

Calculabilité & Complexité

Notes de cours

Xavier Dubuc

Xavier.DUBUC@student.umons.ac.be



Table des matières

1	Préambule	3
2	Introduction	3
3	Alphabets, words, languages and algorithmic problems	4
3.1	Introduction	4
3.2	Notations et Définitions	4
3.3	Représentations	4
3.4	Opérations sur les mots et les langages	5
3.4.1	Définitions & Notations	5
3.4.2	Exemples	5
3.5	Problèmes algorithmiques	6
3.5.1	Problème de décision	6
3.5.2	Fonctions calculables	6
4	Finite automata	7
4.1	Vision graphique d'un automate	8
4.2	Fonctionnement d'un automate	9
4.2.1	Vocabulaire	9
4.2.2	Notations	9
4.2.3	Prouver qu'un langage est régulier	9
4.2.4	Prouver qu'un langage n'est pas régulier	11
4.3	Automates non-déterministes	12
4.3.1	Vocabulaire et définitions revisitées	12
4.3.2	Langage calculé par un automate non-déterministe	13
4.3.3	Questions	14

1 Préambule

Le cours se base sur 2 livres de références dont voici les références :

Theoretical Computer Science

Juraj Hromkovic

Springer

Introduction to the theory of Computation

Michael Sipser

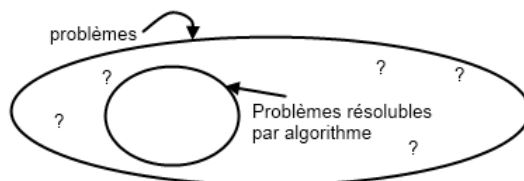
PWS Publishing Company 1997

Sur *moodle* seront placés régulièrement des devoirs consistant en la lecture d'un chapitre du livre ou à la réalisation d'un exercice demandé qui sera éventuellement corrigé en classe. L'*examen écrit* se déroule à **cahier ouvert**.

2 Introduction

Dans le cadre de nos études, nous avons été amenés à faire beaucoup de programmation, il nous a été demandé que nos programmes s'arrêtent et qu'ils soient corrects et qu'ils utilisent des algorithmes efficaces en *temps* et en *espace* (*notion de complexité dans le pire des cas*). Dans ce cours, nous allons nous poser des questions fondamentales sur ce que l'on peut résoudre via l'algorithmique.

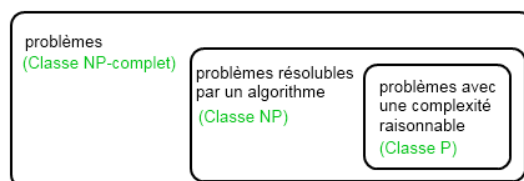
1. *Calculabilité* : "Étant donné un problème bien formulé, peut-on le résoudre par un algorithme?"



On va devoir faire des preuves pour lesquelles nous aurons besoin d'un modèle mathématique pour les algorithmes, nous utiliserons les *machines de Turing* (On a vu précédemment les automates qui sont des machines de Turing simplifiées).

2. *Complexité* : "Étant donné un problème que l'on peut résoudre par un algorithme, quelle est sa complexité?"

La complexité exprimée ici sera calculée sur les *machines de Turing* et non sur les algorithmes, on va ainsi définir des *classes de complexité*.



3 Alphabets, words, languages and algorithmic problems

3.1 Introduction

On va avoir besoin de la notion de *mots*, *textes* sur un *alphabet*.

→ Exemple : un programme en JAVA est un texte sur l'alphabet du clavier :

$$\text{entrées} \rightarrow \boxed{\text{Programme}} \rightarrow \text{sorties}$$

Les entrées et les sorties sont également des textes sur un alphabet donné (par exemple, une entrée $n \in \mathbb{N}$ est un mot sur l'alphabet $\{0, \dots, 9\}$).

→ Exemple 2 : le compilateur prend un programme en entrée et sa sortie, le programme objet est un texte sur l'alphabet $\{0, 1\}$

3.2 Notations et Définitions

- Un *alphabet* Σ est un ensemble fini de *symboles*. Par exemple :
 - $\Sigma_{bool} = \{0, 1\}$
 - $\Sigma_{lat} = \{a, b, \dots, z\}$
 - $\Sigma_{keyboard}$ = alphabet du clavier (il y a le symbole d'espacement)
 - $\Sigma_m = \{0, 1, \dots, m-1\}$, l'alphabet des digits pour écrire les nombres en base m .
 - $\Sigma_{logic} = \{x, 0, 1, (,), \wedge, \vee, \neg\}$
 - ...
- Un *mot* sur l'*alphabet* Σ est une suite finie de *symboles* de Σ . Le *mot vide* existe, on le note λ , il correspond à la suite vide de symboles de Σ .
- La *longueur* d'un *mot* w , notée $|w|$, est la longueur de la suite de symboles formant w . Par exemple, $|\lambda| = 0$, $|\text{"espace"}| = 1$.
- Σ^* est l'ensemble de tous les mots sur l'alphabet Σ .
- Σ^+ est l'ensemble de tous les mots non-vides sur l'alphabet $\Sigma \Rightarrow \Sigma^+ = \Sigma^* \setminus \{\lambda\}$.
Par exemple, $\Sigma_{bool}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$, $\Sigma_{lat}^* = \{zzz, agy, kikoo, lol, mdr, xd, cmb, wtf, \dots\}$
- Un *langage* L sur un *alphabet* Σ est une partie (un sous-ensemble) de Σ^* , $\Rightarrow L \subseteq \Sigma^*$.
A l'extrême, $L = \Sigma^*$ et $L = \emptyset$ sont possibles.

