Théorie des langages

October 12, 2008

1 Introduction

La théorie des langages s'est développée dans les années 1950-60 avec les travaux de chercheurs tels que Noam Chomsky, Marcel-Paul Schützenberger,... Elle se situe à une intersection de la linguistique, des mathématiques et de l'informatique. Mais les structures qui y ont été définies intéressent aussi d'autres disciplines (biologie,...).

En linguistique, cette discipline apporte un cadre formel à l'étude du langage naturel. En informatique, elle a permis de formaliser la conception des langages informatiques, améliorant ainsi l'analyse syntaxique et la compilation des programmes. On peut aussi s'en servir pour la recherche de motifs, la représentation de succession d'événements, la croissance de structures....

1.1 Présentation intuitive des grammaires

Une grammaire est constituée de

- un ensemble fini de symboles terminaux, ou alphabet (terminal), noté X,
- un ensemble fini de variables (ou non-terminaux), noté V,
- un ensemble fini de règles de dérivation, noté R,
- un axiome (parfois un ensemble d'axiomes) qui est une variable particulière, parfois noté S mais dont la notation dépendra de la grammaire.

Grossièrement, une règle de dérivation est une transformation dans laquelle on indique qu'une variable peut être remplacée par une certaine succession de symboles terminaux et de variables.

1.1.1 Exemple : langue naturelle

La grammaire française serait constituée de

- X, les mots de la langue française,
- V, les concepts grammaticaux,
- R, les règles de construction grammaticale,
- un axiome P, correspondant au concept de *Phrase*.

On aurait
$X = \{abaca, abaisse, abaisse{-}langue, ..., kysteux, kystique, la, là, labadens, labarum, ...lazzi, le, lé, leader...\}$ qui contiendrait aussi toutes les formes accordées, toutes les formes conjuguées, etc...

$V = \{P, Gn, Gv, Adv, Adj, ...\}$ pour représenter les concepts de *Phrase*, *Groupe Nominal*, *Groupe Verbal*,...

et $R = \{P \to Gn\ Gv, P \to Gn\ Gv\ Gc, ..., Gn \to D\ N, Gn \to D\ N\ Adj, ..., D \to la, D \to le, D \to les, ...N \to jour, N \to nuit, ..., Adj \to bleu, Adj \to blanche, Adj \to éteinte, ...\}$ pour représenter les règles grammaticales, comme on dit *faites des phrases du style "sujet verbe complément"*.

Cette grammaire permet de dériver les phrases de la langue française.
$P \Rightarrow Gn\ Gv\ Gc \Rightarrow D\ N\ Gv\ Gc \Rightarrow la\ N\ Gv\ Gc \Rightarrow la\ N\ est\ Gc\ .... \Rightarrow la\ nuit\ est\ éteinte$

Attention : par la suite, les termes *lettre*, *mot* seront utilisés avec une signification différente de celle sous-entendue par cet exemple. En théorie des langages, *lazzi* appartient à l'alphabet X ; il faudrait donc dire que *lazzi* est une lettre de l'alphabet ! Mais il n'y aura pas de confusion par la suite, car nous travaillerons avec un alphabet abstrait réduit à quelques lettres.

1.1.2 Exemple : langage informatique

Tous les langages informatiques récents ont une syntaxe définie par une grammaire. Le langage Turbo-Pascal était défini par une grammaire constituée de

- X, contenant les lettres et chiffres, des caractères spéciaux, les mots réservés,
- V, les notions du langage de programmation,
- R, les règles syntaxiques,
- un axiome correspondant à la notion de *Programme*.

Les règles suivantes, appartenant donc à R, sont extraites du *Guide du programmeur* :

$Programme \rightarrow Entête$ **;** $Bloc$ **.**
$Programme \rightarrow Entête$ **;** $ClausesUses\ Bloc$ **.**
$Bloc \rightarrow Déclaration$ **BEGIN** $Instructions$ **END**
$Instruction \rightarrow InstructionSimple$
$Instruction \rightarrow InstructionComposée$
$Instructions \rightarrow Instruction$
$Instructions \rightarrow Instruction$ **;** $Instructions$
....

Les termes en gras **.**, **;**, **BEGIN**, **END**,... appartiennent à l'alphabet X du langage Turbo-Pascal.

Cette grammaire permet de dériver tous les programmes informatiques que l'on peut écrire en Turbo-Pascal. Leur nombre étant infini, cette grammaire n'est pas utilisée pour générer des programmes mais pour savoir si un programme est correctement écrit. C'est ce qu'on appelle *l'analyse syntaxique*.

Mieux encore, la grammaire est conçue de telle façon qu'à la reconnaissance de l'emploi de certaines règles, on peut associer leur traduction en langage machine (langage exécutable par les circuits de l'ordinateur) ; c'est la *compilation*.

1.1.3 Exemple : langage XML

Comme dit précédemment, tous les langages informatiques récents ont une syntaxe définie par une grammaire. La syntaxe de XML est définie dans une recommandation du *W3C* (voir `http://www.w3.org/TR/REC-xml/`).

De plus, des contraintes imposées par certaines règles garantissent un traitement efficace du document XML (voir `http://www.w3.org/TR/REC-xml/#determinism`).

Les règles suivantes pourraient être extraites de la recommandation (les caractères en gras sont des symboles terminaux) :

$document \rightarrow prologue\ élément$
$document \rightarrow prologue\ élément\ diverss$
$diverss \rightarrow divers$
$diverss \rightarrow divers\ diverss$
...
$prologue \rightarrow XMLdecl$
$prologue \rightarrow XMLdecl\ docTypeDecl$
$prologue \rightarrow XMLdecl\ diverss\ docTypeDecl$
...
$docTypeDecl \rightarrow$ **<!DOCTYPE** $S\ nom\ S\ IdExterne$ **>**
$docTypeDecl \rightarrow$ **<!DOCTYPE** $S\ nom\ S\ IdExterne\ S$ (**[** $intSubset$ **]** S) **>**
...
$S \rightarrow s$
$S \rightarrow s\ S$
$s \rightarrow$ **#x20**
$s \rightarrow$ **#x9**
$s \rightarrow$ **#xD**
$s \rightarrow$ **#xA**
....

#x20 est le code héxadécimal de l'*espace*, **#xD** est le code héxadécimal du *carriage return*, **#xA** est le code héxadécimal du *line feed* et **#x9** est le code héxadécimal de la *tabulation*.

Les règles de la recommandation utilisent des symboles permettant une écriture plus concise des règles. Dans la recommandation, vous trouverez la règle *document* → *prologue élément divers** qui est équivalente aux quatre premières règles ci-dessus et la règle $S \to (\#\mathbf{x20}|\#\mathbf{x9}|\#\mathbf{xD}|\#\mathbf{xA})^+$ qui est équivalente aux six dernières règles ci-dessus.

Dans la grammaire définissant XML, il est précisé que certaines variables engendrent des sous-langages particuliers nommés *expressions régulières*. Les règles, à partir de ces variables, ont la forme la plus simple dans la *hiérarchie de Chomsky*. Nous commencerons par étudier ces grammaires.

1.1.4 Exemple : DTD

Quand vous définissez une DTD, vous définissez une grammaire. La DTD suivante

```
<!ELEMENT événements (ville,événement+)+ >
<!ELEMENT événement (titre,résumé,renseignements) >
<!ELEMENT renseignements (horaire,tarif,téléphone?,site?)  >
<!ELEMENT horaire (#PCDATA) >
...
```

pourrait se traduire en termes de règles

événements → **<événements>** contenuEvénements **</événements>**
contenuEvénements → ville événementRépété
contenuEvénements → ville événementRépété contenuEvénements
événementRépété → événement
événementRépété → événement événementRépété
...
événement → **<événements>** contenuEvénement **</événements>**
contenuEvénement → titre résumé renseignements
renseignements → **<renseignements>** contenuRenseignements **</renseignements>**
contenuRenseignements → horaire tarif
contenuRenseignements → horaire tarif téléphone
...
horaire → **<horaire/>**
horaire → **<horaire>** PCDATA **</horaire>**
PCDATA → caractère
PCDATA → caractère PCDATA
caractère → **a**
caractère → **b**
....

qui décriraient tous les documents XML que l'on peut écrire en respectant la DTD.

Là encore, vous remarquez que les caractères `?`, `*`, `+`... vous permettent d'écrire les règles de façon plus concise.

