Alphabet et Mots

Alphabet

- un alphabet est un ensemble ***fini*** et ***non vide*** de symboles appelés ***lettres***

Exemples d'alphabet

$X_1 = \{a, b, c, d\}$, $X_2 = \{0, 1\}$ $X_3 = \{\oplus, \ominus, \otimes, \odot\}$

Mot

- un mot sur un alphabet X est une ***séquence finie***, éventuellement vide, de lettres de X
- pour désigner le **mot vide** (séquence vide : aucune lettre) on utilise le symbole ε

Quelques exemples de mots

- sur l'alphabet X_1 : *babaca*, *a*, *bca*, ε, *d*
- sur l'alphabet X_2 : 001, 100001, ε, 0, 1

Alphabet et Mots

Concaténation (de mots)

- la concaténation des deux mots u et v est un mot constitué des lettres de u puis de celles de v.
- l'opérateur de concaténation est le point

Exemples

si $u = ab$ et $v = bca$, alors $u.v = abbca$ et $v.u = bcaab$
si $u = aba$ et $v = \varepsilon$, alors $u.v = aba$
si $u = \varepsilon$ et $v = \varepsilon$, alors $u.v = \varepsilon$

ε est l'élément neutre de la concaténation

$u.\varepsilon = \varepsilon.u = u$

La concaténation est associative

- $(u.v).w = u.(v.w)$ On peut donc noter $u.v.w$

Longueur d'un mot

Définitions : longueur d'un mot / nombre d'occurrences

Si u est un mot et x une lettre de l'alphabet X

- $|u|$ désigne sa **longueur** (nombre de lettres)
- $|u|_x$ désigne le nombre d'occurrences de x dans u.

Exemples

$|babaca| = 6$, $|a| = 1$, $|bca| = 3$, $|\varepsilon| = 0$, $|100001| = 6$
$|babaca|_a = 3$, $|a|_b = 0$, $|bca|_d = 0$, $|\varepsilon|_a = 0$, $|100001|_0 = 4$

Propriétés

- $|u.v| = |u| + |v|$
- $\forall x \in X, |u.v|_x = |u|_x + |v|_x$

Alphabet et Mots

Itération de la concaténation

La notation u^i (où u est un mot et i un entier naturel) désigne le mot défini par

- $u^0 = \varepsilon$
- $u^{i+1} = u^i.u = u.u^i$ pour $i \geq 0$

Exemples

si $u = ab$, alors $u^0 = \varepsilon$, $u^1 = ab$, $u^2 = abab$, $u^3 = ababab$
$\varepsilon^n = \varepsilon$ pour tout $n \geq 0$

Alphabet et Mots

Facteur (définition formelle)

- un mot u est un **facteur** du mot v ssi il existe des mots p et s tels que $v = p.u.s$
- Si $\exists s$ tq $v = u.s$, alors u est un **facteur gauche** (ou **préfixe**) de v.
- Si $\exists p$ tq $v = p.u$, alors u est un **facteur droit** (ou **suffixe**) de v.
- Si u est facteur de v et $u \neq \varepsilon$ et $u \neq v$, alors u est un **facteur propre** de v.

Facteurs (définition non formelle)

- u facteur de v : u est « sous-chaîne » de v
- u préfixe (ou facteur gauche) de v : v « commence par » u
- u suffixe (ou facteur droit) de v : v « se termine par » u

Alphabet et Mots

Facteurs : exemples

ab est facteur de *aaba* mais il n'en est ni f. gauche, ni f. droit.

ab est facteur gauche (donc facteur) de *aba* mais il n'en est pas f. droit..

ab est facteur gauche et droit de *abbab*

ab est facteur gauche et droit de *ab* (mais ce n'est pas un facteur propre)

ε est facteur gauche et droit de *abbab* (mais ce n'est pas un facteur propre)

Langages

Langage

- Un **langage** sur un alphabet X est un **ensemble de mots** sur X.
- NB : un langage n'est **pas nécessairement** un ensemble fini