3.3 Représentations

Il s'agit ici de parler de la représentation (par des mots sur un alphabet bien choisi) non-ambigüe d'objets manipulés par les programmes/algorithmes. Par exemple, les nombres naturels peuvent être représentés en base 10 ou en base 2, ainsi pour la base 2, $n \in \mathbb{N}$ est représenté par $Bin(n)$ qui est un mot sur Σ_{bool} . La représentation se doit d'être *non-ambigüe* afin que 2 objets ne soient pas représentés par le même mot ; c'est-à-dire qu'à partir de la représentation d'un objet on doit être capable de retrouver l'objet sans ambiguïté.

Exemple :

1. Un ensemble de nombres naturels n_1, \dots, n_k représenté sur l'alphabet $\{0, 1, \#\}$ par le mot $Bin(n_1)\#\dots\#Bin(n_k)$, cette représentation est non-ambigüe grâce au marqueur $\#$.

2. Un graphe orienté G dont la matrice d'adjacence est :
$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$
 représenté sur l'alphabet $\{0, 1, \#\}$

par le mot $100\#0011001101010000$ (100 étant le nombre de sommets du graphe, c'est-à-dire 4) ou par le mot $0011\#0011\#0101\#0000$.

3. Un graphe non-orienté pondéré dont la matrice d'adjacence est : $\begin{pmatrix} 0 & 1 & 10 & 6 \\ 1 & 0 & +\infty & 5 \\ 10 & +\infty & 0 & 3 \\ 6 & 5 & 3 & 0 \end{pmatrix}$ représenté sur

l'alphabet $\{0, 1, \#\}$ par le mot $0\#1\#1010\#110\#\#1\#0\ldots$; on place la représentation binaire de chaque nombre que l'on sépare par $\#$, puis entre chaque ligne du graphe on place 2 marqueurs : $\#\#$.

4. Une formule booléenne φ contenant des *variables booléennes* $X_1, X_2, \dots, X_i, \dots$, des connecteurs logiques \vee, \wedge, \neg et des parenthèses, par exemple, $\varphi \equiv (X_1 \wedge X_7) \vee \neg X_3$ est représentée sur l'alphabet logique $(\Sigma_{logic} = \{x, 0, 1, (,), \wedge, \vee, \neg\})$ par le mot $(X1 \vee X111) \wedge \neg X11$.

Remarque : TOUT peut être codé en binaire (l'ordinateur le fait !), par exemple si on a une représentation sur $\{0, 1, \#\}$, disons le mot $01\#11\#0$, on peut le transformer en mot sur $\{0, 1\}$ en appliquant les transformations suivantes (*homéomorphisme*) :

- $0 \rightarrow 00$
- $1 \rightarrow 10$
- $\# \rightarrow 11$

dans le cas de l'exemple $01\#11\#0 \rightarrow f(0)f(1)f(\#)f(1)f(1)f(\#)f(0) = 00101110101100$

3.4 Opérations sur les mots et les langages

3.4.1 Définitions & Notations

- Étant donnés 2 mots x et y sur l'alphabet Σ ,
 - la *concaténation* $x.y$ (noté aussi xy) est le mot obtenu en concaténant y à la suite de x . (Remarque : $|x.y| = |x| + |y|$, $x\lambda = \lambda x = x$)
 - x est *préfixe* de y si $\exists z \in \Sigma^*$ tel que $x.z = y$,
 - x est *suffixe* de y si $\exists z \in \Sigma^*$ tel que $z.x = y$,
 - x est *facteur* ou *sous-mot* de y si $\exists z_1, z_2 \in \Sigma^*$ tels que $z_1.x.z_2 = y$.
(*suffixe et préfixe sont des cas particuliers de facteur*)
- Soit un langage L ,
 - la *cardinalité* de L (nombre de mots qu'il contient) est notée $|L|$,
 - $\mathcal{P}(L)$ = ensemble des *parties/sous-ensembles* de L ,
 - l'*exposant* i de L , note L^i est défini comme

$$L^0 = \{\lambda\}, L^1 = L, L^2 = L.L, L^i = L.L \dots L \text{ (i fois, } i \geq 2)$$

- l'*étoile* de L , notée L^* est définie comme

$$\bigcup_{i \geq 0} L^i \text{ ou } \{X_1 X_2 \dots X_n | X_i \in L, 1 \leq i \leq n, n \geq 0\}$$

- Soit x un mot sur Σ contenant le symbole a ,
 - $|x|$ est la *longueur* de x ,
 - $|x|_a$ est le *nombre d'occurrences du symbole a* dans x .
- Soit $L_1, L_2 \subseteq \Sigma^*$ deux langages sur l'alphabet Σ , la *concaténation* $L_1.L_2$ est définie comme $\{x_1.x_2 | x_1 \in L_1, x_2 \in L_2\}$,

3.4.2 Exemples

- $L_1 = \{0, 11\}, L_2 = \{1, 10\} \rightarrow L_1.L_2 = \{01, 010, 111, 1110\}$
- $L = \{00, 01, 10, 11\}, L^* = L^0 \cup L^1 \cup \dots = \{\lambda\} \cup L \cup \{\text{mots longueur 4}\} \cup \dots$
 $= \{\text{mots sur } \Sigma_{bool} \text{ de longueur paire}\}.$
- Σ^* est l'*étoile* de Σ : $\{X_1 X_2 \dots X_n | X_i \in \Sigma \text{ (} X_i \text{ symbole), } 1 \leq i \leq n, n \geq 0\}$

3.5 Problèmes algorithmiques

Soit un algorithme \mathbf{A} : entrée $x \in \Sigma_1^* \rightarrow \boxed{\mathbf{A}} \rightarrow$ sortie $y \in \Sigma_2^*$, sera modélisé plus tard par la notion de *machine de Turing*. Le but de l'algorithme est donc de prendre un mot sur un certain alphabet en entrée et de fournir en sortie un autre mot sur un certain alphabet. Un exemple d'algorithme \mathbf{A} serait : x est la représentation non-ambigüe d'un graphe orienté sur $\{0, 1, \#\}$ et $y \in \{0, 1\}$ tel que $y = 0$ si le graphe est acyclique et $y = 1$ sinon.