2 Préliminaires

2.1 Définitions

On notera X l'alphabet des symboles terminaux. Il est aussi très souvent noté Σ, A...

Exemple: $X = \{a, b\}$ ou $X = \{0, 1, 2\}$. ♢

La concaténation est l'opération qui permet de construire des mots. La concaténation de a et b est notée $a.b$ quand on veut vraiment insister sur sa présence ; mais elle sera souvent notée sans aucun symbole, ab tout simplement. On notera $X^2 = XX$, $X^3 = XXX = X^2X$,... ,$X^n = X^{n-1}X$.

X^2 est donc l'ensemble des mots de deux lettres, X^3 est donc l'ensemble des mots de trois lettres, ... écrits avec l'alphabet X.

Le mot vide ε est le mot sans lettre, élément neutre de la concaténation ; si $u \in X^n$, on a $u\varepsilon = \varepsilon u = u$. On a $X^0 = \{\varepsilon\}$.

On pose $X^* = \bigcup_{i\geq 0} X^i$ et $X^+ = \bigcup_{i\geq 1} X^i$. L'ensemble X^* est donc l'ensemble de tous les mots, y compris le mot vide, qu'on peut écrire sur X. L'ensemble X^+ est l'ensemble de tous les mots, sauf le mot vide ; on a $X^+ = X^* \setminus \{\varepsilon\}$.

La concaténation est une opération associative, c'est à dire $(uv)w = u(vw) = uvw$ pour tous mots $u, v, w \in X^*$.

Donc X^*, muni de la concaténation, est un *monoïde* ; c'est à dire un ensemble muni d'une loi interne associative, ayant un élément neutre.

Remarque : un groupe est un monoïde dans lequel chaque élément possède un inverse. L'ensemble des entiers relatifs, muni de l'addition, forme un groupe.

Ce monoïde est dit *libre* parce que la décomposition en éléments de X est unique. L'ensemble des entiers relatifs, muni de l'addition, n'est pas libre car $2 + 3 = 3 + 2$.

Avant d'aller plus loin, prenons quelques exemples 'concrets' :

Exemple: Si $X = \{a, b\}$, alors $X^* = \{varepsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\}$
Si $X = \{a\}$, alors $X^* = \{varepsilon, a, aa, aaa, aaaa, ...\}$ ♢

Un **langage** est un sous-ensemble de X^*.

Exemple: $L = \{\text{mots commençant par } b\} = \{b, ba, bb, baa, bab, bba, bbb, baaa, ...\}$. ♢

On définit la concaténation de langages : $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$

Exemple: Si $L_1 = \{varepsilon, a, ab\}$ et $L_2 = \{\varepsilon, b, aa\}$, alors $L_1L_2 = \{varepsilon, a, b, aa, ab, aaa, abb, abaa\} = \{varepsilon, a, b, a^2, ab, a^3, ab^2, aba^2\}$
On remarquera que le mot ab a plusieurs décompositions $a.b$ et $ab.\varepsilon$. ♢

On définit aussi les puissances d'un langage : $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$,...., $L^* = \bigcup_{i\geq 0} L^i$ et $L^+ = \bigcup_{i\geq 1} L^i$.

Enfin, on parlera de *familles de langages* pour l'ensemble de tous les langages qui vérifie une propriété donnée. Par exemple, on parlera de la famille des langages réguliers pour l'ensemble des langages engendrés par les grammaires régulières.

Exercice 1. *Soit $L_1 = \{ab\}$. Ecrivez les cinq premiers mots, dans l'ordre alphabétique, de L_1^*.*
Soit $L_2 = a^*$. Ecrivez les cinq premiers mots de L_2^*.
Soit $L_3 = \{a, bb\}$. Ecrivez les cinq premiers mots de L_3^*.

2.2 Grammaires

Une grammaire est un quadruplet $G = (X, V, R, S)$ où

- X est l'alphabet des symboles terminaux,
- V est l'ensemble fini des variables,
- R est l'ensemble fini des règles de dérivation,
- S est l'axiome appartenant à V.

Exemple: $G_1 = (X, V, R, S)$ avec
$X = \{a, b\}$
$V = \{S, T\}$
$R = \{S \to aSb, S \to T, T \to bTa, T \to \varepsilon\}$ ♢

Une *dérivation simple* est l'application d'une règle dans un contexte particulier. On notera $ulv \Rightarrow urv$ si $l \to r$ est une règle de R.

On notera $w \overset{*}{\Longrightarrow} w'$ une succession de dérivations.

Exemple: Dans la grammaire G_1 précédente, $baaSaSbTa \overset{*}{\Longrightarrow} baaTaSbbbTaaa$ car
$baaSaSbTa \Rightarrow baaSaSbbTaa \Rightarrow baaSaSbbbTaaa \Rightarrow baaTaSbbbTaaa$ ♢

On appelle *langage engendré par* G, noté $L(G)$, l'ensemble $L(G) = \{w \in X^* \mid S \stackrel{*}{\Longrightarrow} w\}$.

Dans $L(G)$, tout mot est obtenu par une dérivation, à partir de S. Inversement, tout mot obtenu par une dérivation terminale à partir de S, c'est à dire mot qui ne contient que des symboles de l'alphabet X, appartient à $L(G)$.

Exemple: $L(G_1) = \{a^n b^m a^m b^n \mid n, m \in \mathbb{N}\}$ $\diamondsuit$

Remarquons qu'il est possible qu'un mot admette plusieurs dérivations. Il est possible aussi qu'un langage admette plusieurs grammaires.

Exemple: $L(G_1) = L(G_2)$ avec $G_2 = (X, V, R', S)$ où $R' = \{S \to aSb, S \to \varepsilon, S \to bTa, T \to bTa, T \to \varepsilon\}$ $\diamondsuit$

Exercice 2. *$G_3 = (X, V, R, S)$ avec*
$X = \{a, b\}$
$V = \{S, T_1, T_2\}$
$R = \{S \to T_1T_2, T_1 \to aT_1b, T_1 \to \varepsilon, T_2 \to bT_2a, T_2 \to \varepsilon\}$
Est-ce que $ab \in L(G_3)$?
Est-ce que $abbbaa \in L(G_3)$?
Est-ce que $abbaaa \in L(G_3)$?
Trouver d'autres mots de $L(G_3)$.
Quel est le langage engendré par G_3 ?

3 Hiérarchie de Chomsky

Noam Chomsky a défini des familles de langages en définissant des caractéristiques des règles de dérivation.

Grammaires régulières
$R \subseteq ((V \times X^*) \cup (V \times X^*V))$.
Les règles sont donc de la forme $l \to r$ avec $l \in V$ et $r \in X^*$ ou $r \in X^*V$.

Grammaires algébriques (context-free)
$R \subseteq (V \times (X + V)^*)$.
Les règles sont donc de la forme $l \to r$ avec $l \in V$ et $r \in (X + V)^*$.

Grammaires contexte lié
$R \subseteq ((X + V)^n \times (X + V)^m)$ avec $m \geq n$.
Les règles sont donc de la forme $l \to r$ avec $l, r \in (X + V)^*$ avec $|l| \leq |r|$, où $|l|$ représente la longueur de l.

Grammaires à structure de phrase
$R \subseteq ((X + V)^* \times (X + V)^*)$.
Les règles sont donc de la forme $l \to r$ avec $l, r \in (X + V)^*$, sans contrainte.

Un langage est d'un type donné s'il existe une grammaire de ce type qui l'engendre ; la famille des langages réguliers est la famille des langages engendrés par les grammaires régulières...

De part leur définition, chaque famille est incluse dans la famille suivante ; la famille des langages réguliers est incluse dans la famille des langages algébriques... Tout langage régulier est donc algébrique...

4 Famille des langages réguliers, famille des langages rationnels, famille des langages reconnaissables

4.1 Famille des langages réguliers

Une grammaire $G = (X, V, R, S)$ est *régulière* si et seulement si $R \subseteq ((V \times X^*) \cup (V \times X^*V))$.

Un langage est régulier si et seulement si il existe une grammaire régulière qui l'engendre.

On notera $Reg(X^*)$ la famille des langages réguliers sur X^*.

Exemple: Le langage $L_1 = \{$mots commençant par a et finissant par $b\}$ est un langage régulier car on peut trouver une grammaire régulière qui l'engendre.

On a $L(G_1) = L_1$ avec $G_1 = (X, V, R, S)$ où $R = \{S \to aT, T \to aT, T \to bT, T \to b\}$. Cette grammaire G_1 est régulière car les règles ont la forme caractéristique de ces grammaires.