Exemples de langages sur l'alphabet $X = \{a, b\}$

$L_1 = \{ab, ba\}$ (2 *mots*) $\quad L_2 = \{ab, ba, \varepsilon\}$ (3 *mots*)
$L_3 = \{\varepsilon\}$ (1 *mot*) $\quad L_4 = \emptyset$ (0 *mot*)
$L_5 = \{(ab)^i \mid i \geq 0\}$ (infinité de mots) $\quad L_5 = \{\varepsilon, ab, abab, ...\}$

Langages

Concaténation (de langages)

Si L et L' sont des langages sur X,

$$L.L' = \{u.v | u \in L, v \in L'\}$$

Exemple sur l'alphabet X={a,b,c}

$\{a, ab, ba\}.\{c, ca\} = \{ac, abc, bac, aca, abca, baca\}$
$\{a, ab, ba\}.\{c, \varepsilon\} = \{ac, abc, bac, a, ab, ba\}$
$\{a, ab, ba\}.\{\varepsilon\} = \{a, ab, ba\}$
$\{a, ab, ba\}.\{b, \varepsilon\} = \{ab, abb, bab, a, ba\}$

La concaténation est associative

$L_1.(L_2.L_3) = (L_1.L_2).L_3 = L_1.L_2.L_3$

Langages

Élément neutre

$\{\varepsilon\}$ est l'élément neutre de la concaténation de langages.

$$\forall L,\ \{\varepsilon\}.L = L.\{\varepsilon\} = L$$

Élément absorbant

$\emptyset$ est l'élément absorbant de la concaténation de langages.

$$\forall L,\ \emptyset.L = L.\emptyset = \emptyset$$

Langages

Itération de la concaténation de langages

- $L^0 = \{\varepsilon\}$
- $L^{i+1} = L^i.L = L.L^i, \quad i \geq 0$

Clôture de Kleene

- $\bigcup_{i \geq 0} L^i$ est la **clôture de Kleene**
- On note $\mathbf{L^* = \bigcup_{i \geq 0} L^i}$ (notation « étoile de Kleene »)

NB :

- $\{\varepsilon\}^* = \{\varepsilon\}$
- $\emptyset^* = \{\varepsilon\}$, par définition.
- si L contient un mot non vide, alors L^* est infini.

Langages

Exemples

$\{ba\}^3 = \{bababa\}$

$\{ba, c\}^3 = \{bababa, babac, bacba, bacc, cbaba, cbac, ccba, ccc\}$

$\{ba\}^* = \{\varepsilon, ba, baba, bababa, babababa, ...\}$ (langage infini)

$\{ba, c\}^* = \{\varepsilon, ba, c, baba, bac, cba, cc, bababa, babac, bacba, ...\}$
(langage infini)

Langages

Monoïde

- Si X est un alphabet, $X^* = \{mots\ sur\ l'alphabet\ X\}$
- X^* est donc une expression simple pour désigner l'ensemble des mots utilisant l'alphabet X
- X^* est appelé **monoïde libre** engendré par X.

« étoile stricte »

- On note $\mathbf{L^+ = \bigcup_{i \geq 1} L^i}$

Autre définition, équivalente :

- $L^+ = L^*.L = L.L^*$

Langages rationnnels

Expression rationnelle (également appelée régulière)

Un expression rationnelle est formée par

- symboles atomiques :
  - les lettres de l'alphabet
  - le symbole ε
  - le symbole $\emptyset$
- les 3 opérateurs : $\boxed{* \quad . \quad +}$:
  - l'**étoile** (utilisée en exposant) * (le plus prioritaire)
  - le **point** opérateur infixé
  - le **plus** opérateur infixé (le moins prioritaire).
- parenthèses (correctement formées)

Exemples d'expressions rationnelles, alphabet $X = \{a, b, c\}$

$a, \quad a + b, \quad (a + b), \quad a^*, \quad a.b^*, \quad (a.b)^*, \quad (a + b).(b + c).a^*$

Langages rationnnels

Langage associé à une expression rationnelle

Une expression rationnelle est utilisée pour **désigner** un langage

Expressions atomiques :

expression rationnelle	*langage associé*
x (lettre)	$\{x\}$
$\emptyset$	ensemble vide
ε	$\{\varepsilon\}$