3.5.1 Problème de décision

Existe-t-il un algorithme \mathbf{A} qui peut résoudre ce *problème de décision* (Σ, L) (tel que $L \subseteq \Sigma^*$), c'est-à-dire un algorithme qui prend un mot $x \in \Sigma^*$ en entrée et qui fournit en sortie 1 si $x \in L$ et 0 si $x \notin L$? Dit autrement, existe-t-il un algorithme \mathbf{A} qui peut décider, pour x un mot quelconque sur Σ , si oui ou non $x \in L$? Si un tel algorithme existe, on dit que le *problème de décision* (Σ, L) est *décidable*, sinon il est dit *indécidable*. Si un tel algorithme existe, on dit que le *langage* L est *récuratif*.

Exemples :

1. Problème de décision $(\Sigma = \{a, b\}, L = \{a^n b^n | n \geq 0\})$, l'algorithme \mathbf{A} (simple) existe, il s'agit donc d'un problème *décidable* et L est un langage *récuratif*.
2. Problème de décision $(\Sigma_{10}, L = \{x \in \Sigma_{10}^* | x \text{ est l'écriture en base 10 d'un nombre premier}\})$, l'algorithme \mathbf{A} existe, il s'agit donc d'un problème *décidable* et L est un langage *récuratif*.
3. Problème de décision $(\Sigma_{\text{keyboard}}, L = \{x \in \Sigma_{\text{keyboard}}^* | x \text{ est un programme JAVA syntaxiquement correct}\})$, l'algorithme \mathbf{A} existe (il s'agit de l'analyse syntaxique du compilateur **JAVA**, il s'agit donc d'un problème *décidable* et L est un langage *récuratif*.
4. Problème de décision $(\Sigma_{\text{logic}}, L = \{x \in \Sigma_{\text{logic}}^* | x \text{ représente une formule booléenne satisfaisable}\})$, c'est-à-dire qu'il existe une assignation des variables booléennes de x qui rend x vraie), l'algorithme naïf serait de tout tester mais sa complexité est alors exponentielle! Il s'agit en fait d'un problème *NP-complet*.
5. Problème de décision $(\Sigma = \{0, 1, \#\}, L)$ avec
 - $L = \{w \in \Sigma^* | w \text{ représente un graphe } G \text{ non-orienté possédant un cycle Hamiltonien}\}$ (c'est-à-dire un graphe admettant un chemin passant 1 seule fois par chaque sommet)
 - $L' = \{w \in \Sigma^* | w \text{ représente un graphe } G \text{ non-orienté possédant un cycle Eulérien}\}$ (c'est-à-dire un graphe admettant un chemin passant 1 seule fois par chaque arête). (Σ, L) et (Σ, L') sont *décidables* mais (Σ, L) est *NP-complet* (il existe donc un algorithme mais de complexité *exponentielle*) tandis que (Σ, L') est *polynomial*, pour le prouver (et donc construire un algorithme), il suffit de s'appuyer sur le théorème suivant :

G possède un *cycle eulérien* SSI G est connexe et que tout sommet a un nombre pair d'arêtes adjacentes.

6. Étant donné deux programmes **JAVA**, peut-on décider s'ils sont équivalents? (c'est-à-dire est-ce qu'ils produisent les mêmes sorties sur les mêmes entrées)
7. Étant donné un programme, peut-on décider par un algorithme s'il s'arrête toujours?

3.5.2 Fonctions calculables

Définition : Soit $f : \Sigma_1^* \rightarrow \Sigma_2^*$ une fonction, on dit qu'elle est *calculable* s'il existe un algorithme \mathbf{A} tel que :

$$w \in \Sigma_1^* \rightarrow \boxed{\mathbf{A}} \rightarrow f(w) \in \Sigma_2^*$$

Cela généralise la notion de problème de décision : $f : \Sigma^* \rightarrow \{0, 1\}$ tel que :

- $f(w) = 0$ si $w \notin L$
- $f(w) = 1$ sinon

Par exemple, un tel algorithme \mathbf{A} existe :

$$\langle x, y \rangle \rightarrow \boxed{\mathbf{A}} \rightarrow \langle x + y \rangle$$

($\langle \rangle$ signifie qu'il s'agit d'une représentation non ambigüe sur un alphabet donné)

4 Finite automata

Le but de ce chapitre est d'introduire les *machines de Turing*, il s'agit du modèle mathématique utilisé pour définir la notion d'algorithme. Pour aider à bien les comprendre, nous allons commencer par le modèle plus simple d'*automate fini* et voir également des algorithmes "*simples*" pour montrer que certains problèmes de décision sont *décidables*.

Définition d'un automate fini sous la forme d'un programme :

- alphabet $\Sigma = \{a_1, a_2, \dots, a_k\}$ fixé,
 - une seule instruction autorisée :
 select input = a1 goto i1
 input = a2 goto i2
 ...
 input = ak goto ik
 - un programme est une suite finie d'instruction de ce type, chaque instruction étant numérotée (*en commençant par 0*)
 - si le programme a des instructions numérotées de 0 à m alors on dispose d'un ensemble $F \subseteq \{0, 1, \dots, m\}$
- Soit $w \in \Sigma^*$, l'automate accepte w si :
- w va être traité symbole par symbole de la gauche vers la droite par les instructions du programme, en commençant par l'instruction 0,
 - quand w est entièrement traité, il faut que le numéro de la dernière instruction utilisée $\in F$.