Pour concevoir une grammaire, il faut associer une sémantique aux variables. Dans une grammaire régulière, la succession des variables agit un peu comme une tête rotative d'imprimante, écrivant un caractère et se tournant pour le caractère suivant.

Dans la grammaire G_1, on pourrait associer à S la signification *la tête d'imprimante est au début du texte* ; la variable T signifierait *la tête d'imprimante a écrit le a du début, elle doit écrire n'importe quel caractère et arrêter sur un b.* ♢

Exercice 3. *Prouver que le langage $L_2 = \{$mots contenant au moins un $a\}$ est un langage régulier en concevant une grammaire régulière qui l'engendre.*

Exercice 4. *Prouver que le langage $L_3 = \{$mots contenant un nombre pair de $a\}$ est un langage régulier.*

4.2 Famille des langages rationnels

Un langage est *rationel* si et seulement si il est obtenu par union, concaténation et étoile de langages finis.

On notera $Rat(X^*)$ la famille des langages rationnels sur X^*.

Un langage rationnel est décrit par une expression rationnelle, faisant intervenir des langages finis et les opérateurs + (union), . (concaténation) et * (étoile). L'union est parfois notée par son symbole habituel $\cup$. Le symbole de concaténation est souvent omis.

Exemple: Le langage $L_1 = \{$mots commençant par a et finissant par $b\}$ est décrit par l'expression $L_1 = a(a+b)^*b$. Il est donc rationnel. ♢

Il peut exister plusieurs expressions rationnelles pour un même langage.

Exercice 5. *Prouver que le langage $L_2 = \{$mots contenant au moins un $a\}$ est un langage rationnel en concevant une expression rationnelle qui le décrit.*

Exercice 6. *Prouver que le langage $L_3 = \{$mots contenant un nombre pair de $a\}$ est un langage rationnel.*

4.3 Famille des langages reconnaissables

Un automate (à états) fini est un quintuplet $M = (X, Q, Q_I, Q_F, \delta)$ où

- X est l'alphabet,
- Q est l'ensemble fini des états,
- Q_I est l'ensemble fini des états initiaux, $Q_I \subseteq Q$,
- Q_F est l'ensemble fini des états finals, $Q_F \subseteq Q$,
- δ est l'ensemble fini des transitions, $\delta \subseteq Q \times X \times Q$.

On étend les transitions à la succession de transitions $\delta_* \subseteq Q \times X^* \times Q$ de la façon suivante :

- quel que soit $q \in Q$, on a $(q, \varepsilon, q) \in \delta_*$,
- $\delta \subseteq \delta_*$,
- pour tout $(q, u, q) \in \delta_*$ et $(q, v, q) \in \delta_*$, on a $(q, uv, q) \in \delta_*$

Un mot w est reconnu par l'automate si et seulement si il existe un état initial $q_i \in Q_I$ et un état final $q_f \in Q_F$ tel que $(q_i, w, q_f) \in \delta_*$.

Le langage reconnu par un automate M est l'ensemble de tous les mots qui sont reconnus par M ; il est défini par $L(M) = \{w \in X^* \mid \exists q_i \in Q_I, \exists q_f \in Q_F$ tel que $(q_i, w, q_f) \in \delta_*\}$

Un langage L est *reconnaissable* si et seulement si il existe un automate M tel que $L(M) = L$.

On notera $Rec(X^*)$ la famille des langages reconnaissables sur X^*.

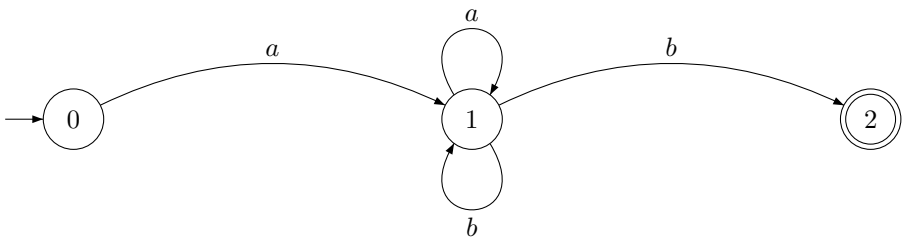

Figure 1: Un automate M_1 reconnaissant L_1

On notera parfois les transitions sous la forme d'une fonction $\delta(q, x) = \{q' \mid (q, x, q') \in \delta\}$.

Les automates ont une représentation graphique, très très pratique.

Exemple: Le langage L_1 est reconnu par l'automate M_1 suivant :

♢

Exercice 7. *Prouver que le langage $L_2 = \{$mots contenant au moins un $a\}$ est un langage reconnaissable en dessinant un automate qui reconnaît ses mots.*

Exercice 8. *Prouver que le langage $L_3 = \{$mots contenant un nombre pair de $a\}$ est un langage reconnaissable.*

Un automate est déterministe si et seulement si

- $||Q_I|| = 1$,
- pour tout $q \in Q$, pour tout $x \in X$, on a $||\delta \cap (\{q\} \times \{x\} \times Q)|| \leq 1$

La deuxième condition peut se lire :
pour tout $q \in Q$, pour tout $x \in X$, il existe au plus un $q' \in Q$ tel que $(q, x, q') \in \delta$.

Autrement dit, il y a un seul état initial et, partant de n'importe quel état, il existe au plus une transition labellée par une lettre donnée.

Exemple: L'automate M_1 n'est pas déterministe.
Remarquons néanmoins que tout mot n'est reconnu que par un seul chemin. Cette notion correspond à la *non ambiguïté.* ♢

Il est possible que certains états, s'ils sont atteints, ne permettent pas de rejoindre un état final. On les appelle *état puits.*

Un automate est *complètement spécifié* si, pour tout état $q \in Q$, pour toute lettre $x \in X$, il existe exactement une transition $(q, x, q') \in \delta$. Pour rendre un automate complètement spécifié, il est parfois nécessaire d'ajouter un état puits.

Exemple: Le langage L_1 est reconnu par l'automate déterministe M_1' suivant :
L'automate n'est pas complètement spécifié. Le langage L_1 est reconnu par l'automate déterministe complètement spécifié M_1'' suivant :

♢

Théorème 1. *Tout langage reconnaissable peut être reconnu par un automate déterministe.*

Pour un langage donné, il existe plusieurs automates déterministes qui le reconnaissent.

Théorème 2. *Pour tout langage reconnaissable, il existe un unique automate déterministe minimal (ayant le nombre minimum d'états) qui le reconnaît.*

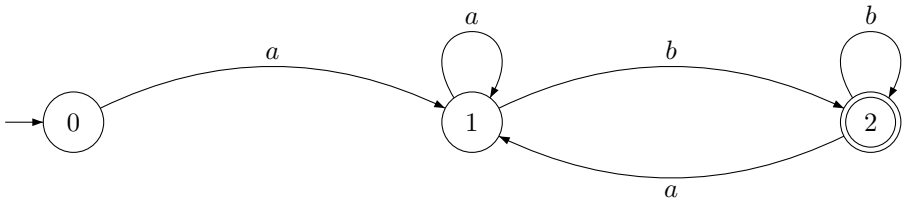

Figure 2: Un automate déterministe M_1' reconnaissant L_1

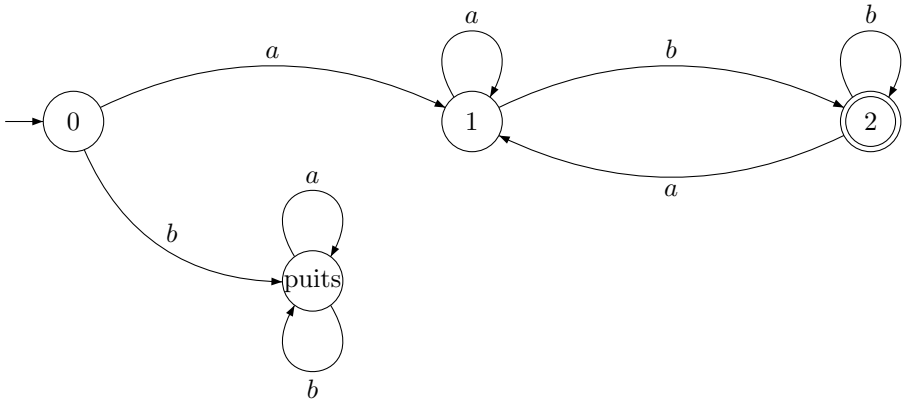

Figure 3: Un automate déterministe complètement spécifié M_1'' reconnaissant L_1

4.4 Equivalence des familles régulières, reconnaissables et rationnelles

Les trois familles que nous venons de présenter sont équivalentes.