Sémantique des opérateurs :

*	clôture de Kleene
$+$	union
$.$	concaténation

Tenir compte de la priorité des opérateurs et du parenthésage

Langages rationnnels

Exemples (alphabet $\{a, b, c\}$)

expr. rat.	*langage associé*
a	$\{a\}$
ε	$\{\varepsilon\}$
$a + b$	$\{a\} \cup \{b\} = \{a, b\}$
$a.b$	$\{a\}.\{b\} = \{ab\}$
a^*	$\{a\}^* = \{\varepsilon, a, aa, ...\}$
$(a.b)^*$	$\{\varepsilon, ab, abab, ababab, , ...\}$
$a + b.c$	$\{a\} \cup (\{b\}.\{c\}) = \{a, bc\}$
$(a + b).c$	$(\{a\} \cup (\{b\}).\{c\} = \{a, b\}.\{c\} = \{ac, bc\}$
$(a + b).(a + b + c)$	$\{a, b\}.\{a, b, c\} = \{aa, ba, ab, bb, ac, bc\}$
$(a + b).c^*$	$\{a, b\}.(\{c\}^*) = \{a, b, ac, bc, acc, bcc, ...\}$

Langages rationnnels

Exemples (alphabet $\{a, b, c\}$)

exp. rat.	*description des mots du langage associé*
a	le mot a
ε	le mot vide
$a + b$	les mots a et b
$a.b$	le mot ab
a^*	les répétions (éventuellement vides) de a
$(a.b)^*$	les répétions (éventuellement vides) de ab
$a + b.c$	le mot a ou le mot bc
$(a + b).c$	mots constitués d'un a ou un b, suivi d'un c
$(a + b).(a + b + c)$	mots constitués d'un a ou un b suivi de a,b ou c
$(a + b).c^*$	mots constitués d'un a ou un b suivi d'une répétition quelconque de c

Langages rationnnels

Variantes usuelles ed'expressions rationnelles :

Les expressions rationnelles sont souvent « étendues » de la façon suivante :

- Omission possible de l'opérateur point :
  $\mathcal{L}(e_1 e_2) = \mathcal{L}(e_1.e_2)$
- opérateur ^{i} (exposant) pour $i \geq 0$ (même priorité que *)
  $\mathcal{L}({e_1}^i) = \mathcal{L}(e_1)^i$
  si $\nexists e'$ *et* e'' *tq* $e_1 = e' + e''$ ou $e_1 = e'.e''$
- opérateur $^+$ (comme exposant, ne pas confondre avec $+$)
  $\mathcal{L}({e_1}^+) = \mathcal{L}(e_1)^+ = \mathcal{L}(e_1).\mathcal{L}(e_1)^*$
  si $\nexists e'$ *et* e'' *tq* $e_1 = e' + e''$ ou $e_1 = e'.e''$

NB : Les expressions implémentées dans les composants logiciels ont leur propre syntaxe (cf TDM)

Langages rationnnels

Langage associé à une expression rationnelle

À chaque expression rationnelle e est associé un langage $\mathcal{L}(e)$ défini par

1. $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(x) = \{x\}$ $\forall x \in X$
2. $\mathcal{L}((e_1)) = \mathcal{L}(e_1)$
3. $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$
4. $\mathcal{L}(e_1.e_2) = \mathcal{L}(e_1).\mathcal{L}(e_2)$
   si $\nexists e'$ *et* e'' *tq* $e_1 = e' + e''$ *ou* $e_2 = e' + e''$
5. $\mathcal{L}({e_1}^*) = \mathcal{L}(e_1)^*$
   si $\nexists e'$ *et* e'' *tq* $e_1 = e' + e''$ ou $e_1 = e'.e''$

$\mathcal{L}(e)$ est appelé **langage dénoté par e**

Langages rationnels

Langages dénotés par une expression rationnelle

- les expressions rationnelles (ou régulières) ont été introduites pour représenter des langages
- **QUESTION** : tout langage peut-il être représenté par une expression régulière ?