Exemple : (programme avec $\Sigma = \{0, 1\}$ et $F = \{0, 3\}$)

```
0  select input = 0    goto 2
   select input = 1    goto 1
1  select input = 0    goto 3
   select input = 1    goto 0
2  select input = 0    goto 0
   select input = 1    goto 3
3  select input = 0    goto 1
   select input = 1    goto 2
```

```
w = 1011
symboles :      1 0 1 1
instructions : 0 1 3 2 3
```

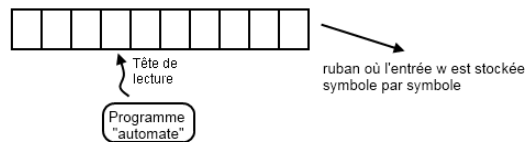
La dernière instruction choisie est 3 et $3 \in F$, w est donc accepté (*par exemple, $w = 101$ n'est pas accepté*).
Un automate peut donc être assimilé à un problème de décision :

$$w \in \Sigma^* \rightarrow \boxed{\text{Automate}} \rightarrow \text{accepté/refusé}$$

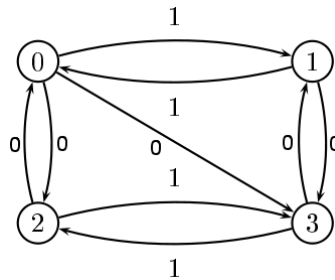
Les programmes de ce type sont de puissances très limitée :

- un seul type d'instruction,
- pas de variables autorisées,
- le mot w ne peut être lu que d'une seule façon (*symbole par symbole de gauche à droite*).

4.1 Vision graphique d'un automate



Autre définition d'automate : cette représentation consiste à considérer les instructions comme les états de l'automate.



Les *goto*'s d'un instruction correspondent aux transitions issues de l'état correspondant.

$F = \{0, 3\}$
 $w = 1011$
 symboles = 1 0 1 1
 etats = 0 1 3 2 3

$3 \in F \Rightarrow w$ accepté.

Définition d'un automate

Un *automate fini* M est de la forme $(Q, \Sigma, \delta, q_0, F)$ tel que :

- Q est un ensemble fini d'états, (*nombre fini d'instructions*)
- Σ est un alphabet,
- q_0 est l'état initial ($q_0 \in Q$), (*instruction de numéro 0*)
- $F \subseteq Q$ est l'ensemble des états accepteurs,
- $\delta : Q \times \Sigma \rightarrow Q$ est une fonction qui à chaque état $q \in Q$, à chaque symbole $a \in \Sigma$ associe un état $p \in Q$: $\delta(q, a) = p$ (*goto pour l'instruction correspondant à l'état q*).

Notations graphiques :

- état : \textcircled{p}
- état initial : $\textcircled{q_0}$
- état accepteur : \textcircled{r}
- transition : $\delta(q, a) = p$: $\textcircled{q} \xrightarrow{a} \textcircled{p}$

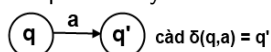
4.2 Fonctionnement d'un automate

4.2.1 Vocabulaire

- **configuration** (q, u) où $q \in Q$, $u \in \Sigma^*$, "photo" indiquant l'état courant q et le mot u qu'il reste à traiter,
- **configuration initiale** (q_0, w) où q_0 état initial et w est le mot d'entrée de l'automate,
- **configuration finale** (p, λ) où $p \in Q$ et λ mot vide,
- une **étape de calcul** consiste à passer d'une **configuration** à la **configuration** suivante :

$$(q, u) \vdash_M (q', u')$$

tq $u = au'$ avec $a \in \Sigma$ et $u' \in \Sigma^*$ (et a le premier symbole de u),



- un **calcul** est une suite d'étapes de calcul consécutives,
- le **calcul effectué par l'automate M sur l'entrée w** est un **calcul** commençant en la configuration (q_0, w) et se terminant en une configuration finale qui est donc de la forme (p, λ) . Le **mot d'entrée** $w \in \Sigma^*$ est **accepté** si ce calcul aboutit à (p, λ) avec $p \in F$, il est **refusé** sinon.

Exemple : (voir automate précédent) $w = 1011$ est-il accepté ?

$$(0, 1011) \vdash_M (1, 011) \vdash_M (3, 11) \vdash_M (2, 1) \vdash_M (3, \lambda)$$

Ok car $3 \in F$, w est donc **accepté**.

4.2.2 Notations

- \vdash_M^* pour décrire un calcul (0, 1 ou plusieurs étapes de calcul), sur l'exemple ci-dessus on a :

$$(0, 1011) \vdash_M^* (3, \lambda)$$

- $L \subseteq \Sigma^*$ est dit **régulier** s'il existe un automate fini M tel que $L(M) = L$
- $\hat{\delta}$ étend δ aux mots, $\hat{\delta} : Q \times \Sigma^* \rightarrow Q : (p, u) \rightarrow q$ (où $u = a_1 a_2 \dots a_n$, $n \geq 0$ et les $a_i \in \Sigma$ $1 \leq i \leq n$)
- $L(M) = \{w \in \Sigma^* \mid w \text{ est accepté par } M\}$ ou encore $\{w \in \Sigma^* \mid (q_0, w) \vdash_M^* (p, \lambda) \text{ tel que } p \in F\}$, est le **langage accepté par M** .

M résout donc le problème de décision suivant $(\Sigma, L(M))$:

$$w \in \Sigma^* \rightarrow \boxed{M} \rightarrow \text{accepté } (w \in L(M)) \text{ ou refusé } (w \notin L(M))$$

Pour l'exemple donné, $L(M) = \{w \in \{0, 1\}^* \mid |w| \text{ pair}\}$.