Théorème 3. $Reg(X^*) = Rat(X^*) = Rec(X^*)$

Donc, cette famille de langages peut être appelée indifféremment rationnelle, régulière ou reconnaissable. Et pour décrire un langage, on peut donner une expression rationnelle, une grammaire régulière ou un automate fini.

Comme souvent, les égalités se démontrent par des double inclusions. Certaines des inclusions sont simples.

$Rec(X^*)$ contient les langages finis. Il est simple de construire (avec des epsilon-transitions) des automates reconnaissant les unions, concaténation et étoile d'automates. Donc $Rat(X^*)$ est inclus dans $Rec(X^*)$.

Soit M un automate déterministe. A chaque état q, on associe une variable S_q ; l'état initial q_0 sera l'axiome S_{q_0}. A chaque transition $(q, x, q') \in \delta$, on associe une règle $S_q \to xS_{q'}$. A chaque état final S_{q_f}, on associe une règle $S_{q_f} \to \varepsilon$. Il n'est pas trop difficile de vérifier que le langage engendré par la grammaire ainsi obtenue, est identique au langage reconnu par l'automate. Donc $Rec(X^*)$ est inclus dans $Reg(X^*)$.

Si on transforme préalablement la grammaire de telle façon que $R \subseteq ((V \times \{\varepsilon\}) \cup (V \times XV))$, la construction inverse est possible. La transformation de la grammaire n'est pas difficile, donc $Reg(X^*)$ est inclus dans $Rec(X^*)$.

D'autres inclusions sont plus techniques.

4.4.1 De l'automate à l'expression rationnelle

On va associer un système d'équations à l'automate. Résoudre ce système nous donnera le langage reconnu par chaque état.

Pour chaque transition $(q_i, x, q_j) \in \delta$, on considère que le langage reconnu à partir de l'état q_i, c'est la lettre x suivie du langage reconnu à partir de l'état q_j. Cette transition donne l'équation $L_i = xL_j$.

Pour un état final, l'équation fait apparaître ε car le langage reconnu à partir d'un état final contient ε.

Le langage qui nous intéresse est celui *reconnu à partir des états initiaux*. On utilise le lemme suivant pour résoudre le système d'équations.

Lemme 1. *L'équation sur les langages $Z = BZ + A$ d'inconnue $Z \subseteq X^*$, de paramètres $A \subseteq X^*$ et $B \subseteq X^*$ a pour plus petite solution $Z = B^*A$. Cette solution est unique si ε n'appartient pas à B.*

Exemple: Le langage L_1 est reconnu par l'automate déterministe M_1 suivant :

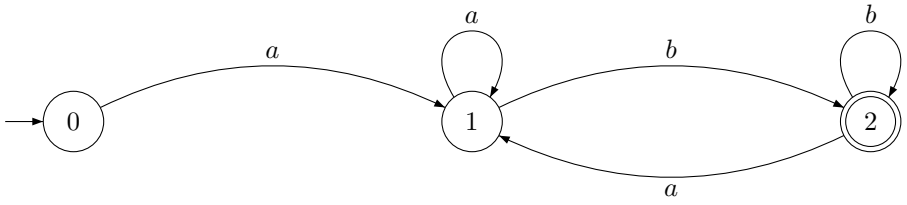

Figure 4: Un automate déterministe M_1 reconnaissant L_1

Le système d'équations est

$$\begin{aligned} L_0 &= aL_1 \\ L_1 &= aL_1 + bL_2 \\ L_2 &= bL_2 + aL_1 + \varepsilon \end{aligned}$$

En posant $B = \{b\}$ et $A = aL_1 + \varepsilon$, on peut appliquer le lemme et cela donne $L_2 = b^*(aL_1 + \varepsilon)$.
On reporte cette valeur de L_2 dans l'équation de L_1. On obtient $L_1 = aL_1 + bL_2 = aL_1 + b(b^*(aL_1 + \varepsilon)) = aL_1 + b^+aL_1 + b^+ = b^*aL_1 + b^+$.
En posant $B = b^*a$ et $A = b^+$, on peut appliquer le lemme et cela donne $L_1 = (b^*a)^*b^+$.
On reporte cette valeur de L_1 dans l'équation de L_0. On obtient $L_0 = aL_1 = a(b^*a)^*b^+ = a(b^*a)^*b^*b$.
Quand on sait que $(b^*a)^*b^* = (a + b)^*$, on retrouve $L_0 = aL_1 = a(a + b)^*b$.

♢

Comme, à tout automate, on peut associer une expression rationnelle, on a montré que $Rec(X^*)$ est inclus dans $Rat(X^*)$. Ceci clôt la démonstration de l'équivalence des trois familles.

4.4.2 De l'expression rationnelle à l'automate : l'automate de Glushkov

Le but est de construire un automate correspondant à une expression rationnelle.

Soit E une expression rationnelle décrivant un langage L.

On commence par indicer les lettres qui interviennent dans l'expression. On note E' l'expression indicée, qui décrit un langage L'. On note $x^{\#}$ la lettre à laquelle on retire l'indice ; par exemple, $a_3^{\#} = a$.

Exemple: Soit $E = a(a + b)^*b$. L'expression indicée est $E' = a_1(a_2 + b_3)^*b_4$. ♢

On calcule ensuite trois fonctions
$first(L') = \{b \mid \exists w \in X^* \text{ tel que } bw \in L'\}$
$last(L') = \{b \mid \exists w \in X^* \text{ tel que } wb \in L'\}$
Pour tout $a \in X$, $follow(L', a) = \{b \mid \exists v,\ w \in X^* \text{ tel que } vabw \in L'\}$

Il existe une construction inductive de ces fonctions, décrivant ce que valent ces fonctions sur une lettre et ce que valent ces fonctions pour une union, une concaténation ou une étoile de langages (voir Anne Brüggemann-Klein, *Regular Expressions into Finite Automata*, Theoretical Computer Science[1]).

L'automate de Glushkov est défini par $M = (X, Q \cup \{\bar{q}_0\}, \{\bar{q}_0\}, Q_F, \delta)$ avec

[1]Cet article n'est pas l'article qui a défini les automates de Glushkov, mais un article dans lequel on trouve les informations.

- $Q = alphabet(E')$, c'est à dire les états sont labellés par les lettres de l'expression indicée,
- $\delta(\bar{q}_0, a) = \{x \mid x \in first(E'), x^{\#} = a\}$,
- $\delta(x, a) = \{y \mid y \in follow(E', x), y^{\#} = a\}$ pour tout $x \in Q$,
- $Q_F = last(L') \cup \{\bar{q}_0\}$ si $\varepsilon \in L$
  $Q_F = last(L')$ sinon.

On peut remarquer que toute les transitions qui atteignent un état $x \in X'$ ont un label $x^{\#}$.

Exemple: Soient $E = a(a+b)^*b$ et $E' = a_1(a_2 + b_3)^*b_4$.

On a
$first(L') = \{a_1\}$
$last(L') = \{b_4\}$
$follow(L', a_1) = \{a_2, b_3, b_4\}$
$follow(L', a_2) = \{a_2, b_3, b_4\}$
$follow(L', b_3) = \{a_2, b_3, b_4\}$
$follow(L', b_4) = \emptyset$

On peut donc définir

- $Q = \{a_1, a_2, b_3, b_4$,
- $\delta(\bar{q}_0, a) = \{a_1\}$,
- $\delta(a_1, a) = \{a_2\}$ et $\delta(a_1, b) = \{b_3, b_4\}$,
- $\delta(a_2, a) = \{a_2\}$ et $\delta(a_2, b) = \{b_3, b_4\}$,
- $\delta(b_3, a) = \{a_2\}$ et $\delta(b_3, b) = \{b_3, b_4\}$,
- $Q_F = \{b_4\}$.