Réponse : NON

- certains langages ne peuvent pas être représentés par une expression régulière
- ils sont « trop complexes à décrire » pour cela (c'est une façon de voir les choses)
- un exemple (sans preuve) :
  $\{a^n b^n | n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, ...\}$

Langages rationnels

Définition 1

- Un **langage rationnel** est un langage qui peut être dénoté par une expression rationnelle
- On appelle RAT la **famille des langages rationnels**

Propriété

- RAT est la plus petite famille de langages qui
  - contient $\emptyset, \{\varepsilon\}, \{x\}, \forall x \in X$
  - est close par union, concaténation et étoile de Kleene

Propriété

- Tout langage fini $\in RAT$

Propriété

- Il existe des langages non rationnels (la preuve sera apportée plus tard).

Automates finis déterministes

Définition

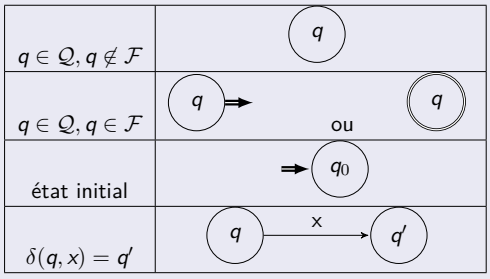
Un automate fini déterministe est défini par

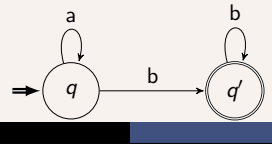
- un alphabet X
- un ensemble fini non vide $\mathcal{Q}$ appelé **ensemble d'états**
- un **état initial** $q_{ini} \in \mathcal{Q}$
- un sous-ensemble $\mathcal{F} \subseteq \mathcal{Q}$ d'états dits **acceptants** (ou encore **finals** ou encore **terminaux**)
- une fonction $\delta : \mathcal{Q} \times X \to \mathcal{Q}$ appelée **fonction de transition**

- Notation : $\delta(q_d, x) = q_a$ pourra être noté $q_d \xrightarrow{x} q_a$
- q_d est appelé état de départ et q_a état d'arrivée

Automates finis déterministes

Représentation graphique



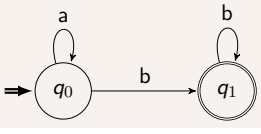


Automates finis déterministes

Exemple : automate A_1

- alphabet $\Sigma = \{a, b\}$
- ensemble d'états $Q = \{q_0, q_1\}$
- ensemble d'états acceptants $\mathcal{F} = \{q_1\}$
- état initial : q_0
- $\delta(q_0, a) = q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$:

δ	a	b
q_0	q_0	q_1
q_1		q_1

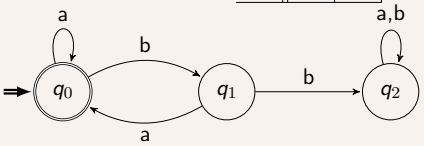


Automates finis déterministes

Exemple : automate A_2

- alphabet $\Sigma = \{a, b\}$
- ensemble d'états $Q = \{q_0, q_1, q_2\}$
- ensemble d'états acceptants $\mathcal{F} = \{q_0\}$
- état initial : q_0
- $\delta(q_0, a) = q_0 \quad \delta(q_0, b) = q_1$
  $\delta(q_1, a) = q_0 \quad \delta(q_1, b) = q_2$
  $\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2



Automates finis déterministes

Automate : « machine » à lire des mots

on part de l'état initial et on lit le mot de gauche à droite ; à chaque lettre lue on change d'état en appliquant la fonction de transition.

exemple

abb dans l'automate A_1 :
$q_0 \xrightarrow{a} \delta(q_0, a) = q_0 \xrightarrow{b} \delta(q_0, b) = q_1 \xrightarrow{b} \delta(q_1, b) = q_1$

ba dans l'automate A_1 :
$q_0 \xrightarrow{b} \delta(q_0, b) = q_1 \xrightarrow{a} \delta(q_1, a)$ indéfini (échec de lecture)

ba dans l'automate A_2 :
$q_0 \xrightarrow{b} \delta(q_0, b) = q_1 \xrightarrow{a} \delta(q_1, a) = q_0$

