Rightarrow [\delta^*_\epsilon(e,m) = \hat{\epsilon}(\delta^*(e,m))]$$

- $\Pi(0)\ :\ \delta^*_\epsilon(e,\epsilon) = \hat{\epsilon}(e) = \hat{\epsilon}(\delta(e,\epsilon))$ trivial
- $\Pi(1)$ correspond au lemme ci dessus.
- Prouvons $\forall l \geq 1\ :\ \Pi(l) \Rightarrow \Pi(l+1)$
  Soit m un mot de longueur $l+1$, α la dernière lettre de ce mot et m' le préfixe tel que $m = m'\alpha$.
  D'après l'hypothèse de récurrence,
  $$\delta^*_\epsilon(e,m') = \hat{\epsilon}(\delta^*(e,m')) \tag{4.1}$$

et on a vu le théorème qui dit que

$$\delta_\epsilon^*(e,m) = \delta_\epsilon^*(\delta_\epsilon^*(e,m'),\alpha) \tag{4.2}$$

et

$$\delta^*(e,m) = \delta^*(\delta^*(e,m'),\alpha) \tag{4.3}$$

et on a alors

$$\delta_\epsilon^*(e,m) \overset{4.2}{=} \delta_\epsilon^*(\delta_\epsilon^*(e,m'),\alpha) \overset{4.1}{=} \delta_\epsilon^*(\hat\epsilon(\delta_\epsilon^*(e,m')),\alpha) \overset{def\ \delta_\epsilon^*}{=}$$
$$\hat\epsilon(\delta_\epsilon(\hat\epsilon(\hat\epsilon(\delta_\epsilon^*(e,m'))),\alpha)) \overset{\hat\epsilon\ transitif}{=} \hat\epsilon(\delta_\epsilon(\hat\epsilon(\delta_\epsilon^*(e,m')),\alpha)) \overset{def\ \delta}{=}$$
$$\hat\epsilon(\delta(\delta^*(e,m'),\alpha)) \overset{def\ \delta^*}{=} \hat\epsilon(\delta^*(e,m))$$

Preuve de l'égalité des langages reconnus par $\mathcal{A}_\epsilon$ et $\mathcal{A}$ En appliquant le théorème à $J = I_\epsilon = I$

$$\forall m \in \Sigma^*,\ \forall J \in \mathcal{P}(E)\ :\ \delta_\epsilon^*(I_\epsilon,m) = \hat\epsilon(\delta^*(I_\epsilon,m))$$

donc m est reconnu par $\mathcal{A}$ si et seulement si m est reconnu par $\mathcal{A}_\epsilon$

Chapitre 5

Déterminisation d'automate

5.1 Théorème sur la déterminisation d'automate

5.1.1 Énoncé

Un langage est rationnel seulement si il est reconnu par un automate fini déterministe.
Comme un automate fini déterministe est un automate indéterministe particulier, si un langage est reconnu par un automate déterministe, il est évidement rationnel.

Explication : Soit $\mathcal{A}_i = (\Sigma, E_i, I_i, F_i, \delta_i)$ un automate indéterministe sans ϵ-transition, nous allons définir (construire) un automate déterministe
$\mathcal{A}_d = (\Sigma, E_d = \mathcal{P}(E_i), i_d, F_d, \delta_d)$ qui reconnaisse le même langage que $\mathcal{A}_i$.

5.1.2 Application : complémentaire d'un langage rationnel

Soit un langage rationnel L, le théorème ci dessus énonce qu'il existe un automate déterministe qui le reconnaît, donc il existe un automate déterministe complet $(\Sigma,\ E_L,\ i_L,\ F_L,\ \delta_L)$ qui le reconnaît, et il est trivial que l'automate $(\Sigma,\ E_L,\ i_L,\ E_L \setminus F_L,\ \delta_L)$ reconnaît le langage complémentaire de L.

5.1.3 Définition de l'automate déterministe.

définition de F_d : $F_d = \{E' \in E_d = \mathcal{P}(E_i) \mid E' \cap F_i \neq \emptyset\}$

définition de i_d : $i_d = I_i \in E_d = \mathcal{P}(E_i)$.

définition de δ_d : $\delta_d(E',\ \alpha) = \delta_i(E',\ \alpha) \in \mathcal{P}(E_i)$

Cette définition implique que $\mathcal{A}_d$ est déterministe.

Exemple

Considérons l'automate indéterministe de la figure 5.1 qui reconnaît le langage des mots sur $\{a, b\}$ qui contiennent au moins une occurrence de aba.

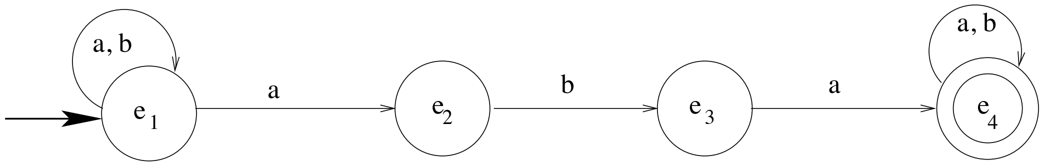


Figure 5.1 – Déterminisons cet automate

L'état d'entrée de l'automate déterministe correspondant est $\{e_1\}$ (figure 5.2).

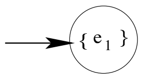


Figure 5.2 – voici l"tat d'entrée

Par la transition b, envoie dans l'automate indéterministe en e_1, donc dans l'automate déterministe en construction en $\{e_1\}$ (figure 5.3).

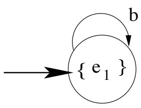


Figure 5.3 – et la transition b

Mais dans l'automate indéterministe la transition a envoie de e_1 soit en e_1 soit en e_2, donc dans l'automate déterministe en $\{e_1,\ e_2\}$ (figure 5.4).
Dans l'automate indéterministe la transition a envoie de e_1 soit en e_1 soit en e_2 et aucune transition étiquetée a ne permet de quitter e_2, donc dans l'automate déterministe, depuis l'état $\{e_1,\ e_2\}$ la transition a envoie en $\{e_1,\ e_2\}$ (figure 5.5).

Par contre, dans l'automate indéterministe, la transition b envoie de e_1 en e_1 et de e_2 en e_3, donc dans l'automate déterministe, de l'état $\{e_1,\ e_2\}$ la transition b envoie en $\{e_1,\ e_3\}$ (figure 5.6).

Dans l'automate indéterministe la transition b boucle sur e_3 comme sur e_1, donc dans l'automate déterministe la même transition envoie de $\{e_1,\ e_3\}$ à $\{e_1\}$. Quant à la transition a, elle envoie de $\{e_1,\ e_3\}$ à $\{e_1, e_2, e_4\}$ qui est un état final puisque son étiquette contient e_4 (figure 5.7).

La transition a boucle sur $\{e_1, e_2, e_4\}$ par l et la transition b envoie sur l'état $\{e_1, e_3, e_4\}$ (figure 5.8). Ce nouvel état est aussi final puisqu'il contient e_4.

La figure 5.9 nous indique les transitions qui partent de l'état $\{e_1, e_2, e_4\}$.

et la figure 5.10 nous indique les transitions qui partent de l'état $\{e_1, e_4\}$.

A ce stade, les états qui n'ont pas été dessinés sont inaccessibles.