Ceci donne l'automate suivant :

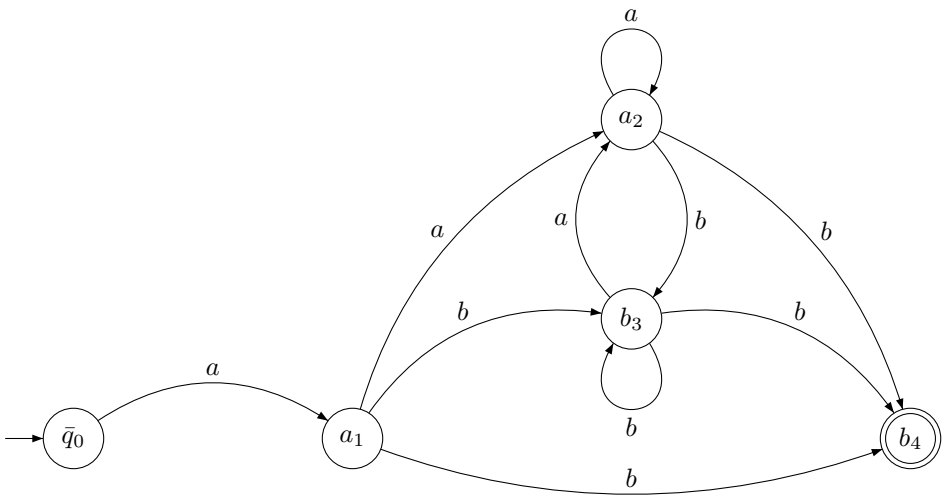

Figure 5: Un automate déterministe M_1 reconnaissant L_1

♢

Remarque : cette construction a un rapport avec XML. Dans une DTD, les expressions décrivant les éléments doivent être 1 non-ambigües. Dans ce cas, l'automate de Glushkov associé à l'expression est un automate déterministe.

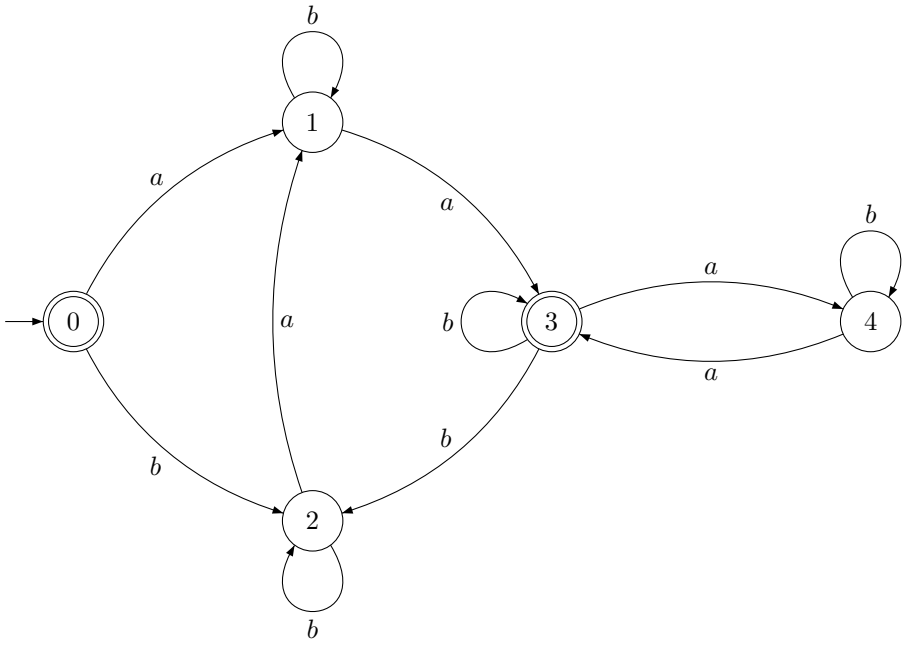

Figure 6: Un automate non déterministe

5 Déterminisation

Considérons l'automate suivant.

Il y a un seul état initial : l'état 0.

Il y a deux états finals : les états 0 et 3.

Le non déterminisme apparaît à l'état 3 qui atteint deux états pour la lettre b.

Pour déterminiser, il est pratique d'établir la table des transitions. Cette table se lit de la façon suivante : de l'état 0, on atteint l'état 1 par la lettre a ; de l'état 1, on atteint l'état 3 par la lettre a,....

	a	b
0	1	2
1	3	1
2	1	2
3	4	2,3
4	3	4

On déterminise par la subset-construction, c'est à dire la construction d'un automate déterministe sur des ensembles d'états.

On commence avec un état initial qui est l'état constitué de tous les états initiaux. Dans notre cas, l'état initial est $\{0\}$. Donc rien ne change au début, pour les états $\{0\}$, $\{1\}$ et $\{2\}$.

L'état 3 va, par la lettre b, vers les deux états 2 et 3; nous allons créer un **seul** état $\{2,3\}$, et la lettre b ira vers cet état. Puis on repartira de cet état $\{2,3\}$.

De l'état $\{2,3\}$, on atteint par la lettre a, tous les états qu'on atteint à partir de l'état 2 et de l'état 3. De l'état 2, on atteint l'état 1 et de l'état 3, on atteint l'état 4. Donc du nouvel état $\{2,3\}$, on atteint un état $\{1,4\}$

Et on continue... tant que de nouveaux ensembles d'états sont mis en évidence.

Voici la table de transition du déterministe.

	a	b
{0}	{1}	{2}
{1}	{3}	{1}
{2}	{1}	{2}
{3}	{4}	{2,3}
{4}	{3}	{4}
{2,3}	{1,4}	{2,3}
{1,4}	{3}	{1,4}

Les états finals sont les ensembles d'états qui contiennent au moins un état final. Donc, les états finals sont $\{0\}$, $\{3\}$ et $\{2,3\}$.

Et on obtient l'automate suivant qui est déterministe.

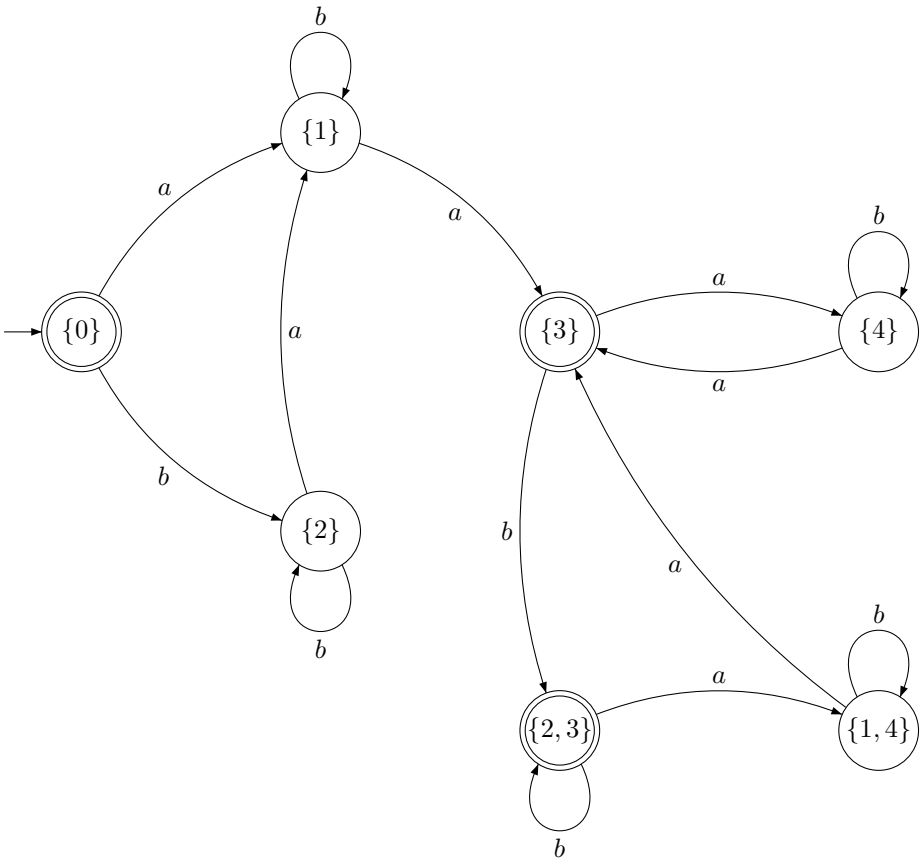

Figure 7: l'automate déterministe

Vous avez d'autres exemples de déterminisation par la subset-construction dans la section 7. En particulier, des déterminisations qui commencent avec un ensemble d'états initiaux non réduit à un singleton.

6 Minimalisation par la méthode de Moore

6.1 Construction

Pour simplifier l'explication, on renomme les états.