ε dans l'automate A_2 :
q_0

Automates finis déterministes

Langage reconnu par un automate (informellement)

Ensemble des mots qui peuvent être lus **en entier** dans l'automate, **en partant de l'état initial** et qui **après lecture complète amènent dans un état acceptant**.

exemple (automate A_1)

- *abb* est reconnu
- *ba* n'est pas reconnu (lecture du dernier *a* impossible)
- *a* n'est pas reconnu (on ne termine pas dans un état acceptant).
- ε n'est pas reconnu (on ne termine pas dans un état acceptant).

exemple (automate A_2)

- *abb* n'est pas reconnu
- *ba* est reconnu
- *a* est reconnu
- ε est reconnu

Automates finis déterministes

Extension de la fonction de transition

La fonction de transition peut être étendue aux mots :

$$\begin{array}{rcl} \hat{\delta} : \mathcal{Q} \times X^* & \to & \mathcal{Q} \\ \hat{\delta}(q, \varepsilon) & = & q \\ \forall x \in X, \forall w \in X^* \quad \hat{\delta}(q, w.x) & = & \delta(\hat{\delta}(q, w), x) \end{array}$$

$\hat{\delta}(q, w)$ **désigne l'état atteint en lisant** w **à partir de** q

- quand $q' = \hat{\delta}(q, w)$ on pourra utiliser la notation $q \xrightarrow[*]{w} q'$

exemples

- pour A_1 : $\hat{\delta}(q_0, abb) = q_1$, $\hat{\delta}(q_0, ba)$ indéfini $\hat{\delta}(q_0, \varepsilon) = q_0$
- pour A_2 : $\hat{\delta}(q_0, abb) = q_2$, $\hat{\delta}(q_0, ba) = q_0$ $\hat{\delta}(q_0, \varepsilon) = q_0$
  $\hat{\delta}(q_2, abb) = q_2$, $\hat{\delta}(q_1, aba) = q_0$ $\hat{\delta}(q_1, \varepsilon) = q_1$

Automates finis déterministes

Langage reconnu

Le langage reconnu par un automate déterministe $\mathcal{A} = (X, \mathcal{Q}, \delta, q_{ini}, \mathcal{F})$ est :

$$\mathcal{L}(\mathcal{A}) = \{w \in X^* | \hat{\delta}(q_{ini}, w) \in \mathcal{F}\}$$

Remarque : propriété

$\varepsilon \in \mathcal{L}(\mathcal{A}) \iff q_{ini} \in \mathcal{F}$

Automates finis déterministes

Automate déterministe complet

Un automate déterministe A est dit **complet** si sa fonction de transition δ est définie $\forall (q, x) \in \mathcal{Q} \times X$

- A_1 n'est pas complet
- A_2 est complet

Attention : non complet ne veut pas dire incorrect. A_1 **est un automate correct.**

Propriété

Tout langage reconnu par un automate peut être reconnu par un automate complet.

Si un automate déterministe A n'est pas complet, il existe un algorithme simple permettant de construire un automate déterministe complet qui reconnaît le même langage que A

Automates finis déterministes : algo de reconnaissance

Require : A est un automate fini déterministe
Require : u est un mot quelconque.
Si $u \neq \varepsilon$, on appelle x_i $(0 \leq i < |u|)$ sa lettre de rang i
Ensure : Renvoie TRUE si et seulement si u est accepté par A.

$etatCourant \leftarrow$ état initial de A ;
$index = 0$;
while $(index < |u|)$ **do**
  $lettre \leftarrow x_{index}$;
  if $(\delta(etatCourant, lettre)$ est indéfini$)$ **then**
    return FALSE ;
  end if
  $etatCourant \leftarrow \delta(etatCourant, lettre)$;
  $index \leftarrow index + 1$;
end while
return $etatCourant \overset{?}{\in} \mathcal{F}$;

Automates finis déterministes

État accessible

Soit A, un automate. Un état q est dit **accessible** s'il existe un mot w tel que $\hat{\delta}(q_{ini}, w) = q$

Automate accessible

Un automate est dit **accessible** si tous ses états sont accessibles.