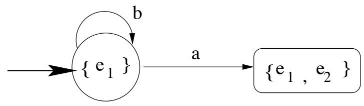


FIGURE 5.4 – puis la transition a

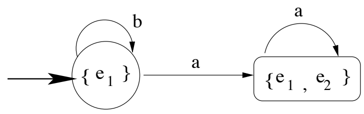


FIGURE 5.5 – encore une transition a

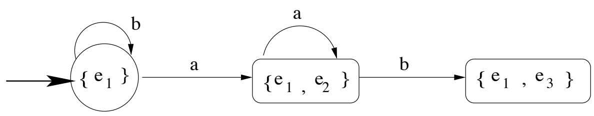


FIGURE 5.6 – et une transition b

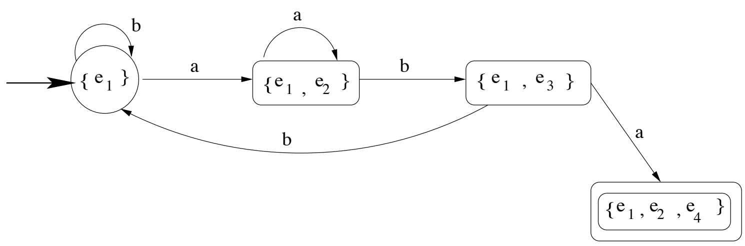


FIGURE 5.7 – encore une transition a et une transition b

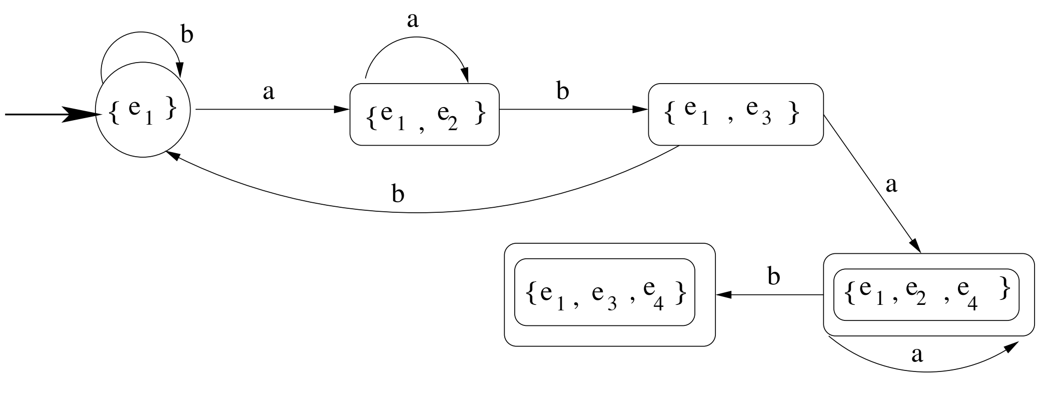


FIGURE 5.8 – deux anté pénultièmes transitions

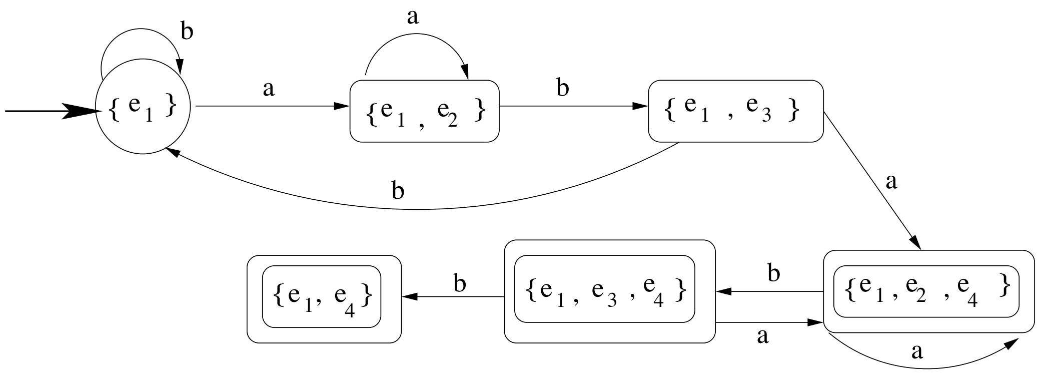


FIGURE 5.9 – et deux avant dernières transitions

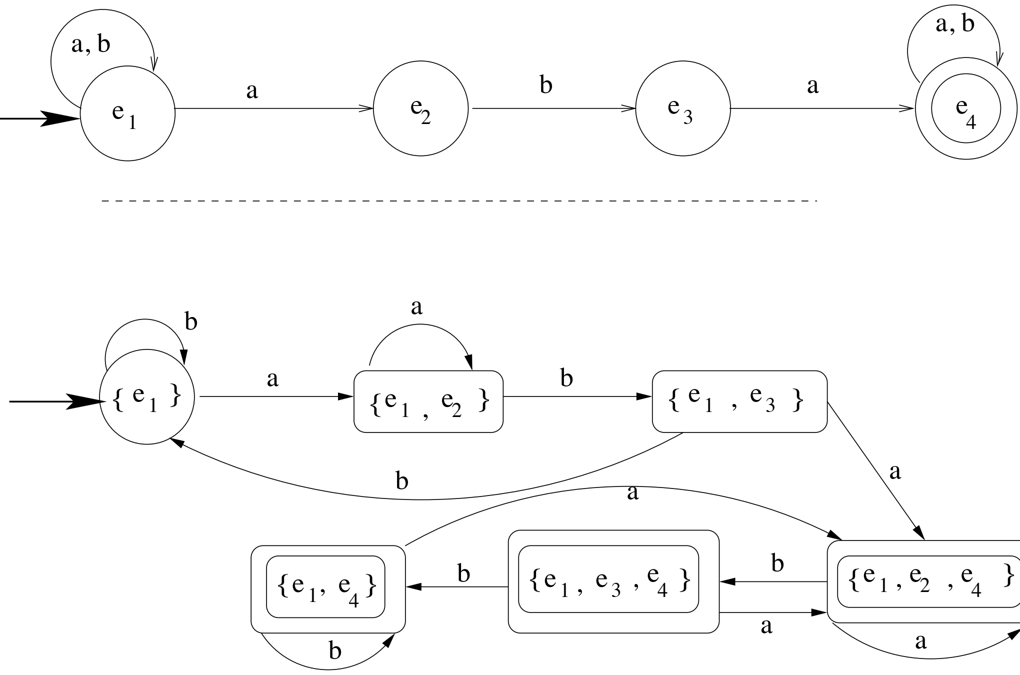


FIGURE 5.10 – Automate déterministe

5.1.4 Théorème : $\delta_d^*(Y,m) = \delta_i^*(Y,m)$

Conséquence : égalité des deux langages

Soit $\mathcal{A}_i = (\Sigma,\ E, I,\ F_i,\ \delta_i)$ un AF sans ϵ-transition et $\mathcal{A}_d = (\Sigma,\ E_d = \mathcal{P}(E_i),\ \{I\},\ F_d,\ \delta_d)$ l'AFD associé à $\mathcal{A}_i$, et δ_i^* et δ_d^* leurs fonctions de transitions itérées.

Remarque : Dans l'automate déterministe $\mathcal{A}_d$, la fonction δ_d^* est une fonction de $E_d = \mathcal{P}(E_i) \times \Sigma^* \longrightarrow E_d = \mathcal{P}(E_i)$.

On va démontrer que :

$$\forall m \in \Sigma^*,\ \forall Y \in E_d = \mathcal{P}(E_i),\ \delta_d^*(Y,m) = \bigcup_{e \in Y} \delta_i^*(e,m) = \delta_i^*(Y,m)$$

Conséquence : $\mathcal{A}_d$ reconnaît le même langage que $\mathcal{A}_i$

$\forall m \in \Sigma^*$: $\delta_d^*(I_d,m) = \delta_i^*(I_d,m) = \delta_i^*(I_i,m)$ et $F_d = \{E' \in E_d = \mathcal{P}(E_i)$ tel que $E' \cap F_i \neq \emptyset\}$ donc

$\forall m \in \Sigma^*$: $\delta_d^*(I_d,m) \cap F_d \neq \emptyset \iff$

$\exists e_0 \in I_i$ tel que $\delta_i^*(i,m) \in F_d$ c'est à dire tel que $\delta_i^*(e_0,m) \cap F_i \neq \emptyset$.

Preuve par récurrence sur $|m|$

- $m = \epsilon \ \Rightarrow\ \delta_d^*(Y,\epsilon) = Y = \bigcup_{e \in Y} \{e\} = \bigcup_{e \in Y} \delta_i^*(e,\epsilon) = \delta_i^*(Y,\epsilon)$

• pour $\alpha \in \Sigma$ et $Y \in E_d = \mathcal{P}(E_i)$, par définition de δ_d^*
$\delta_d^*(Y,\alpha) = \delta_i^*(Y,\alpha)$

• Supposons que pour tout mot m de longueur n
$\forall Z \in E_d = \mathcal{P}(E_i)$: $\delta_d^*(Z,m) = \delta_i^*(Z,m)$
Soit m' un mot de longueur $n+1$. m' s'écrit $m' = am$ où a est une lettre de Σ et m un mot de longueur n.

- $\delta_d^*(Y,m') = \delta_d^*(Y,am) = \delta_d^*(\delta_d(Y,a),m)$
- mais $\delta_d(Y,a) = \delta_i(Y,a)$
- donc $\delta_d^*(Y,m') = \delta_d^*(\delta_i(Y,a),m)$
- en utilisant l'hypothèse de récurrence sur $Z = \delta_i(Y,a)$, à savoir $\delta_d^*(Z,m) = \delta_i^*(Z,m)$ ou autrement dit $\delta_d^*(\delta_i(Y,a),m) = \delta_i^*(\delta_i(Y,a),m)$ on obtient $\delta_d^*(Y,m') = \delta_i^*(\delta_i(Y,a),m) = \delta_i^*(Y,m') = \delta_i^*(Y,am)$

Chapitre 6

Expressions rationnelles

6.1 Définitions récursives

6.1.1 Définition syntaxique récursive des expressions rationnelles

Les expressions rationnelles (ER) sur un alphabet Σ sont définies inductivement par :

- $\emptyset$, ϵ et a, où $a \in \Sigma$, sont des ER
- Si r et s sont des ER,
  - (r)
  - $r + s$
  - rs
  - r^*

  sont des ER.

6.1.2 Définition récursive des langages décrits par des expressions rationnelles

Le langage $L(r)$ décrit par une expression rationnelle r sur l'alphabet Σ est défini par :

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $\forall a \in \Sigma,\ L(a) = \{a\}$
- r et s étant 2 expressions rationnelles qui décrivent les langages $L(r)$ et $L(s)$,
  - $L((r)) = L(r)$
  - $L(r + s) = L(r) \cup L(s)$
  - $L(rs) = L(r).L(s)$
  - $L(r^*) = (L(r))^*$

6.2 Équivalence des automates d'états finis et des expressions rationnelles

Automates d'états finis et expressions rationnelles sont des **modèles** équivalents. Ce qui s'énonce plus formellement par

- Pour tout langage L reconnu par un automate d'états fini $\mathcal{A}$, il existe une expression rationnelle r qui définit L.
- Pour tout langage L défini par une expression rationnelle r, il existe un automate d'états fini $\mathcal{A}$ qui reconnaît L.