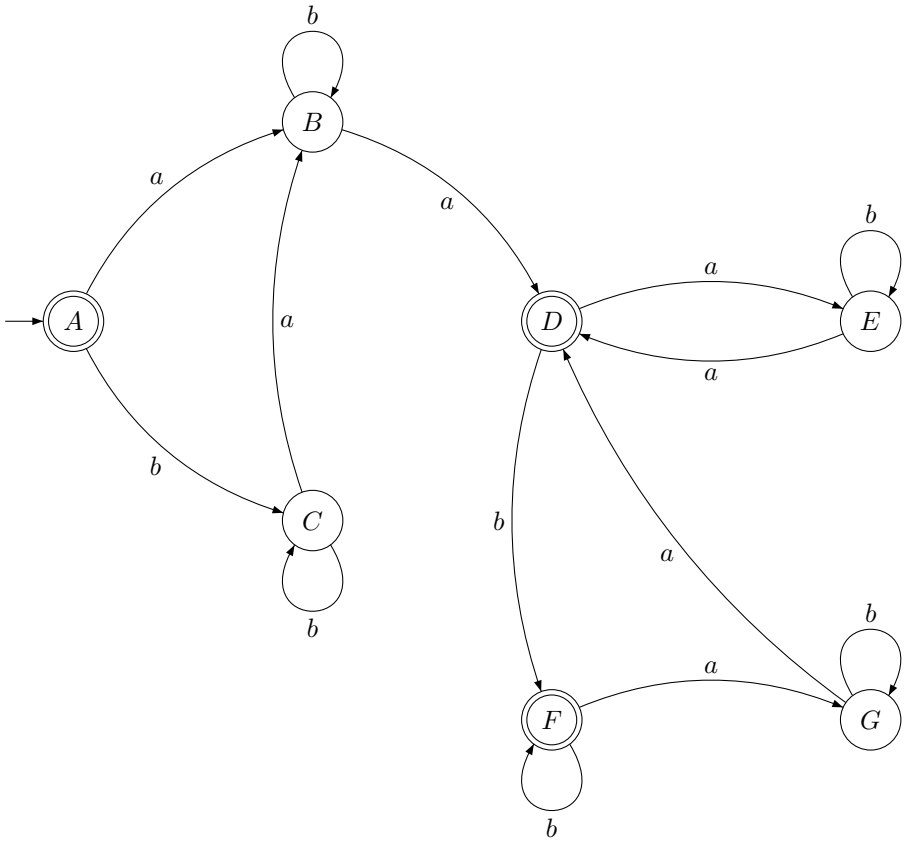

Figure 8: l'automate déterministe

La méthode de minimalisation s'appuie sur la séparation des états qui n'ont pas le même comportement. Cette méthode est très rapide. **Elle s'applique sur un automate déterministe.**

On commence par construire une partition en deux sous-ensembles : un ensemble qui contient les états finals et un ensemble qui contient les états non finals.

$$\{A, D, F\} \qquad \{B, C, E, G\}$$

On cherche ensuite une transition qui met en évidence une différence de comportement. C'est à dire une transition pour laquelle, on peut séparer les états contenus dans un sous-ensemble. Dans le sous-ensemble, certains états atteindront, par une certaine lettre, un sous-ensemble et les autres un autre sous-ensemble.

On peut passer en revue toutes les transitions. Mais dès qu'une lettre met en évidence une différence de comportement, on peut l'utiliser pour définir une nouvelle partition.

Prenons la lettre a partant de l'ensemble A, D, F. Pour l'état A, la lettre a mène dans un sous-ensemble (celui des états non finals). Est-ce que D et F ont le même comportement ? Oui, ils atteignent E et G qui sont non finals. Donc, cette lettre a ne nous permet pas de différencier les états dans l'ensemble $\{A, D, F\}$. Cette lettre a ne nous permet pas de changer la partition.

Par contre, la lettre b met en évidence des comportements différents. L'état A atteint l'ensemble des états non finals. Mais les états D et F atteignent l'ensemble des états finals. Donc, on obtient une nouvelle partition plus fine

$$\{A\} \qquad \{D, F\} \qquad \{B, C, E, G\}$$

On continue... la lettre a permet de séparer l'état C des états B, E et G. En effet, l'état C atteint l'état B, c'est à dire l'ensemble $\{B, C, E, G\}$ mais les états $\{B, E, G\}$ atteignent l'ensemble $\{D, F\}$.

$$\{A\} \qquad \{D,F\} \qquad \{C\} \qquad \{B,E,G\}$$

On continue... non, c'est déjà terminé ! Nous avons la partition la plus fine. Aucun sous-ensemble ne peut plus être scindé. Vous pouvez examiner chaque sous-ensemble d'états, les lettres a et b ne permettent plus de rien séparer.

En conclusion, certains sous-ensembles sont constitués d'états finals. Ils sont évidemment finals dans l'automate déterministe minimal.

Comme on commence avec un automate déterministe, il y a un seul état initial. Le sous-ensemble contenant l'état initial est initial dans l'automate déterministe minimal.

Donc, le sous-ensemble $\{A\}$ est initial ; les sous-ensembles $\{A\}$ et $\{D,F\}$ sont finals.

Pour les transitions, il n'y a pas d'ambiguïté puisqu'on ne peut plus scinder les sous-ensembles d'états.

Et on obtient

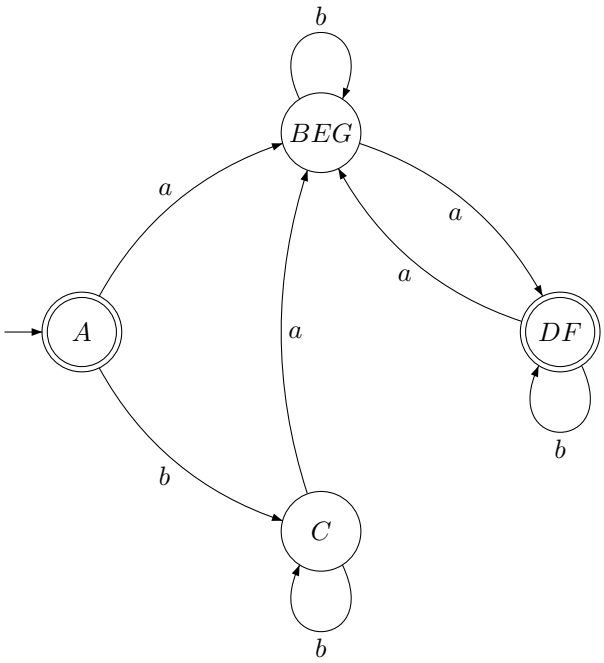

Figure 9: l'automate déterministe minimal

6.2 Autre exemple

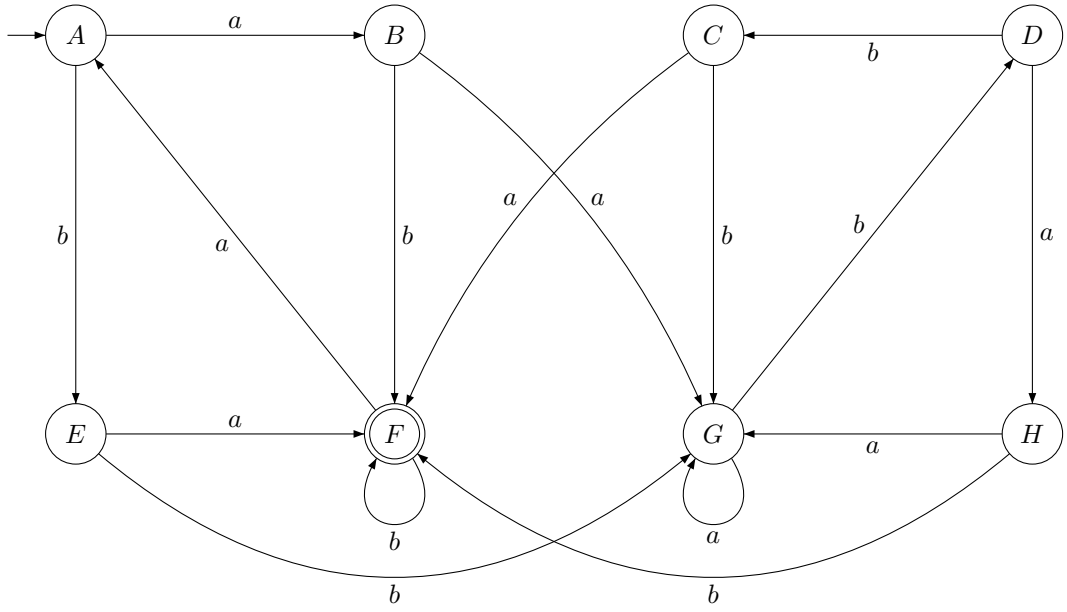

Figure 10: un automate déterministe

Partition initiale

$$\{F\} \qquad \{A, B, C, D, E, G, H\}$$

Par la lettre a, on peut séparer $\{C, E\}$ de $\{A, B, D, G, H\}$

$$\{F\} \qquad \{C, E\} \qquad \{A, B, D, G, H\}$$

Par la lettre b, on peut séparer $\{B, H\}$ de $\{A, D, G\}$

$$\{F\} \qquad \{C, E\} \qquad \{B, H\} \qquad \{A, D, G\}$$

Par la lettre b, on peut séparer $\{G\}$ de $\{A, D\}$

$$\{F\} \qquad \{C, E\} \qquad \{B, H\} \qquad \{A, D\} \qquad \{G\}$$

On ne peut plus séparer de sous-ensemble.