Propriété

Tout langage reconnu par un automate peut être reconnu par un automate accessible

État co-accessible

Soit A, un automate. Un état q est dit **co-accessible** s'il existe un mot w tel que $\hat{\delta}(q, w) \in \mathcal{F}$

Automates finis déterministes

Automate co-accessible

Un automate est dit **co-accessible** si tous ses états sont co-accessibles.

Propriété

Tout langage reconnu par un automate peut être reconnu par un automate co-accessible

Automate émondé

Un automate est dit **émondé** s'il est **accessible ET co-accessible**

Propriété

Tout langage reconnu par un automate peut être reconnu par un automate émondé

Automates finis déterministes

- un automate fini déterministe (AFD) est un moyen de définir un langage : chaque automate définit implicitement son langage reconnu.
- tout langage peut-il être reconnu par un automate ?
  - NON (affirmation sans preuve... pour l'instant)
  - **on appelle** *REC* **l'ensemble des langages reconnaissables** par un AFD.
- y a-t-il un lien avec les langages réguliers (rationnels) ?
  - un langage régulier est-il toujours reconnaissable par un AFD ?
  - un langage reconnaissable par un AFD est-il toujours régulier ?

réponses à venir...

Déterminisme et non-déterminisme

Déterminisme

Une même situation produit **toujours** le même effet.
Exemple, dans un automate déterministe :

- un seul point de départ dans l'automate (l'état initial)
- pour un état et une lettre donnés : un seul état d'arrivée (ou indéfini)

$\Rightarrow$ aucun choix.

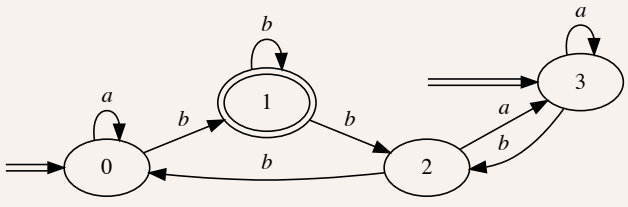
Non déterminisme

Dans une même situation, on peut avoir **des choix** à faire.
Dans un automate non déterministe :

- il **peut** y avoir un choix entre plusieurs points de départs.
- pour un état et une lettre donnés, il **peut** y avoir un choix entre plusieurs états d'arrivée.

Automate non déterministe

Exemple



Lectures possibles du mot *abb* :
$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_2$
$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1$
$q_3 \xrightarrow{a} q_3 \xrightarrow{b} q_2 \xrightarrow{b} q_0$
Le mot *abb* permet d'atteindre 3 états : q_2, q_1, q_0
abb est accepté car il permet d'atteindre un état acceptant (q_1)

Automates finis non déterministes

Définition

Un automate fini non déterministe est défini par

- un alphabet X
- un ensemble fini $\mathcal{Q}$ appelé **ensemble d'états**
- un **ensemble non vide d'états initiaux** : $Ini \in 2^{\mathcal{Q}}, Ini \neq \emptyset$
- un sous-ensemble $\mathcal{F} \subseteq \mathcal{Q}$ d'états dits **acceptants** (ou encore **finals** ou encore **terminaux**)
- une fonction $\delta : \mathcal{Q} \times X \to 2^{\mathcal{Q}}$ appelée **fonction de transition**

$\mathcal{Q} = \{0, 1, 2, 3\},\ \mathcal{I}ni = \{0, 3\},\ \mathcal{F} = \{1\},$

δ	a	b
0	$\{0\}$	$\{1\}$
1	$\emptyset$	$\{1, 2\}$
2	$\{3\}$	$\{0\}$
3	$\{3\}$	$\{2\}$

Automates finis non déterministes

Extension de la fonction de transition

La fonction de transition peut être étendue aux mots :

$$\begin{array}{rcl} \hat{\delta} : \mathcal{Q} \times X^* & \rightarrow & 2^{\mathcal{Q}} \\ (q, \varepsilon) & \mapsto & \{q\} \\ (q, w.x) & \mapsto & \bigcup_{q' \in \hat{\delta}(q,w)} \delta(q', x) \ (x \in X, w \in X^*) \end{array}$$