4.2.3 Prouver qu'un langage est régulier

Preuve : **idée sous-jacente importante :** associer à chaque état l'information :

$$Kl[p] = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) = p\}$$

c'est-à-dire l'ensemble des étiquettes des chemins allant de l'état initial q_0 à l'état p .

Dans notre exemple ci-dessus :

$$(*) \left\{ \begin{array}{lcl} Kl[q_0] & = & \{w \in \Sigma^* \mid |w|_0 \text{ et } |w|_1 \text{ sont pairs}\} \\ Kl[q_1] & = & \{w \in \Sigma^* \mid |w|_0 \text{ pair, } |w|_1 \text{ impair}\} \\ Kl[q_2] & = & \{w \in \Sigma^* \mid |w|_0 \text{ impair, } |w|_1 \text{ pair}\} \\ Kl[q_3] & = & \{w \in \Sigma^* \mid |w|_0 \text{ et } |w|_1 \text{ sont impairs}\} \end{array} \right.$$

De manière générale,

$$L(M) = \bigcup_{p \in F} Kl[p]$$

Dans notre exemple,

$$\begin{aligned} L(M) &= Kl[q_0] \cup Kl[q_3] \\ &= \{w \in \Sigma^* \mid |w|_0 \text{ et } |w|_1 \text{ sont pairs ou } |w|_0 \text{ et } |w|_1 \text{ sont impairs}\} \\ &= \{w \in \Sigma^* \mid |w|_0 + |w|_1 \text{ est pair}\} \end{aligned}$$

On va à présent prouver (*) par récurrence sur la longueur des mots $w \in \Sigma^*$:

— **Cas de base** : $|w| \leq 2$

1. $|w| = 0$ c'est-à-dire $w = \lambda$ et donc $|w|_0 = |w|_1 = 0$, c'est donc pair et $\hat{\delta}(q_0, w) = q_0$ de manière évidente.
2. $|w| = 1$ c'est-à-dire
 - $w = 0$ en quel cas, $|w|_0 = 1$ (*impair*) et $|w|_1 = 0$ (*pair*) et $\hat{\delta}(q_0, w) = q_2$ (cf dessin).
 - $w = 1$ en quel cas, $|w|_0 = 0$ (*pair*) et $|w|_1 = 1$ (*impair*) et $\hat{\delta}(q_0, w) = q_4$ (cf dessin).
3. $|w| = 2$ c'est-à-dire
 - $|w| = 00 \dots$
 - $|w| = 01 \dots$
 - $|w| = 10 \dots$
 - $|w| = 11 \dots$

— **Cas général** : soit $w \in \Sigma^*$ tel que $|w| = k \geq 3$,

Écrivons w sous la forme $w = za$ où $a \in \Sigma^*$, $|z| = |w| - 1$, par hypothèse de récurrence, (*) est vrai pour z , montrons (*) pour w .

— **1er cas**, supposons que $z \in Kl[q_2]$,

1. et que $a = 0$, par hypothèse de récurrence, $|z|_0$ impair et $|z|_1$ pair, donc :

$$\begin{cases} |w|_0 &= |z|_0 + 1 \rightarrow \text{pair} \\ |w|_1 &= |z|_1 \rightarrow \text{pair} \end{cases}$$

Dans l'automate on a $\hat{\delta}(q_0, z) = q_2$ et $\delta(q_2, 0) = q_0$.

2. et que $a = 1$, par hypothèse de récurrence, $|z|_0$ impair et $|z|_1$ pair, donc :

$$\begin{cases} |w|_0 &= |z|_0 \rightarrow \text{impair} \\ |w|_1 &= |z|_1 + 1 \rightarrow \text{impair} \end{cases}$$

Dans l'automate on a $\hat{\delta}(q_0, z) = q_2$ et $\delta(q_2, 1) = q_3$.

- **2ème cas**, supposons que $z \in Kl[q_1]$, ...
- **3ème cas**, supposons que $z \in Kl[q_3]$, ...
- **4ème cas**, supposons que $z \in Kl[q_4]$, ...

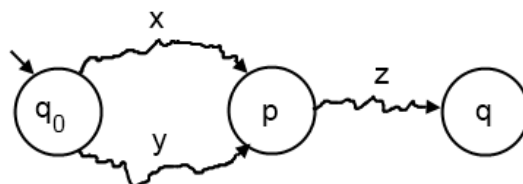
Remarque : pour atteindre q_3 (et les autres) il faut envisager les mots de longueurs ≤ 2 dans le cas de base.

4.2.4 Prouver qu'un langage n'est pas régulier

Il faut montrer qu'il n'existe aucun automate M tel que $L(M) = L$, ce qui est plus difficile. Pour nous aider à faire ces preuves, nous allons voir 2 outils,

1. Lemme 3.12

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate fini, soit $x, y \in \Sigma^*$, $x \neq y$ et tels que $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$,
alors $\forall z \in \Sigma^*$, $xz \in L(M) \Leftrightarrow yz \in L(M)$



Comme on peut le voir sur le dessin, si $q \in F$ alors les 2 mots seront acceptés, ils seront refusés sinon ; ceci est dû au fait que $\hat{\delta}$ est une fonction et donc, si on choisit un mot et un état de départ alors il n'y a qu'un seul état d'arrivée. $x, y \in Kl[p]$, M n'est pas capable de distinguer x et y une fois que l'état p est atteint.

Exemples de preuves utilisant cet outil : → cf feuilles d'exercices.