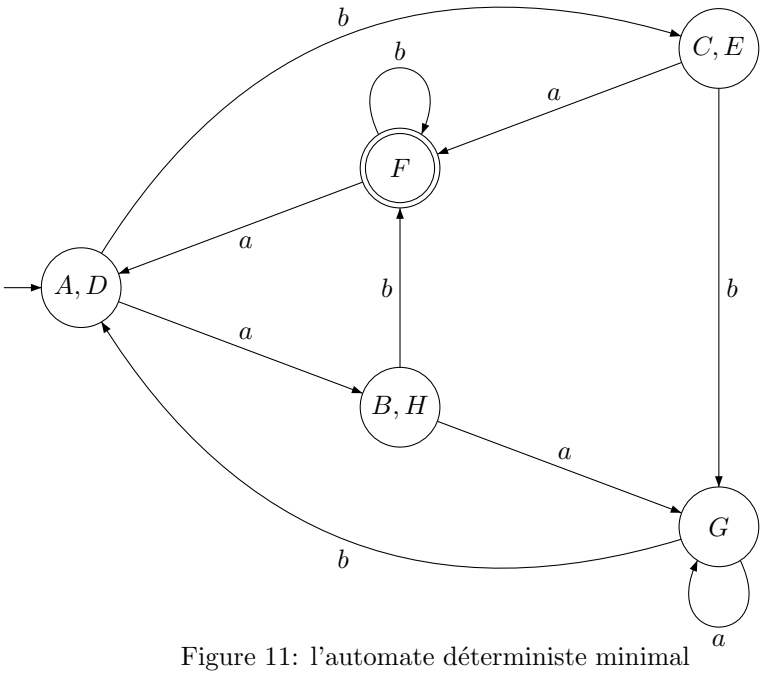

Figure 11: l'automate déterministe minimal

6.3 Exercice

Construire l'automate déterministe minimal équivalent à l'automate ci-dessous.

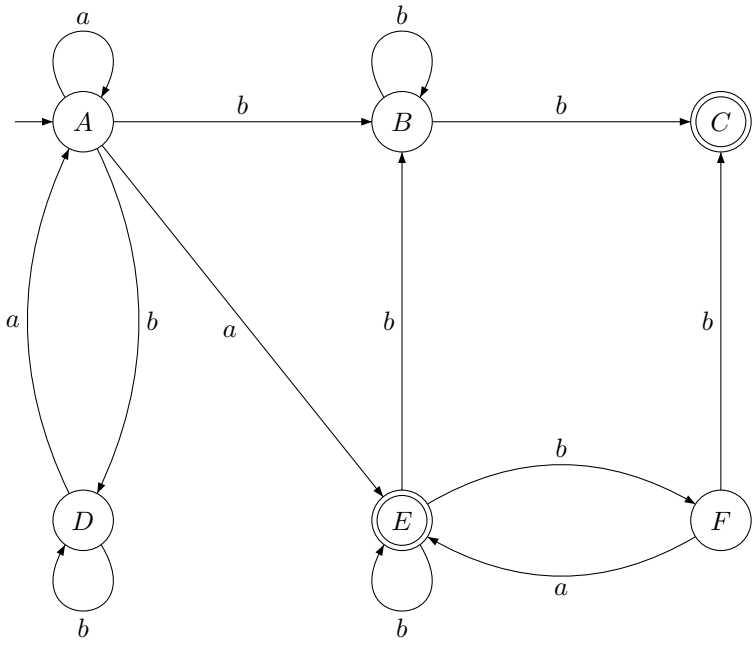

Figure 12: un automate non déterministe

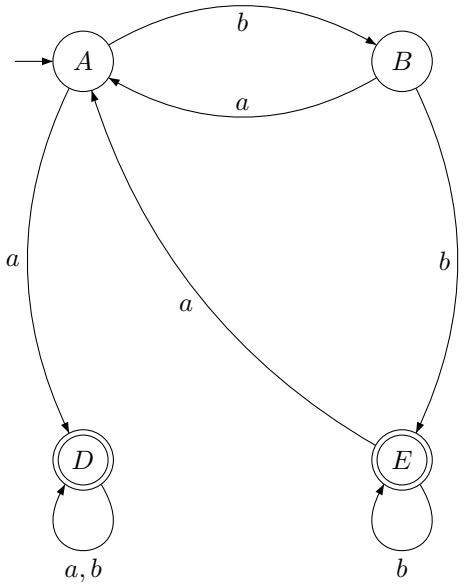

Figure 13: l'automate minimal

7 Minimalisation par la méthode de Brzozowski

7.1 Construction

Soit A un automate quelconque.

Notons A_{min} l'automate déterministe minimal, qu'on cherche à calculer.

Notons $D(A)$ l'automate déterministe équivalent à l'automate A qu'on obtient par la subset-construction (voir section 5).

Notons A^R l'automate miroir de A, c'est à dire l'automate obtenu en

- transformant les états initiaux en états finals
- transformant les états finals en états initiaux
- renversant toutes les transitions

Il est possible de prouver que $A_{min} = D((D(A^R))^R)$.

Donc pour calculer l'automate déterministe minimal A_{min} à partir d'un automate quelconque A, on applique la méthode suivante :

- on calcule le miroir de A, soit $A_1 = A^R$
- on calcule le déterministe de A_1, soit $A_2 = D(A_1)$
- on calcule le miroir de A_2, soit $A_3 = {A_2}^R$
- on calcule le déterministe de A_3, soit $A_4 = D(A_3)$
- l'automate obtenu est l'automate déterministe minimal, soit $A_{min} = A_4$

7.2 Exemple

Prenons l'automate de la figure 6 comme automate de départ (figure 14).