6.2.1 Propriétés de fermeture des langages rationnels

Sont évidement des langages rationnels :

- le langage vide ($\emptyset$), ainsi que le langage réduit au mot vide $\{\epsilon\}$,
- $\forall\Sigma$ alphabet fini, Σ.

On a déjà vu dans les chapitre précédents sur les automates que sont des langages rationnels :

- l'union de deux langages rationnels,
- la concaténation de deux langages rationnels,
- et la fermeture de Kleene d'un langage rationnel.

Et ce qu'on a vu dans le chapitre précédent prouve que sont des langages rationnels :

- le complémentaire dans Σ^* d'un langage rationnel sur Σ,
- donc l'intersection de deux langages rationnels.

6.3 Calcul d'une expression rationnelle correspondant à un automate

6.3.1 par variation des états d'entrée

Ce calcul consiste à résoudre un système d'équations sur les langages $L(e)$ associés à chaque état e de l'automate complet $(\Sigma,\ E,\ I,\ F,\ \delta)$. On associe à chaque état $e \in E$ le langage $L_e = \{m \in \Sigma^* \mid \delta^*(e, m) \cap F \neq \emptyset\}$. Ce langage peut être défini par les langages associés aux successeurs de l'état e grâce aux équations suivantes :

$$L_e = \bigcup_{eaf \in \delta} \{a\}.L_f \quad \text{si } e \notin F$$

$$L_e = \{\epsilon\} \cup \bigcup_{eaf \in \delta} \{a\}.L_f \quad \text{si } e \in F$$

En associant à chaque langage L_e son expression rationnelle R_e, on en déduit les équations :

$$R_e = \sum_{eaf \in \delta} a.R_f \quad \text{si } e \notin F$$

$$R_e = \epsilon \ + \sum_{eaf \in \delta} a.R_f \quad \text{si } e \in F$$

Le système d'équations $\mathcal{S}_\mathcal{A}$ est l'ensemble des équations associées à chaque état de $\mathcal{A}$ par les règles ci dessus.

Exemple : calculons le système d'équations pour l'automate de la figure 6.1.

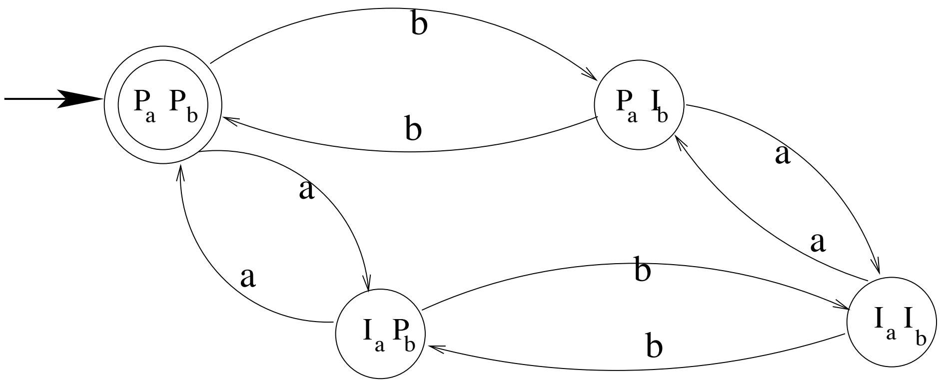


FIGURE 6.1 – Quelle est l'expression rationnelle du langage reconnu par cet automate ?

- $R_{P_aP_b} \ = \ aR_{I_aP_b} \ + \ bR_{P_aI_b} \ + \ \epsilon$
- $R_{I_aP_b} \ = \ aR_{P_aP_b} \ + \ bR_{I_aI_b}$
- $R_{I_aI_b} \ = \ aR_{P_aI_b} \ + \ bR_{I_aP_b}$
- $R_{P_aI_b} \ = \ aR_{I_aI_b} \ + \ bR_{P_aP_b}$

Le système d'équation $\mathcal{S}_\mathcal{A}$ associé à un automate fini $\mathcal{A}$ est un ensemble de relations que doivent vérifier les expressions rationnelles des langages associés aux états. Une solution de ce système est une valeur pour chaque R_e, solution vérifiant l'ensemble de ces relations. L'expression rationnelle du langage reconnu par $\mathcal{A}$ est la valeur donnée à $\sum_{i \in I} R_i$. Pour résoudre un système d'équations sur les langages on utilise les 2 règles :

1. *règle de substitution* : si $R_e = \Phi$, où Φ est une expression ne contenant pas R_e, est une équation de $\mathcal{S}_\mathcal{A}$ alors on peut substituer uniformément Φ à chaque occurrence de R_e dans les autres équations de $\mathcal{S}_\mathcal{A}$;
2. *règle du point fixe* : une solution de l'équation $R_e = R_1.R_e + R_2$, où R_1 et R_2 sont des expressions sans occurrence de R_e, est $R_e = R_1^*.R_2$.
   On remplace donc dans $\mathcal{S}_\mathcal{A}$ l'équation $R_e = R_1.R_e + R_2$ par l'équation $R_e = R_1^*.R_2$

En appliquant l'une de ces deux règles on supprime une variable du système d'équations [1]. En réitérant le procédé tant que possible, on obtient un système d'équations dont les parties droites sont des ER (sans variable R_e).

suite de l'exemple

En appliquant la règle de substitution en remplaçant dans le système obtenu précédemment $R_{P_aI_b}$ par $aR_{I_aI_b} + bR_{P_aP_b}$ on obtient :

- $R_{P_aP_b} = aR_{I_aP_b} + b(aR_{I_aI_b} + bR_{P_aP_b}) + \epsilon = aR_{I_aP_b} + baR_{I_aI_b} + bbR_{P_aP_b} + \epsilon$
- $R_{I_aP_b} = aR_{P_aP_b} + bR_{I_aI_b}$
- $R_{I_aI_b} = aaR_{I_aI_b} + abR_{P_aP_b} + bR_{I_aP_b}$

De même en remplaçant $R_{I_aP_b}$ par $aR_{P_aP_b} + bR_{I_aI_b}$ on obtient

- $R_{P_aP_b} = aaR_{P_aP_b} + abR_{I_aI_b} + baR_{I_aI_b} + bbR_{P_aP_b} + \epsilon$
- $R_{I_aI_b} = aaR_{I_aI_b} + abR_{P_aP_b} + baR_{P_aP_b} + bbR_{I_aI_b}$

ce qui par factorisation se réecrit en

- $R_{P_aP_b} = (aa + bb)R_{P_aP_b} + (ab + ba)R_{I_aI_b} + \epsilon$
- $R_{I_aI_b} = (aa + bb)R_{I_aI_b} + (ab + ba)R_{P_aP_b}$

en appliquant la règle du point fixe [2] à la deuxième équation on obtient $R_{I_aI_b} = (aa + bb)^*(ab + ba)R_{P_aP_b}$,

puis en substituant cette valeur dans la première équation on obtient $R_{P_aP_b} = (aa + bb)R_{P_aP_b} + (ab + ba)(aa + bb)^*(ab + ba)R_{P_aP_b} + \epsilon$,

ou après factorisation
$R_{P_aP_b} = ((aa + bb) + (ab + ba)(aa + bb)^*(ab + ba))R_{P_aP_b} + \epsilon$

et en appliquant la règle du point fixe [3] à cette équation on obtient
$R_{P_aP_b} = ((aa + bb) + (ab + ba)(aa + bb)^*(ab + ba))^*$
ce qui est une expression rationnelle du langage de l'automate de la figure 6.1.

6.3.2 par variation des états de sortie

On associe à chaque état $e \in E$ le langage $L_e = \{m \in \Sigma^* \mid \delta^*(I, m) \cap e \neq \emptyset\}$ et on appelle R_e une expression rationnelle de ce langage. Cette expression rationnelle peut être définie par les expressions rationnelles des langages associés aux successeurs de l'état e grâce aux équations suivantes :

$$R_e = \bigcup_{fae \in \delta} R_f.\{a\} \quad \text{si } e \notin I$$

$$R_e = \{\epsilon\} \cup \bigcup_{fae \in \delta} R_f.\{a\} \quad \text{si } e \in I$$

1. plus précisément, le nombre de variables R_e apparaissant dans la partie droite des équations diminue de 1.
2. avec $R_1 = (aa + bb)$ et $R_2 = (ab + ba)R_{P_aP_b}$
3. avec $R_1 = ((aa + bb) + (ab + ba)(aa + bb)^*(ab + ba))$ et $R_2 = \{\epsilon\}$

Exemple : voici le système d'équations pour l'automate de la figure 6.1.
- $R_{P_aP_b} = R_{I_aP_b}a + R_{P_aI_b}b + \epsilon$
- $R_{I_aP_b} = R_{P_aP_b}a + R_{I_aI_b}b$
- $R_{P_aI_b} = R_{I_aI_b}a + R_{P_aP_b}b$
- $R_{I_aI_b} = R_{P_aI_b}a + R_{I_aP_b}b$