2. Lemme de la pompe (Lemme 3.14)

Soit $L \subseteq \Sigma^*$ un langage régulier, alors il existe une constante $n_0 \in \mathbb{N}$ telle que $\forall w \in \Sigma^*$ avec $|w| \geq n_0$, on a $w = yxz$ t.q. :

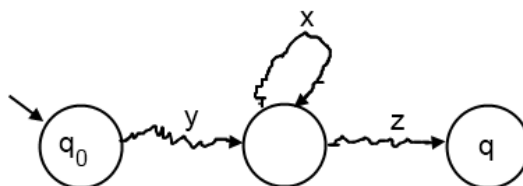
- (a) $|yx| \leq n_0$
- (b) $|x| \geq 1$
- (c) soit $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$,
soit $\{yx^kz \mid k \in \mathbb{N}\} \cap L = \emptyset$ ("ne contient aucun mot de L ")

Commentaires :

- la *pompe* désigne le fait d'itérer le mot x ,
- ou bien tous les mots de la forme yx^kz avec $k \geq 0$ sont **acceptés** ou bien ils sont tous **rejetés**.

Preuve : comme L est *régulier*, il existe un automate fini M tel que $L(M) = L$, prenons $n_0 = |Q|$, c'est-à-dire le nombre d'états que contient M .

Soit w tel que $|w| \geq n_0$, en partant de q_0 et en lisant w , on doit nécessairement passer 2 fois par le même état, cela apparaîtra dès qu'on aura lu n_0 symboles (*) de $w = yxz$.



On a bien :

- (a) $|yx| \leq n_0$ car (*),
- (b) $|x| \geq 1$ car cycle non-vide d'étiquette x ,
- (c) Si $q \in F$, alors $\forall k yx^kz$ est **accepté**, sinon $\forall k yx^kz$ est **refusé**.

Exemples de preuves utilisant cet outil : → cf feuilles d'exercices.

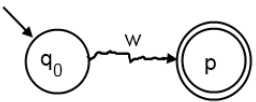
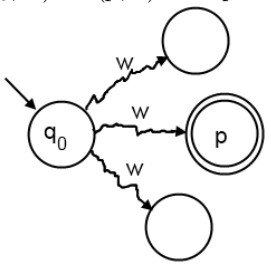
4.3 Automates non-déterministes

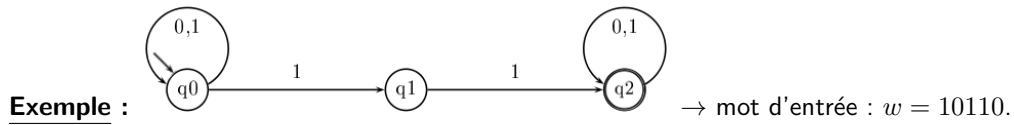
Le *non-déterminisme* est une notion importante, surtout pour les machines de **Turing**. Pour rappel, jusqu'à présent les automates utilisés étaient déterministes en ce sens que pour tout mot, il n'existe qu'un seul calcul possible pour ce mot à partir de l'état initial. En d'autres mots, δ est une *fonction totale* et $\delta(q_0, w)$ donne un unique état.

Ici, on va permettre pour un mot w plusieurs calculs possibles, en modifiant la définition de δ .

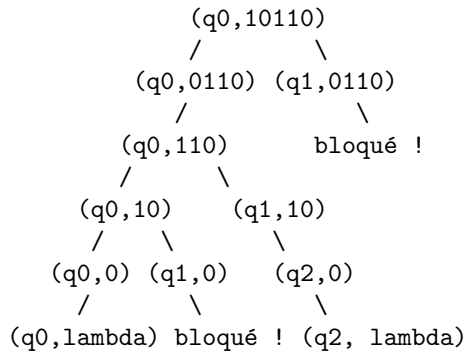
Définition : $M = (Q, \Sigma, \delta, q_0, F)$ est un *automate non-déterministe* si $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ (au lieu de $\delta : Q \times \Sigma \rightarrow Q$ pour les automates déterministes, les définitions pour Q, Σ, q_0 et F restent identiques).

4.3.1 Vocabulaire et définitions revisitées

	Automates déterministes	Automates non-déterministes
Instructions	select input = a_1 goto i_1 input = a_2 goto i_2 ... input = a_k goto i_k	select input = a_1 goto i_1 or goto j_1 or input = a_k goto i_k or goto j_k ...
Configuration	$(q, u) \in Q \times \Sigma^*$	$(q, u) \in Q \times \Sigma^*$
Étape de calcul	$(q, u) \vdash_M (p, v)$ ($u = av$, p unique) Calcul unique sur le mot d'entrée w $(q_0, w) \vdash_* (p, \lambda)$ (p unique) w accepté si $(q_0, w) \vdash_* (p, \lambda)$ avec $p \in F$	$(q, u) \vdash_M (p_i, v)$ ($u = av$) $\delta(q, a) = \{p_1, \dots, p_k\}$ (<i>plus d'unicité!</i>) L'automate choisit de poursuivre le calcul avec p_i , $1 \leq i \leq k$. Remarque : Si $\delta(q, a) = \{\}$, alors le calcul est <i>bloqué</i> et u ne peut pas être traité par l'automate à partir de l'état q . Plusieurs calculs sur le mot d'entrée w sont possibles, dont certains ont été bloqués. un <i>arbre de calcul</i> sur l'entrée w est utilisé, il s'agit d'un arbre reprenant tous les calculs possibles, y compris ceux qui sont bloqués. w accepté si parmi tous les calculs sur l'entrée w , il en existe un de la forme $(q_0, w) \vdash_* (p, \lambda)$ avec $p \in F$
		
		Dans ce cas, l'automate choisit ce calcul pour montrer qu'il accepte w .



Voici l'arbre de calcul sur l'entrée w :



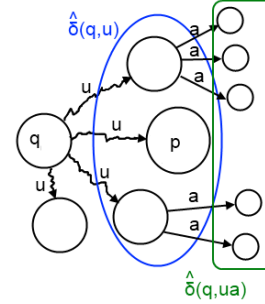
Vu que l'on a $q_2 \in F$, le chemin choisit par l'automate est celui se terminant par (q_2, λ) car le calcul est terminé et w est ainsi accepté.