Langage reconnu

Le langage reconnu par un automate non déterministe $\mathcal{A} = (X, \mathcal{Q}, \delta, Ini, \mathcal{F})$ est défini par

$$\mathcal{L}(\mathcal{A}) = \{w \in X^* | \exists q_{ini} \in Ini, \hat{\delta}(q_{ini}, w) \cap \mathcal{F} \neq \emptyset\}$$

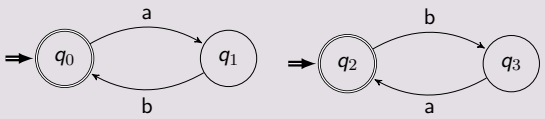
Équivalence entre automates finis déterministes et non déterministe ?

Pourquoi des automates non déterministes ?

- conception parfois plus facile .

exemple

Automate non déterministe pour $(ab)^* + (ba)^*$:



- Outil formel plus aisé à manipuler (ex union, concaténation, kleene)

Équivalence entre automates finis déterministes et non déterministe ?

Comparaison des outils

- un automate non déterministe permet-il de reconnaître tous les langages reconnus par un automate déterministe ?
- un automate déterministe permet-il de reconnaître tous les langages reconnus par un automate non déterministe ?
- et les langages réguliers ? peuvent-ils être reconnus par un automate déterministe ? par un non-déterministe ?

Notations

- Rappel : REC est l'ensemble des langages pouvant être reconnus par un automate déterministe.
- On note REC_{ND} l'ensemble des langages pouvant être reconnus par un automate NON-déterministe.

Équivalence entre automates finis déterministes et non déterministe ?

$REC \subseteq REC_{ND}$

Pour tout automate fini déterministe A, il existe un automate fini non déterministe A_{nd} qui reconnaît le même langage

A_{nd} peut être défini de façon triviale par

- $\mathcal{Q}_{A_{nd}} = \mathcal{Q}_A$
- $Ini_{A_{nd}} = \{q_{ini}\}$
- $\mathcal{F}_{A_{nd}} = \mathcal{F}_A$
- $\delta_{A_{nd}}(q, x) = \{\delta_A(q, x)\}$ si $\delta_A(q, x)$ est défini.
- $\delta_{A_{nd}}(q, x) = \emptyset$ si $\delta_A(q, x)$ est indéfini.

Équivalence entre automates finis déterministes et non déterministe ?

$REC_{ND} \subseteq REC$

Pour tout automate fini **non** déterministe A, il existe un automate fini déterministe A_d qui reconnaît le même langage

Automate déterministe équivalent

Pour un automate non déterministe A, un automate **déterministe** équivalent A_d peut être défini par

- $\mathcal{Q}_{A_d} = 2^{\mathcal{Q}_A}$
- $q_{ini} = Ini_A$
- $\mathcal{F}_{A_d} = \{q \in \mathcal{Q}_{A_d}, q \cap \mathcal{F}_A \neq \emptyset\}$
- $\forall q \in 2^{\mathcal{Q}_A}, \forall x \in X, \quad \delta_{A_d}(q, x) = \bigcup_{e \in q} \delta_A(e, x)$

Cette construction est formelle, simple à exprimer et apporte le résultat théorique mais elle définit beaucoup d'états inutiles

Équivalence entre automates finis déterministes et non déterministe ?

OUI

- $REC = REC_{ND}$

Équivalence entre automates finis déterministes et non déterministe

Algorithme de déterminisation

- permet de calculer pour chaque automate non déterministe un automate déterministe équivalent
- l'algorithme présenté ci-dessous fournit un automate déterministe **accessible et complet**.
- attention : si l'automate déterministe possède n états, l'automate déterministe calculé peut en posséder 2^n (dans le pire des cas).