L'expression rationnelle du langage reconnu par $\mathcal{A}$ est la valeur donnée à $\sum_{f\in F} R_f$. Pour résoudre un système d'équations sur les langages on utilise la même *règle de substitution* que ci dessus mais une nouvelle *règle du point fixe* : une solution de l'équation $R_e = R_e.R_1 + R_2$, où R_1 et R_2 sont des expressions sans occurrence de R_e, est $R_e = R_2.R_1^*$.

suite de l'exemple

- $RI_aI_b = R_{P_aI_b}a + R_{I_aP_b} = (R_{I_aI_b}a + R_{P_aP_b}b)a + (R_{P_aP_b}a + R_{I_aI_b}b)b = R_{I_aI_b}(aa+bb) + R_{P_aP_b}(ba+ab)$
  $= R_{P_aP_b}(ba+ab)(aa+bb)^*$
- $R_{P_aI_b} = R_{I_aI_b}a + R_{P_aP_b}b = R_{P_aP_b}[(ba+ab)(aa+bb)^*a + b]$
- $R_{I_aP_b} = R_{P_aP_b}a + R_{I_aI_b}b = R_{P_aP_b}\{a + (ba+ab)(aa+bb)^*b\}$
- $R_{P_aP_b} = R_{I_aP_b}a + R_{P_aI_b}b + \epsilon$
  $= R_{P_aP_b}\{[a + (ba+ab)(aa+bb)^*b]a + [(ba+ab)(aa+bb)^*a + b]b\} + \epsilon$
  $= ([a + (ba+ab)(aa+bb)^*b]a + [(ba+ab)(aa+bb)^*a + b]b)^*$
- $R = ([a + (ba+ab)(aa+bb)^*b]a + [(ba+ab)(aa+bb)^*a + b]b)^*$

6.4 Calcul d'un automate correspondant à une expression rationnelle

Nous savons que la classe des langages reconnaissables sur un alphabet Σ
- contient le langage vide, le langage réduit au mot vide et les langages réduits à une lettre de Σ
- et est fermée pour les opérations union, intersection, complémentaire, concaténation et étoile.

exemple : calculons l'automate reconnaissant le langage défini par l'expression rationnelle $(ab + ba)(aa + bb)^*$.

La figure 6.2 montre les premiers pas de la construction.

On standardise ces deux automates de façon classique dans la figure 6.3

Ce qui permet d'obtenir l'automate correspondant à l'expression rationnelle $(ab + ba)(aa + bb)$ dans la figure 6.4

Et une dernière ϵ-transition permet d'obtenir l'automate correspondant à l'expression rationnelle $((ab + ba)(aa + bb))^*$ dans la figure 6.5

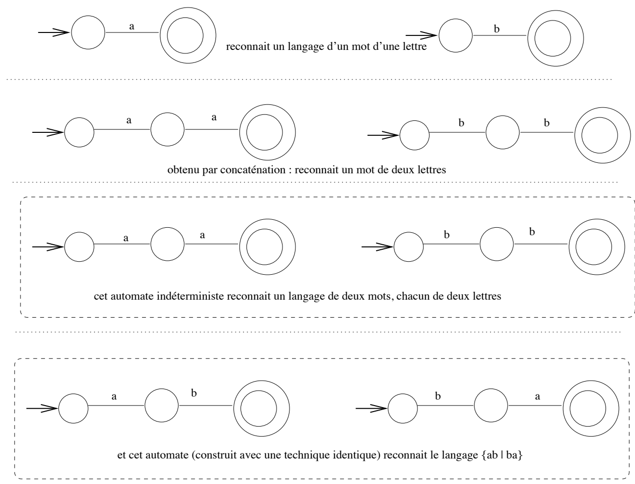
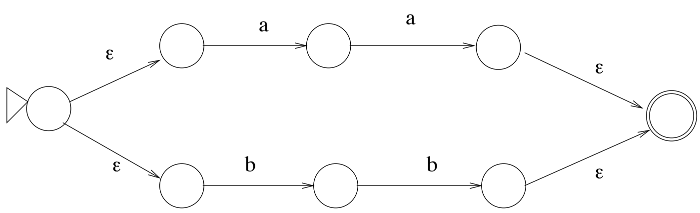


Figure 6.2 – premiers pas de la construction



Automate standard reconnaissant le langage défini par l'expression aa + bb

Automate standard reconnaissant le langage défini par l'expression ab + ba

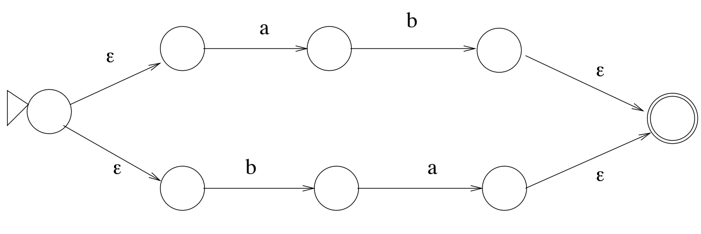


Figure 6.3 – standardisation

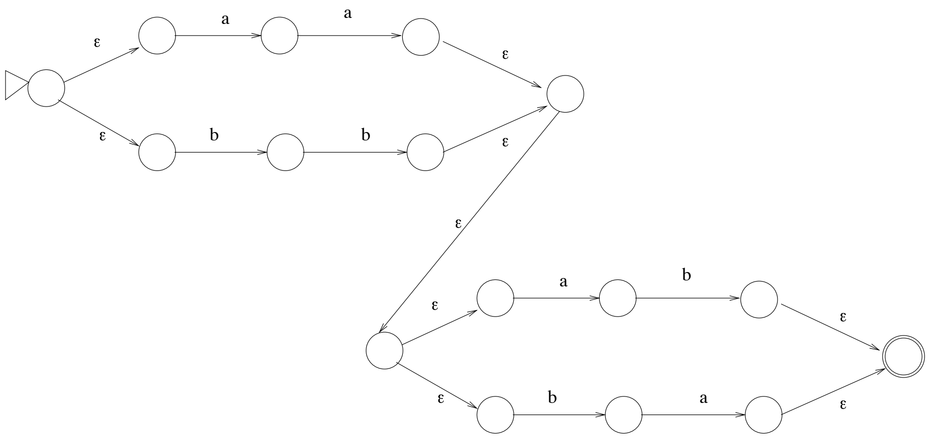


FIGURE 6.4 – Automate standard correspondant à $(aa + bb)(ab + ba)$

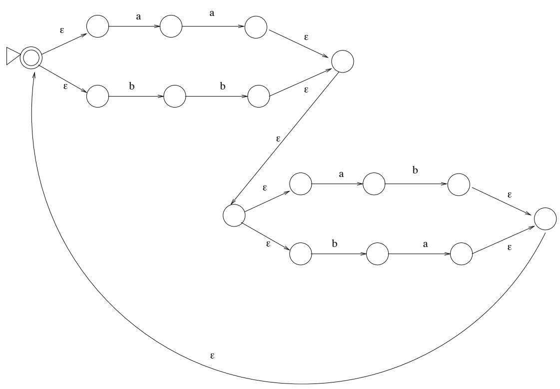


FIGURE 6.5 – Automate standard correspondant à $((aa + bb)(ab + ba))^*$

Chapitre 7

Minimisation d'automate déterministe complet.

7.1 Présupposés : automate déterministe non trivial monogène

automates monogènes :
ce sont des automates dont tous les états sont accessibles.

automate déterministe triviaux monogènes :
les automate déterministe triviaux sont ceux qui reconnaissent les seuls langages $\emptyset$ et Σ^*.
Un automate déterministe $\{\Sigma,\ E,\ i,\ F,\ \delta\}$ monogène est trivial si et seulement si $F = E$ ou $F = \emptyset$.
Un automate déterministe trivial monogène minimum a un seul état.

On veut minimiser les automates déterministes complets monogènes non triviaux.

7.2 Equivalence de Nérode

7.2.1 Définition des états distinguables

Deux états e_1 et e_2 d'un automate déterministe complet $\mathcal{A} = \{\Sigma,\ E, i,\ F,\ \delta\}$ sont distinguables si et seulement si il existe au moins un mot $m \in \Sigma^*$ tel que $\delta^*(e_1, m) \in F \iff \delta^*(e_2, m) \notin F$. On dit que e_1 et e_2 sont **séparés** par un tel mot m.
Dans la figure 7.1, q_1 et q_2 sont séparés par aa, q_6 et q_7 ne sont pas distinguables.