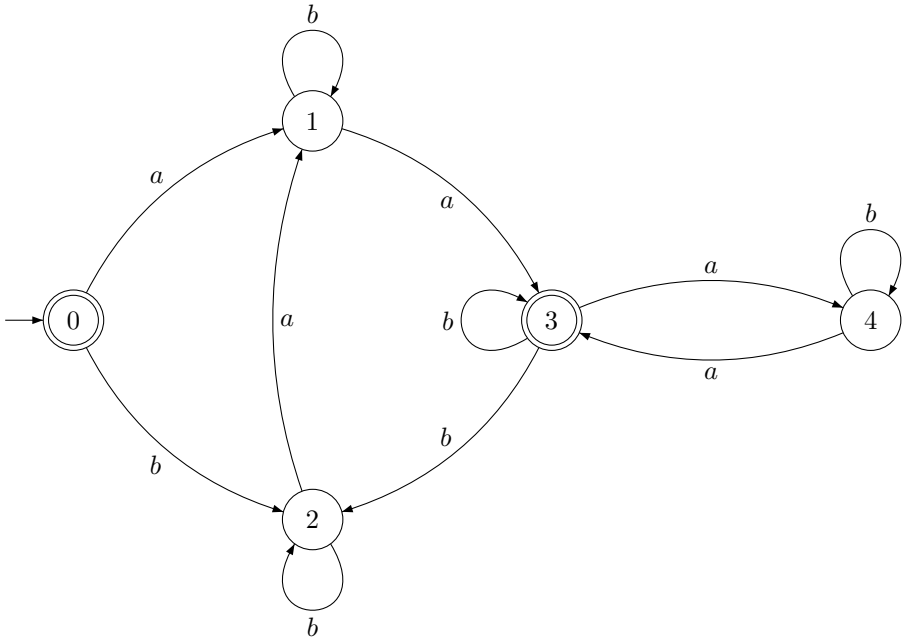

Figure 14: Un automate non déterministe A

On prend le miroir

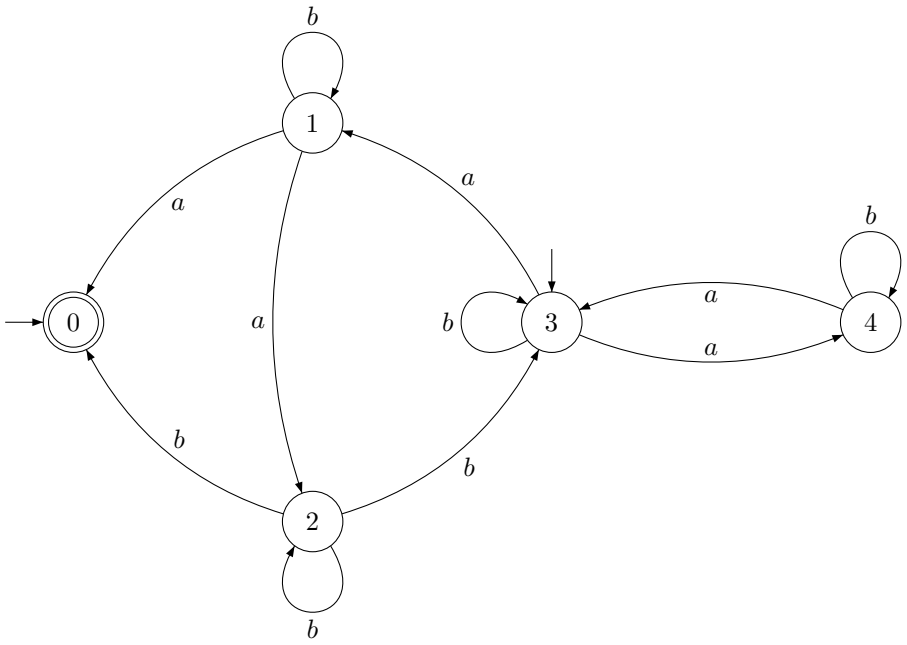

Figure 15: L'automate miroir $A_1 = A^R$

Table de transition

	a	b
0		
1	0,2	1
2		0,2,3
3	1,4	3
4	3	4

Les états 0 et 3 sont initiaux ; l'état 0 est final.

On construit l'automate déterministe par la subset-construction. Celui-ci commence avec l'ensemble des états initiaux $\{0, 3\}$.

Table de transition du déterministe

	a	b
{0,3}	{1,4}	{3}
{1,4}	{0,2,3}	{1,4}
{0,2,3}	{1,4}	{0,2,3}
{3}	{1,4}	{3}

Les états $\{0, 3\}$et $\{0, 2, 3\}$ sont finals car ils contiennent 0 qui était final.

dont on renomme les états

		a	b
{0,3}	A	B	D
{1,4}	B	C	B
{0,2,3}	C	B	C
{3}	D	B	D

c'est à dire les états A et C

on obtient l'automate de la figure 16

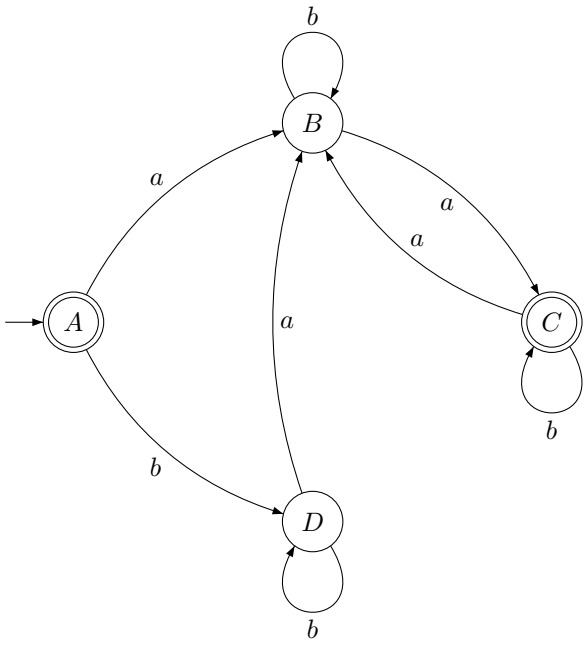

Figure 16: l'automate déterministe $A_2 = D(A_1)$

On prend le miroir

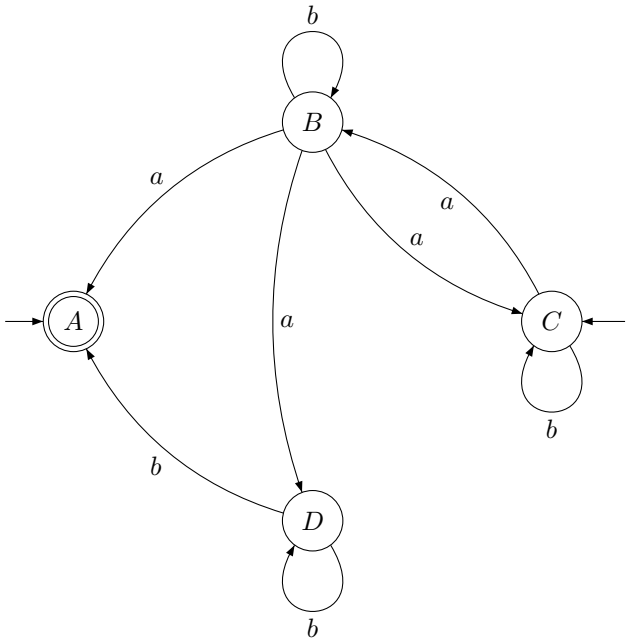

Figure 17: l'automate miroir $A_3 = {A_2}^R$

Table de transition de l'automate A_3

	a	b
A		
B	A,C,D	B
C	B	C
D		A,D

Les états A et C sont initiaux ; l'état A est final.

On construit l'automate déterministe par la subset-construction. Celui-ci commence avec l'ensemble des états initiaux $\{A, C\}$.

Table de transition du déterministe

	a	b
$\{A, C\}$	$\{B\}$	$\{C\}$
$\{B\}$	$\{A, C, D\}$	$\{B\}$
$\{C\}$	$\{B\}$	$\{C\}$
$\{A, C, D\}$	$\{B\}$	$\{A, C, D\}$

dont on renomme les états

		a	b
$\{A, C\}$	1	2	3
$\{B\}$	2	4	2
$\{C\}$	3	2	3
$\{A, C, D\}$	4	2	4

Les états $\{A, C\}$ et $\{A, C, D\}$ sont finals car ils contiennent A qui était final.

c'est à dire les états 1 et 4

on obtient enfin l'automate déterministe minimal de la figure 18

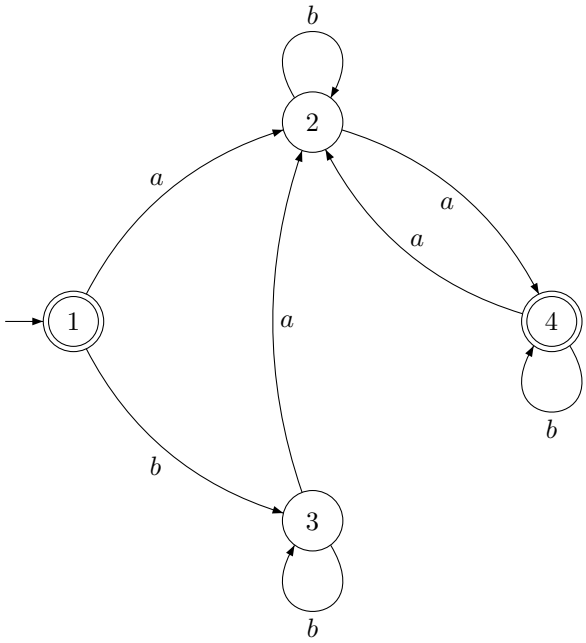

Figure 18: l'automate déterministe $A_{min} = A_4 = D(A_3)$

Vous pouvez comparer cet automate avec celui de la figure 9