4.3.2 Langage calculé par un automate non-déterministe

$$L(M) = \{w \in \Sigma^* \mid \exists \text{ un calcul sur l'entrée } w \text{ de la forme } (q_0, w) \vdash_* (p, \lambda) \text{ avec } p \in F\}$$

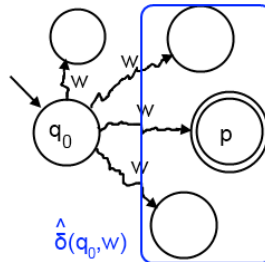
Extension de :

- $\hat{\delta}(q, \lambda) = \{q\}$
- $\hat{\delta}(q, u)$ déjà calculé
- $\hat{\delta}(q, ua) = \{p \in Q \mid \exists r \in \hat{\delta}(q, u) \text{ et } p \in \delta(r, a)\}$



⇒ $\hat{\delta}$ défini par récurrence.

$$\text{Donc, on a aussi : } L(M) = \{w \in \Sigma^* \mid \exists p \in F, p \in \hat{\delta}(q_0, w)\}$$

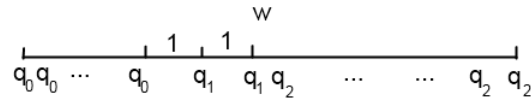


Le langage calculé par l'exercice précédent est donc : $L(M) = \{w \in \Sigma_{bool}^* \mid w \text{ possède le facteur } 11\}$.

Preuve :

1. $L(M) \subseteq \{w \in \Sigma_{bool}^* \mid w \text{ possède le facteur } 11\}$?

Soit $w \in L(M)$, parmi les calculs sur w , il y en a un de la forme $(q_0, w) \vdash_* (q_2, \lambda)$



$\Rightarrow 11$ est bien facteur de w .

2. $\{w \in \Sigma_{bool}^* \mid w \text{ possède le facteur } 11\} \subseteq L(M)$?

Soit w de la forme $w = u11v$ où $u, v \in \Sigma_{bool}^*$. Parmi tous les calculs sur w , on considère le suivant :

$$(q_0, u11v) \vdash_* (q_0, 11v) \vdash_* (q_1, 1v) \vdash_* (q_2, v) \vdash_* (q_2, \lambda), q_2 \in F$$

$\Rightarrow w \in L(M)$.

4.3.3 Questions

1. Les automates non déterministes acceptent-ils plus de langages que les automates déterministes ?

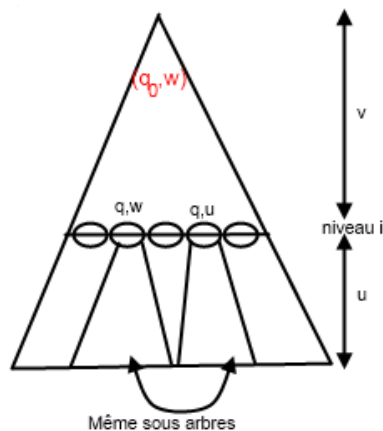
La réponse est *non*, on introduit alors le théorème suivant :

Soit M un automate non déterministe, alors il existe un automate déterministe M' t.q.

$$L(M') = L(M)$$

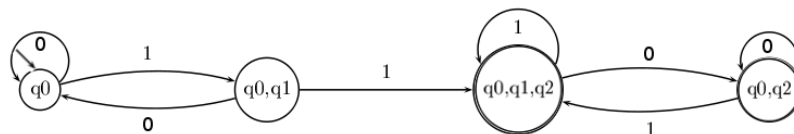
(il existe un algorithme pour transformer un automate non déterministe en automate déterministe)

L'idée de cet algorithme est de *simuler tous les calculs* possibles de l'automate non déterministe par un *parcours en largeur* de l'arbre de calcul sur w . w quelconque, arbre de calcul sur l'entrée w (par M) :



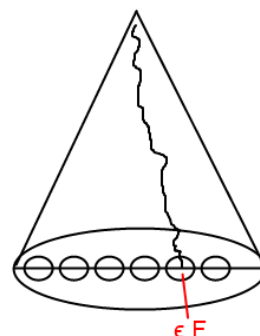
Soit i un niveau, l'automate déterministe M' doit retenir toutes les configurations du niveau $i \rightarrow$ pas vraiment, il suffit de retenir les différents états apparaissant dans les configurations de ce niveau. En effet, si un état q apparaît au moins 2 fois, c'est dans des configurations identiques de la forme (q, u) où u est le reste du mot w à traiter et le sous arbre de racine (q, u) est le même pour chaque configuration (q, u) du niveau i .

Exemple : construction de M' sur l'exemple précédent



$F' = ?$

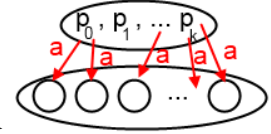
états de M' = ensemble des états du dernier niveau de l'arbre de calcul, il faut que cet ensemble



En général la construction de M' à partir de M se fait comme suit :

Soit $M = (Q, \Sigma, \delta, q_0, F)$, montrons comment construire $M' = (Q', \Sigma, \delta', q'_0, F')$,

- $Q' = \mathcal{P}(Q) = \{ \langle P \rangle \mid P \subseteq Q \}$ (chaque état de M' mémorise les états de M apparaissant à un niveau donné d'un arbre de calcul.
- $q'_0 = \{q_0\}$ (noté $\langle q_0 \rangle$ dans le livre)
- $F' = \{ \langle P \rangle \in Q' \mid P \cap F \neq \emptyset \}$, un état $\langle P \rangle$ est final dans M' s'il est de la forme $P = \{p_1, p_2, \dots, p_k\}$ avec l'un des $p_i \in F$



- $\delta' : Q' \times \Sigma \rightarrow Q', \delta'(\langle P \rangle, a) = \langle R \rangle$ tq $R = \{r \in Q \mid \exists p \in P, r \in \delta(p, a)\}$

Il faudrait encore prouver que $L(M) = L(M')$, mais voir livre.