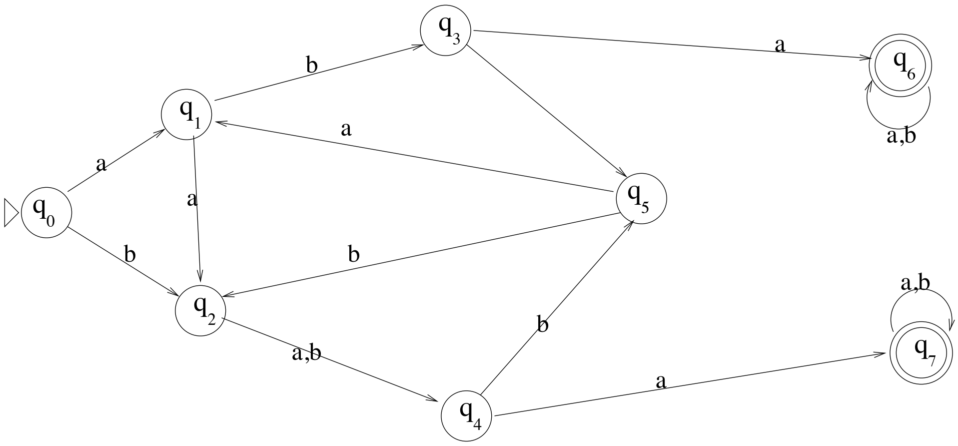


FIGURE 7.1 – états distinguables

7.2.2 relation de Nérode : $\equiv_N$

C'est une relation sur E : $e_1 \equiv_N e_2 \iff e_1$ et e_2 ne sont pas distinguables, ou autrement dit :

$$e_1 \equiv_N e_2 \iff \forall m \in \Sigma^* \ : \ (\delta^*(e_1, m) \in F \iff \delta^*(e_2, m) \in F)$$

la relation de Nérode est une relation d'équivalence : trivial.

remarque : un état terminal et un état non terminal ne peuvent être équivalents, puisqu'ils sont distinguables par ϵ. Donc si une classe d'équivalence de $\equiv_N$ contient un état terminal, tous les états de la classe sont terminaux.

7.3 Quotient d'automate déterministe complet par la relation de Nérode

Soit un automate déterministe complet $\mathcal{A} = \{\Sigma,\ E,\ i,\ F,\ \delta\}$, définissons un automate déterministe complet $\overline{\mathcal{A}} = \{\Sigma,\ \overline{E},\ \overline{i},\ \overline{F},\ \overline{\delta}\}$ avec

- $\overline{E}$ est l'ensemble des classes d'équivalence de E pour $\equiv_N$,
- $\overline{i}$ est la classe qui contient i, l'état initial de $\mathcal{A}$,
- $\overline{F}$ est l'ensemble des classes constituées d'états de F et
- $\overline{\delta}$ défini par : $\overline{e'}\ \alpha\ \overline{e''} \in \overline{\delta} \iff \exists e' \in \overline{e'}, e'' \in \overline{e''}$ tel que $e'\ \alpha\ e'' \in \delta$. Autrement dit $\overline{\delta}(\overline{e}, \alpha) = \overline{\delta(e, \alpha)}$

7.3.1 L'automate quotient est complet

car l'automate initial est complet.

7.3.2 L'automate quotient est déterministe

parce que si deux états e_1' et e_2' d'une même classe ont une transition étiquetée α vers respectivement e_1'' et e_2'', alors aucun mot m ne peut distinguer e_1'' et e_2'' sinon le mot αm distinguerait e_1' et e_2' (figure 7.2).

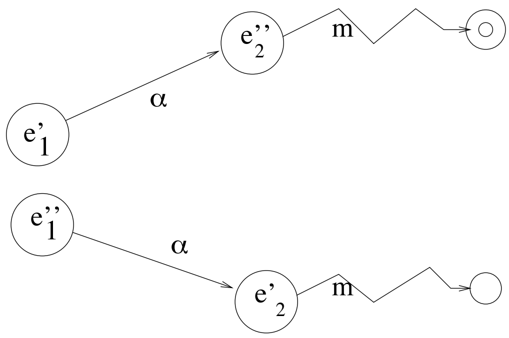


FIGURE 7.2 – e_1' et e_2' ne peuvent appartenir à la même classe d'équivalence

7.3.3 $\mathcal{A}$ et $\overline{\mathcal{A}}$ reconnaissent le même langage

$\mathcal{L}_\mathcal{A} \subseteq \mathcal{L}_{\overline{\mathcal{A}}}$

Soit un mot $m = \alpha_1\alpha_2 \ldots \alpha_p$ tel qu'il existe dans $\mathcal{A}$ un chemin $i\alpha_1 e_1\alpha_2 \ldots \alpha_p e_p$ avec $e_p \in F$. Alors par construction de $\overline{\mathcal{A}}$, $\overline{i}\alpha_1\overline{e_1}\alpha_2 \ldots \alpha_p\overline{e_p}$ est un chemin dans $\overline{\mathcal{A}}$ de l'état initial à un état final. C'est à dire que tout mot reconnu par $\mathcal{A}$ est reconnu par $\overline{\mathcal{A}}$ (figure 7.3).

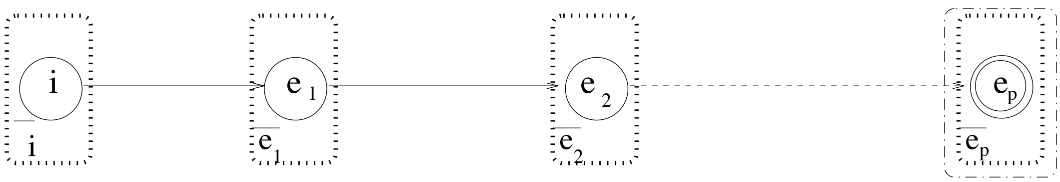


FIGURE 7.3 – tout chemin $\mathcal{A}$ dans entraine l'existence d'un même chemin dans $\overline{\mathcal{A}}$

$\mathcal{L}_{\overline{\mathcal{A}}} \subseteq \mathcal{L}_\mathcal{A}$

On va démontrer par récurrence sur la longueur l des mots m la propriété $\forall l\ P(l)$

avec $P(l) : \forall e \in E,\ \forall m \in \Sigma^*\ :\ |m| \leq l \Rightarrow (\overline{\delta}^*(\overline{e}, m) = \overline{\delta^*(e, m)})$

Base : $P(0)$ **est évident** : $\overline{\delta}^*(\overline{e},\epsilon) = \overline{e} = \overline{\delta(e,\epsilon)}$

Prouvons $P(1)$

Par définition de $\overline{\delta}$: $\overline{\delta}(\overline{e},\alpha) = \overline{\delta(e,\alpha)}$ donc $\overline{\delta}^*(\overline{e},\alpha) = \overline{\delta^*(e,\alpha)}$

Induction : $P(l) \Rightarrow P(l+1)$: (figure 7.4)

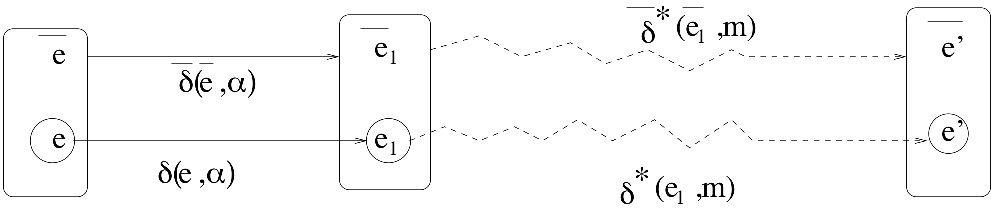


FIGURE 7.4 – induction

Soit αm un mot de longueur $l+1$ (avec $|m| = l \geq 1$).
Soit e un état quelconque de $\mathcal{A}$ et e_1 l'état de $\mathcal{A}$ tel que $e\alpha e_1 \in \delta$.
Par hypothèse de récurrence (puisque $|m| = l$) : $\overline{\delta}^*(\overline{e_1},m) = \overline{\delta^*(e_1,m)}$.
D'autre part $\overline{\delta}^*(\overline{e},\alpha m) = \overline{\delta}^*(\overline{\delta}(\overline{e},\alpha),m) = \overline{\delta}^*(\overline{e_1},m)$ d'après $P(1)$
Donc $\overline{\delta}^*(\overline{e},\alpha m) = \overline{\delta}^*(\overline{e_1},m) = \overline{\delta^*(e_1,m)} = \overline{\delta^*(e,\alpha m)}$

Propriété démontrée : $\forall e \in E,\ \forall m \in \Sigma^*\ \overline{\delta}^*(\overline{e},m) = \overline{\delta^*(e,m)}$

Conclusion : $\mathcal{L}_{\mathcal{A}} \subseteq \mathcal{L}_{\overline{\mathcal{A}}}$
en appliquant la propriété ci dessus à l'état de départ i de $\mathcal{A}$, on obtient
$\forall m \in \Sigma^*$: $\overline{\delta}^*(\overline{i},m) \in \overline{F} \Rightarrow \overline{\delta^*(i,m)} \in \overline{F} \Rightarrow \delta^*(i,m) \in F$ car $\overline{F}$ est constitué uniquement d'états terminaux de $\mathcal{A}$.

7.4 Construction de l'équivalence de Nérode

7.4.1 Relation $\equiv_i$ sur E

Définition

$e' \equiv_i e''$ si et seulement si e' et e'' ne sont distinguables par aucun mot de longueur $\leq i$.

Propriétés évidentes

$\forall e', e'' \in E\ :\ e' \equiv_N e'' \iff (\forall i \in \mathbb{N}\ :\ e' \equiv_i e'')$
$\forall i \in \mathbb{N},\ \forall e', e'' \in E\ :\ e' \equiv_{i+1} e'' \Rightarrow e' \equiv_i e''$

7.4.2 Deux exemples de minimisation d'automate

Premier exemple : minimisons l'automate de la figure 7.5 :

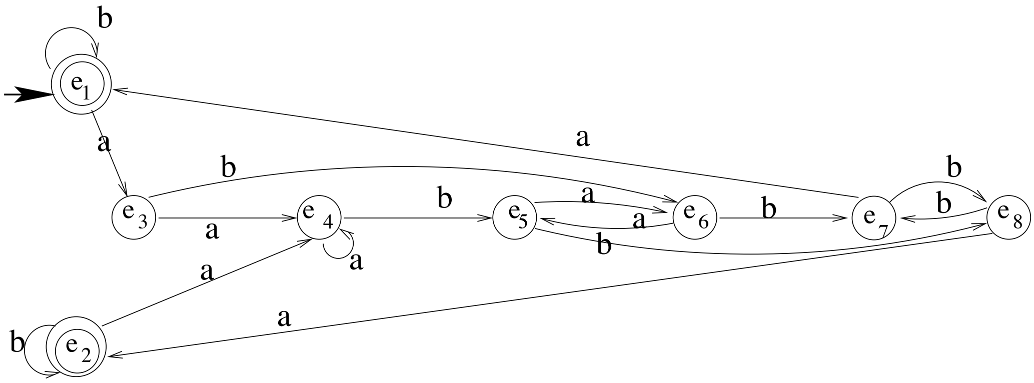


FIGURE 7.5 – Automate à minimiser

Construisons les classes d'équivalence des relation $\equiv_i$ successives :

– $\equiv_0$: ϵ sépare $\{e_1, e_2\}$ de $\{e_3, e_4, e_5, e_6, e_7, e_8\}$

– $\equiv_1$ a pour classes d'équivalence $\{e_1,\ e_2\}$, $\{e_3,\ e_4,\ e_5,\ e_6\}$ et $\{e_7,\ e_8\}$:

. aucun mot de longueur 1 ne sépare e_1 de e_2, ni e_7 de e_8, ni $e_3,\ e_4,\ e_5$ ou e_6

. e_7 et e_8 sont séparés de $e_3,\ e_4,\ e_5$ et e_6 par le mot a

– $\equiv_2$ a pour classes d'équivalence $\{e_1,\ e_2\}$, $\{e_3,\ e_4\}$, $\{e_5,\ e_6\}$ et $\{e_7,\ e_8\}$:

puisqu'un mot m de longueur 1 sépare	le mot bm sépare
e_5 de e_7	e_4 de e_6
e_6 de e_8	e_3 de e_5

puisqu'aucun mot de longueur 1 ne sépare	aucun mot de longueur 2 ne sépare
e_7 de e_8 ni e_5 de e_6	e_5 de e_6
e_5 de e_6 ni e_3 de e_4	e_3 de e_4
e_7 de e_8 ni e_1 de e_2	e_7 de e_8
e_3 de e_4 ni e_1 de e_2	e_1 de e_2

– $\equiv_3 = \equiv_2$ (pour les mêmes raisons que juste au dessus, en accroissant de 1 la longueur des mots) donc[1] $\equiv_N = \equiv_2$

ce qui donne l'automate de la figure 7.6 :

Deuxième exemple : minimisons l'automate de la figure 7.7

Voici la construction de la relation :

$\equiv_0$	$\{q_0,\ q_1\}$	$\{q_2,\ q_3,\ q_4,\ q_5,\ q_6,\ q_7\}$			séparés par ϵ
$\equiv_1$	"	$\{q_2,\ q_3,\ q_4,\ q_5\}$		$\{q_6,\ q_7\}$	séparés par a
$\equiv_2$	"	$\{q_2,\ q_3\}$	$\{q_4,\ q_5\}$	"	séparés par ba

1. on verra plus loin, en 7.4.3, la raison de ce *donc*.

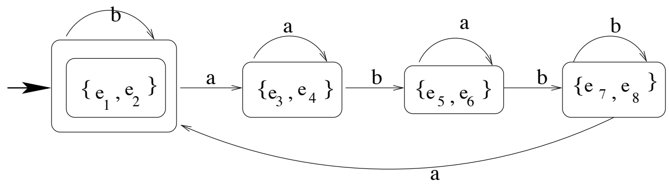


FIGURE 7.6 – Automate minimisé

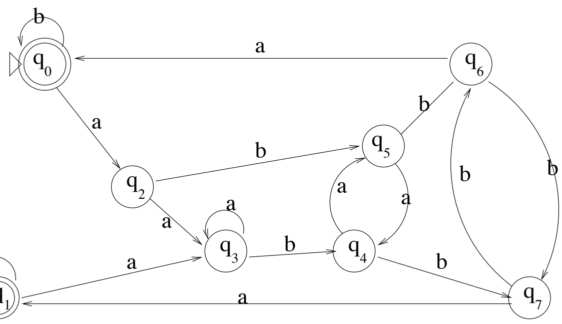


FIGURE 7.7 – Construisons l'équivalence de Nérode

ce qui donne comme résultat la figure 7.8 :

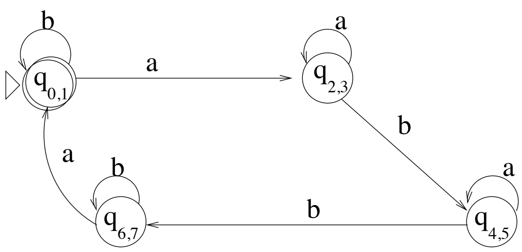


FIGURE 7.8 – Résultat

La façon dont jflap calcule de la relation de Nérode
Soit l'automate de la figure 7.9

L'algorithme suivant calcule la relation de Nérode :

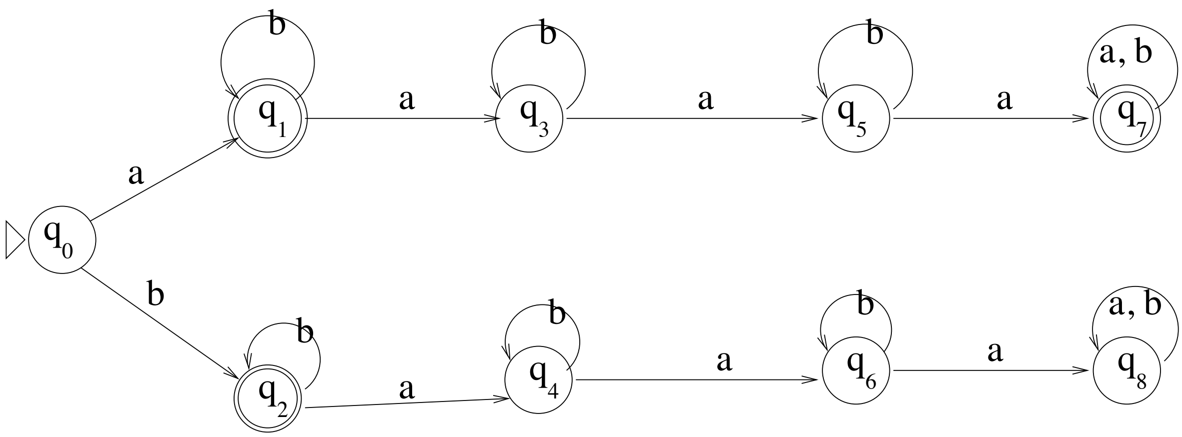


FIGURE 7.9 – Construisons l'équivalence de Nérode

Algorithme : Calcul de la relation de Nérode (technique jflap)

Données : $\mathcal{A} = (\Sigma,\ E,\ i_0,\ F,\ \delta)$ un AFD complet
Résultat : Une partition de E en classes d'équivalence d'états non distinguables

$j \leftarrow 0$;
Initialiser la partition $P_1 \leftarrow \{F,\ E - F\}$;
répéter
  $P_2 \leftarrow \emptyset$;
  pour chaque $C \in P_1$ **faire**
    si $\forall e', e'' \in C,\ \forall \alpha \in \Sigma\ :\ \delta(e', \alpha)$ et $\delta(e'', \alpha)$ sont dans la même partie de la partition P_1 **alors**
      Recopier C dans P_2
    sinon
      choisir $\alpha \in \Sigma$ tel que $\exists e', e'' \in C$ tel que $\delta(e', \alpha)$ et $\delta(e'', \alpha)$ n'appartiennent pas à la même partie de la partition P_1;
      Partitionner C en un nombre minimal n de sous ensembles $C_1,\ C_2, \ldots,\ C_n$ tels que $\forall k \in [1 \ldots n]\ \exists C' \in P_1$ tel que $\forall e \in C_k\ :\ \delta(e, \alpha) \in C'$;
      Recopier $C_1,\ C_2, \ldots,\ C_n$ dans P_2;
    fin
  fin
  $j \leftarrow j + 1$;
jusqu'à $P_1 = P_2$;
Renvoyer P_1;

Voici la trace de l'algorithme : (à chaque fois la lettre α choisie est a)

P_0	$\{q_1,\ q_2,\ q_7\}$			$\{q_0,\ q_3,\ q_4,\ q_5,\ q_6,\ q_8\}$			
P_1	$\{q_1,\ q_2\}$		$\{q_7\}$	$\{q_0,\ q_5\}$		$\{q_3,\ q_4,\ q_6,\ q_8\}$	
P_2	’ ’		’ ’	$\{q_0\}$	$\{q_5\}$	$\{q_3\}$	$\{q_4,\ q_6,\ q_8\}$
P_3	$\{q_1\}$	$\{q_2\}$	’ ’	’ ’	’ ’	’ ’	’ ’

L'arbre construit par jflap à la fin de la construction de la ligne P_2 est donné figure 7.10 : c'est un arbre d'inclusion des parties, étiqueté par une lettre de Σ sur chaque noeud interne.