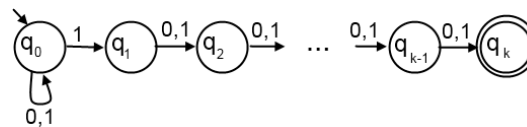
Commentaires sur ce théorème :

- les automates non déterministes ne sont pas plus puissants que les automates déterministes,
- il existe un algorithme de construction de M' à partir de M ,
- la taille de M' , si M a n états ($|Q| = n$) alors M' a 2^n états ($|\mathcal{P}(Q)| = 2^n$), **taille exponentielle**, mais l'exemple précédant montre que si l'on se limite aux états accessibles de l'état initial, la taille reste convenable.
- le théorème suivant est négatif : soit L_k défini comme suit :

$$L_k = \{x1y \mid x \in \Sigma_{bool}^*, y \in \Sigma_b^{k-1}ool\}, k \geq 1 \text{ fixé}$$

$$= \{w \in \Sigma_{bool}^* \mid w \text{ possède un 1 en position } k \text{ en partant de la fin}\}$$

L_k est accepté par l'automate non-déterministe suivant :



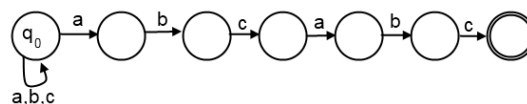
Alors, l'automate déterministe M'_k construit par l'algorithme précédent a au moins 2^k états parmi les états accessibles de l'état initial.

2. Les automates non-déterministes sont peu naturels, quel est leur intérêt ?

C'est un concept intéressant car pour certains exemples, il est plus facile de trouver un automate **non-déterministe**. Dans ces exemples, on compte le langage L_k ci-dessus ou encore en exercices, on a vu :

$$\{w \in \{a, b, c\}^* \mid w = xabcabc, x \in \{a, b, c\}^*\}$$

Nous avons du avoir une certaine réflexion afin d'obtenir l'automate déterministe alors que l'automate non-déterministe est très simple :

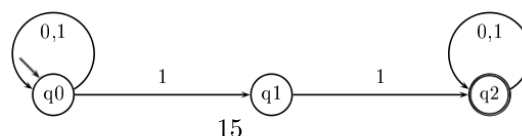


(Si on applique l'algorithme pour obtenir un automate déterministe, on retombe sur celui que l'on a trouvé en exercice, dans ce cas-ci, on a donc pas d'explosion de taille)

Rappel important : si w est accepté par un automate non-déterministe, parmi tous les calculs possibles sur ce w , l'automate sait lequel choisir pour accepter w .

3. Étant donné un langage régulier, comment peut-on montrer que tout automate déterministe qui l'accepte a une taille $\geq c$ où c est une constante bien choisie ?

Nous allons le montrer sur un exemple simple : $L = \{x11y \mid x, y \in \{0,1\}^*\}$, son automate non-déterministe est simple :



Proposition : *Tout automate déterministe acceptant L a une taille ≥ 3*

Rappel : Lemme 3.12

Soit $M = (Q, \Sigma, \delta, q_0, F)$ un automate fini, soit $x, y \in \Sigma^$, $x \neq y$ et tels que $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$,
alors $\forall z \in \Sigma^*$, $xz \in L(M) \Leftrightarrow yz \in L(M)$*

Preuve : soit M un automate déterministe acceptant L , donc M est de la forme $M = (Q, \{0, 1\}, \delta, q_0, F)$.

Intuition : exhiber 3 états nécessaires pour mettre en évidence une certaine information associée à ces états. (l'automate doit tester si le facteur **11** figure dans le mot d'entrée w)

$$\left\{ \begin{array}{ll} 1^{er} \text{ état} & : \text{ pas vu de '1'} \\ 2^e \text{ état} & : \text{ vu un '1'} \\ 3^e \text{ état} & : \text{ vu deux '1'} \end{array} \right.$$

Plus proprement, prenons $x = \lambda$, $y = 1$, on a que $\hat{\delta}(q_0, \lambda) = q_0$ et $\hat{\delta}(q_0, 1) = q_1$, et on veut montrer que $q_0 \neq q_1$. Par l'**absurde**, supposons que $q_0 = q_1$, par le **lemme 3.12**, avec $z = 1$, on a donc $xz(= 1) \in L \Leftrightarrow yz(= 11) \in L$ ce qui est impossible car $11 \in L$ alors que $1 \notin L$.

$$\Rightarrow \boxed{q_0 \neq q_1}$$

Soit $\hat{\delta}(q_0, 11) = q_2$, on veut montrer que $q_2 \neq q_0$ et $q_2 \neq q_1$.

Par l'**absurde**, supposons $q_2 = q_0$, par le **lemme 3.12**, avec $z = 0$, on a donc $xz(= 0) \in L \Leftrightarrow yz(= 110) \in L$ ce qui est impossible car $110 \in L$ alors que $0 \notin L$.

$$\Rightarrow \boxed{q_0 \neq q_2}$$

Par l'**absurde**, supposons $q_2 = q_1$, par le **lemme 3.12**, avec $z = 0$, on a donc $xz(= 10) \in L \Leftrightarrow yz(= 110) \in L$ ce qui est impossible car $110 \in L$ alors que $10 \notin L$.

$$\Rightarrow \boxed{q_1 \neq q_2}$$

\Rightarrow **3 états distincts** q_0 , q_1 et q_2 .

□