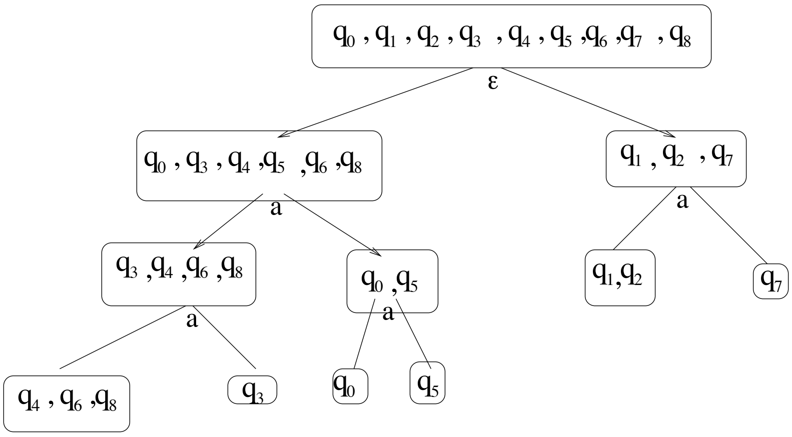


FIGURE 7.10 – jflap ne construit pas exactement $\equiv_i$

7.4.3 Lemme : la suite $(\equiv_i)$ est finie.

$(\equiv_i\ =\ \equiv_{i+1})\ \Rightarrow\ (\forall k \in \mathbb{N}\ \equiv_i\ =\ \equiv_{i+k})$ et donc $(\equiv_i\ =\ \equiv_{i+1})\ \Rightarrow\ (\equiv_i\ =\ \equiv_N)$

Explicitation de $(\equiv_i\ =\ \equiv_{i+1})\ \Rightarrow\ (\equiv_i\ =\ \equiv_N)$

$$(\forall e', e'' \in E\ :\ (e' \equiv_i e'' \iff e' \equiv_{i+1} e''))\ \Rightarrow$$
$$(\forall e', e'' \in E\ :\ (e' \equiv_i e'' \iff e' \equiv_N e''))$$

Démonstration de $(\equiv_i\ =\ \equiv_{i+1})\ \Rightarrow\ (\forall k \in \mathbb{N}\ \equiv_i\ =\ \equiv_{i+k})$

Faisons une démonstration par l'absurde : supposons que $(\equiv_i\ =\ \equiv_{i+1})$ et $\exists k \in \mathbb{N}\ :\ \equiv_i\ \neq\ \equiv_{i+k})$, alors si on prend le plus petit des k possibles on a (avec $j = i + k - 2$) $\equiv_j = \equiv_{j+1}$ et $\equiv_{j+1} \neq \equiv_{j+2}$, c'est à dire

$$\forall e', e'' \in E\ :\ (e' \equiv_{j+1} e'') \iff (e' \equiv_j e'')\ \text{et}$$

$$\exists f', f'' \in E\ :\ (f' \equiv_{j+1} f'')\ \text{et}\ (f' \not\equiv_{j+2} f'')$$

soit en français il existe deux états f' et f'' qui ne sont séparés par aucun mot de longueur $j+1$ mais qui sont séparés par un mot de longueur $j+2$. Mais alors soit un tel mot m de longueur $j+2$ qui sépare f' et f'' et soit α la première lettre de m et n le suffixe de $m = \alpha n$. Soient enfin g' et g'' tels que $f'\ \alpha\ g'$ et $f''\ \alpha\ g''$ soient des transitions de l'automate.
g' et g'' sont séparés par n qui est un mot de longueur $j+1$, mais (comme $\equiv_j = \equiv_{j+1}$) g' et g'' sont séparés par un mot p de longueur j, donc αp (de longueur $j+1$) sépare f' et f''. *Contradiction.*

7.4.4 $\equiv_N = \equiv_{|E|-2}$

$\equiv_0$ possède deux classes ($\overline{A}$ et $\overline{E \setminus A}$) et $\equiv_i \neq \equiv_{i+1}$ implique que $\equiv_{i+1}$ possède au moins une classe de plus que $\equiv_i$, or le nombre maximal de classes est $|E|$, donc le nombre maximum d'itérations est $|E|-2$.

7.5 Minimalité et unicité de l'automate ainsi construit

7.5.1 Définition d'un langage résiduel

Soit Σ un alphabet, L un langage sur cet alphabet et m un mot (appartenant ou non à L) sur cet alphabet. On appelle *résiduel du langage L par rapport à m*, que l'on note $m^{-1}L$ le langage : $m^{-1}L = \{n \in \Sigma^* \mid mn \in L\}$.
Regardons quelques exemples ; on prendra pour chacun de ces exemples
$m = aaaaab$:

1. un langage fini : $L = \{aa,\ aaaaab,\ aaba,\ aaaabba,\ aaaaabba\}$ alors $m^{-1}L = \{\epsilon,\ ba\}$
2. un langage infini simple : L est le langage des mots sur $\{a,\ b\}$ qui ont un nombre pair de a ; alors $m^{-1}L$ est le langage des mots sur $\{a,\ b\}$ qui ont un nombre impair de a.
3. un langage infini plus compliqué : $L = \{m \in \{a,\ b\} \mid |m|_a = |m_b|\}$ alors $m^{-1}L = \{m \in \{a,\ b\} \mid |m|_a = |m_b| - 4\}$
4. dans ce nouvel exemple, on confondra le langage et une expression rationnelle de ce langage ; en particulier, r étant une expression rationnelle et m un mot, on parlera du langage $m^{-1}r$ (et non de $m^{-1}L(r)$.
   Soit $L = (a+b)^*b(a+ab)^*$. On va calculer
   - $a^{-1}L$:
     $L = (a+b)^*b(a+ab)^* = (a+b)(a+b)^*b(a+ab)^* + b(a+b)^* =$
     $$a(a+b)^*b(a+ab)^* + b(a+b)^*b(a+ab)^* + b(a+b)^*$$
     donc $a^{-1}L = L$
   - d'après le même calcul $b^{-1}L = L + (a+ab)^*$
   - et trivialement $(bb)^{-1}L = b^{-1}(b^{-1}L) = b^{-1}(L + (a+ab)^*) = b^{-1}L$
   - le difficile est

$(ba)^{-1}L = a^{-1}(b^{-1}L) = a^{-1}(L + (a+ab)^*) =$
$a^{-1}L + (\epsilon + b)(a+ab)^* = L + (a+ab)^* + (a+ab)^*$
mais $b(a+ab)^* \subset L$ donc $(ba)^{-1}L = b^{-1}L$
Considérons alors l'ensemble $\mathcal{L} = \{L,\ b^{-1}L\}$. On a la propriété :
$\forall \alpha \in \{a,\ b\},\ \forall K \in \mathcal{L}\ :\ \alpha^{-1}K \in \mathcal{L}$ car $a^{-1}L = L \in \mathcal{L}$, $b^{-1}L \in \mathcal{L}$, $a^{-1}(b^{-1}L) = (ba)^{-1}L = b^{-1}L \in \mathcal{L}$, $b^{-1}(b^{-1}L) = b^{-1}L \in \mathcal{L}$
donc par récurrence sur $|m|$: $\forall m \in \{a,\ b\}^*,\ \forall K \in \mathcal{L}\ :\ m^{-1}K \in \mathcal{L}$:
- $|m| = 0 \Rightarrow m^{-1}K = K$
- $|m| = 1$ c'est la propriété vue ci-dessus
- $(m\alpha)^{-1}K = \alpha^{-1}(m^{-1}K)$ et $K' = m^{-1}K \in \mathcal{L}$ (hypothèse de récurrence) donc $\alpha^{-1}K' \in \mathcal{L}$ (vu ci dessus).

7.5.2 Théorème : un langage rationnel possède un nombre fini de résiduels.

Dans les exemples ci dessus :

1. pour un langage fini L, l'ensemble des résiduels est $\{m^{-1}L \mid m \in Prefs(L)\}$ où $Prefs(L)$ est l'ensemble des préfixes des mots de l.
2. si L est le langage des mots sur $\{a,\ b\}$ qui ont un nombre pair de a alors l'ensemble des résiduels de L est composé de deux langages : celui des mots sur $\{a,\ b\}$ qui ont un nombre impair de a et celui des mots sur $\{a,\ b\}$ qui ont un nombre pair de a.
3. $L = \{m \in \{a,\ b\} \mid |m|_a = |m_b|\}$ possède une infinité de résiduels : pour $z \in \mathbb{Z}$, $L_z = \{m \in \{a,\ b\} \mid |m|_a - |m_b| = z\}$ est un résiduel.

Démonstration
Soit L un langage rationnel et $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ un automate fini déterministe complet monogène qui reconnaisse L.
Pour tout état e de E, définissons le langage $L_e = \{m \in \Sigma^* \mid \delta^*(e,m) \in F\}$, autrement dit le langage reconnu par l'automate déterministe $\mathcal{A}_e = (\Sigma,\ E,\ e,\ F,\ \delta)$.
- $\forall e \in E$ L_e est un résiduel : $L_e = m^{-1}L$ pour tout m reconnu par l'automate $(\Sigma,\ E,\ i,\ \{e\},\ \delta)$.
- si M est un résiduel de L, alors il existe un état e de E tel que $M = L_e$: M est un résiduel de L si et seulement si $\exists m \in \Sigma^*$ tel que $M = m^{-1}L$. Mais puisque $\mathcal{A}$ est complet, $\delta^*(i,m)$ définit un état e de E, et $M = L_e$.

Comme il y a un nombre fini d'états, il y a un nombre fini de résiduels.

7.5.3 Corollaire sur le nombre minimum d'états d'un automate déterministe

Si un langage possède un nombre fini r de résiduels, alors il il n'existe pas d'automate complet de moins de r états qui le reconnaisse.
La démonstration du théorème ci dessus montre qu'un langage reconnu par un automate complet de p états possède *au plus* p résiduels.

7.5.4 Théorème réciproque

Si un langage possède un nombre fini r de résiduels, alors il existe un AFD complet unique (au nom des états près) de r états qui le reconnaît.

début de l'exemple : pour $L = (a+b)^*b(a+ab)^*$on a vu que l'ensemble des résiduels est $\{L, b^{-1}L\}$ avec $a^{-1}L = L$ et $b^{-1}(b^{-1}L) = a^{-1}(b^{-1}L) = b^{-1}L$.

Démonstration

Soit $L \subseteq \Sigma^*$ un langage ayant un nombre fini de résiduels, autrement dit tel que l'ensemble des résiduels $\{u^{-1}L \mid u \in \Sigma^*\}$ est fini. Si à chaque résiduel $R_u = u^{-1}L$ on fait correspondre un état noté e_u, l'ensemble E des états ainsi défini est fini. En posant $R_\epsilon = L = \epsilon^{-1}L$,on note e_ϵ l'état correspondant.
Définissons alors l'automate dont on démontrera en TD qu'il reconnaît L :
$\mathcal{R} = (\Sigma,\ E,\ e_\epsilon,\ F,\ \delta)$ avec $F = \{e_u \mid \epsilon \in u^{-1}L\}$ et
$\delta = \{e_u\alpha e_{u\alpha} \mid \alpha \in \Sigma,\ e_u \in E\}$.

Et il y a bijection entre les résiduels de L et les états de $\mathcal{R}$.

fin de l'exemple : comme $\epsilon \in b^{-1}L$ et $\epsilon \notin L$ on obtient l'automate 7.11

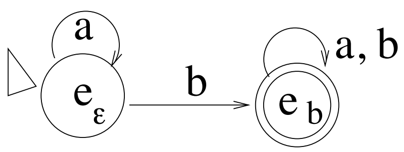


FIGURE 7.11 –

7.5.5 Conclusion : unicité de l'automate minimum

$\forall L$ langage rationnel, il existe un unique (au nom des états près) automate fini déterministe complet avec un nombre minimum d'états qui reconnaît L.

Corollaire Donc, étant donnés deux automates finis quelconques, on peut savoir s'ils reconnaissent le même langage.

Démonstration de ce que l'automate construit par la relation de Nérode possède un nombre minimal d'états

Soit L un langage rationnel, $\mathcal{A} = (\Sigma,\ E,\ i,\ F,\ \delta)$ un automate déterministe complet qui reconnaît L, et $\equiv_N$ la relation de Nérode associée à $\mathcal{A}$.
Pour tout état $e \in E$ appelons L_e le langage reconnu par l'automate $(\Sigma,\ E,\ e,\ F,\ \delta)$.

$\forall e, e' \in E$ $L_e = L'_e$ si et seulement si $e \equiv_N e'$.
D'où la bijection entre les états de l'automate quotient et les résiduels de L : à tout état correspond un seul résiduel (et évidemment un).

Chapitre 8

Lemme de la pompe

8.0.6 Introduction

On va démontrer qu'on peut définir des langages qui ne sont pas réguliers.

Principe de la démonstration

Pour démontrer ce résultat, on va énoncer une condition nécessaire sur un langage pour que celui-ci soit régulier, puis on exhibera des langages qui ne vérifient pas cette condition.

Résumé de la condition nécessaire

Si un automate reconnaît un langage infini, il doit pouvoir reconnaître des mots d'une longueur arbitrairement grande, et en particulier d'une longueur supérieure au nombre d'états de cet automate.
Donc de tels mots *doivent passer dans un circuit*, circuit dans lequel on peut faire un nombre de passages arbitraire.
Autrement dit, pour tout langage rationnel L, il existe un entier k_L [1] tel que tout mot $m \in L$ de longueur $|m| \geq k_L$ peut être factorisé (au moins d'une façon) en $m = m_d m_B m_f$ avec $m_B \neq \epsilon$ et $\forall l \in \mathbb{N} \ : \ m_d m_B^l m_f \in L$.

Utilisation

Pour prouver qu'un langage L n'est pas régulier, il faudra donc exhiber une famille $\{m_k \ , \ k \in \mathbb{N}\}$ telle que $|m_k|$ soit une fonction strictement croissante de k et telle que $\forall k \in \mathbb{N}$, $\forall m_1, \ m_2, \ m_3$ tels que $m_k = m_1 m_2 m_3$ et $m_2 \neq \epsilon$, $\exists l \in \mathbb{N}$ tel que $m_1 m_2^l m_3 \notin L$.

1. qui correspond au nombre d'états d'un automate qui reconnaît L

Exemple

Soit $L = \{a^n b^n \mid n \in \mathbb{N}\}$. Ce langage n'est pas algébrique : en posant $\forall k \in \mathbb{N} \ : \ m_k = a^k b^k$ on a évidemment $\forall k \in \mathbb{N} \ : \ m_k \in L$ et $\forall m_1, \ m_2, \ m_3$ tels que $m_k = m_1 m_2 m_3$ et $m_2 \neq \epsilon$

- soit $|m_2|_a \neq |m_2|_b$ mais alors $m_1 m_2^2 m_3 \notin L$ car $m_1 m_2^2 m_3$ n'a pas autant de a et de b,
- soit $|m_2|_a = |m_2|_b \neq 0$ mais alors $m_1 m_2^2 m_3 \notin L$ car $m_1 m_2^2 m_3$ a une occurrence de a après une occurrence de b.

8.0.7 Enoncés

Lemme de la pompe : version de base

L n'est pas régulier des que pour tout $k \in \mathbb{N}$ il existe $m_k \in L$ tel que
- $|m_k| \geq k$ et
- $\forall m_d, \ m_B, \ m_f$ tels que $m = m_d m_B m_f$ et $m_B \neq \epsilon$
il existe $l \in \mathbb{N}$ tel que $m_d m_B^l m_f \notin L$

Lemme de la pompe : version améliorée 1

m_d sera le préfixe de m_k au moment où on atteint le premier état du circuit et m_B sera le facteur correspondant à un seul passage dans le circuit. Donc quelle que soit la taille de m_k, celle de $m_d m_B$ est limitée par le nombre k d'états de l'automate. C'est ce que dit cette version du lemme :

L n'est pas régulier dés que pour tout $k \in \mathbb{N}$ il existe $m_k \in L$ tel que
- $|m_k| \geq k$
- $\forall m_d, \ m_B, \ m_f$ tels que $m = m_d m_B m_f$ et $m_B \neq \epsilon$ et $|m_d m_B| \leq k$
il existe $l \in \mathbb{N}$ tel que $m_d m_B^l m_f \notin L$

Lemme de la pompe : version améliorée 2

m_B sera le facteur correspondant au dernier passage dans le circuit et m_f sera le suffixe de m_k après que l'on aie fini de passer dans le circuit. Donc quelle que soit la taille de m_k, celle de $m_B m_f$ est limitée par le nombre k d'états de l'automate. C'est ce que dit cette version du lemme :
L n'est pas régulier des que pour tout $k \in \mathbb{N}$ il existe $m_k \in L$ tel que
- $|m_k| \geq k$
- $\forall m_d, \ m_B, \ m_f$ tels que $m = m_d m_B m_f$ et $m_B \neq \epsilon$ et $|m_B m_f| \leq k$
il existe $l \in \mathbb{N}$ tel que $m_d m_B^l m_f \notin L$

Exemple d'utilisation des versions améliorées

Soit le langage
$L = \{m \in \{a,b\}^* \mid |m|_a = |m|_b$ et $\forall i \in [1 \ldots |m|] \ : \ |m[1 \ldots i]|_a \geq |m[1 \ldots i]|_b\}$.
Ce langage n'est toujours pas régulier mais cette fois ci une occurrence de b peut être suivie d'une occurrence de a, et c'est pour cela que la démonstration

de l'exemple précédent n'est plus valide.
Posons toujours $\forall k \in \mathbb{N} \ : \ m_k = a^k b^k$ on a évidemment $\forall k \in \mathbb{N} \ : \ m_k \in L$ et $\forall m_1, \ m_2, \ m_3$ tels que $m_k = m_1 m_2 m_3$ et $m_2 \neq \epsilon$ et $m_1 m_2 \leq k$, m_2 qui n'est composé que de a, donc $m_1 m_2^2 m_3 \notin L$ donc d'après la première version améliorée [2] L n'est pas régulier.

2. on pourrait utiliser la deuxième version avec la même famille m_k en n'examinant que les décompositions telles que $|m_2 m_3| \leq k$.