Déterminisation

Déterminisation de l'exemple précédent

init ?	*accept ?*	*state*	*transition table*	
			a	*b*
=>		*{0,3}*	*{0,3}*	*{1,2}*
	<=	*{1,2}*	*{3}*	*{0,1,2}*
		{3}	*{3}*	*{2}*
	<=	*{0,1,2}*	*{0,3}*	*{0,1,2}*
		{2}	*{3}*	*{0}*
		{0}	*{0}*	*{1}*
	<=	*{1}*	*∅*	*{1,2}*
		∅	*∅*	*∅*

Déterminisation

Algorithme

$q_{ini} \leftarrow Ini_A$
$\mathcal{Q}_{A_d} \leftarrow \{Ini_A\}$
while $\exists (q,x) \in \mathcal{Q}_{A_d} \times X$, $\delta_{A_d}(q,x)$ *is undefined* **do**
  $(q,x) \leftarrow$ one of such pair
  $q' \leftarrow \bigcup_{e \in q} \delta_A(e,x)$
  $\mathcal{Q}_{A_d} \leftarrow \mathcal{Q}_{A_d} \cup \{q'\}$
  $\delta_{A_d}(q,x) \leftarrow q'$
end while
$\mathcal{F}_{A_d} \leftarrow \{q \in \mathcal{Q}_{A_d}, q \cap \mathcal{F}_A \neq \emptyset\}$

- L'automate obtenu est **complet** et **accessible**.
- Il n'est, en général, **pas minimal**

Stabilité de REC par $\cup, ., ^*$

Théorème

REC est close par union, concaténation et étoile de Kleene.

Pour tous langages reconnaissables L_1 et L_2 ($\forall L_1, L_2 \in REC$)

- $L_1 \cup L_2$ est reconnaissable ($L_1 \cup L_2 \in REC$)
- $L_1.L_2$ est reconnaissable ($L_1.L_2 \in REC$)
- ${L_1}^*$ est reconnaissable (${L_1}^* \in REC$)

- soit A_1, un automate reconnaissant L_1 et possèdant un seul état initial. $A_1 = (Q_1, \{ini_1\}, \mathcal{F}_1, \delta_1)$
- Idem, $A_2 = (Q_2, \{ini_2\}, \mathcal{F}_2, \delta_2)$ reconnaissant L_2

On va construire un automate non déterministe pour $L_1 \cup L_2$, un pour $L_1.L_2$ et un pour ${L_1}^*$

Stabilité de REC par $\cup, ., ^*$

Un automate non déterministe pour $L_1 \cup L_2$

- $\mathcal{Q}_A = \mathcal{Q}_1 \cup \mathcal{Q}_2$
- $Ini_A = \{ini_1, ini_2\}$
- $\mathcal{F}_A = \mathcal{F}_1 \cup \mathcal{F}_2$
- $\delta_A(q,x) = \begin{cases} \delta_1(q,x) & \forall q \in \mathcal{Q}_1 \\ \delta_2(q,x) & \forall q \in \mathcal{Q}_2 \end{cases}$

reconnait le langage $L_1 \cup L_2$
Donc $L_1 \cup L_2 \in REC$.

Stabilité de REC par $\cup, ., ^*$

Un automate non déterministe pour $L_1.L_2$

- $\mathcal{Q}_A = \mathcal{Q}_1 \cup \mathcal{Q}_2$
- $Ini_A = \{ini_1\}$
- $\mathcal{F}_A = \mathcal{F}_2$ si $\varepsilon \notin L_2$ ou $\mathcal{F}_A = \mathcal{F}_2 \cup \mathcal{F}_1$ si $\varepsilon \in L_2$
- $\delta_A(q,x) = \begin{cases} \delta_1(q,x) & \forall q \in \mathcal{Q}_1 - \mathcal{F}_1 \\ \delta_1(q,x) \cup \delta_2(ini_2,x) & \forall q \in \mathcal{F}_1 \\ \delta_2(q,x) & \forall q \in \mathcal{Q}_2 \end{cases}$

reconnait le langage $L_1.L_2$
Donc $L_1.L_2 \in REC$.

Stabilité de REC par $\cup, ., ^*$

Un automate non déterministe pour $L_1{}^*$

- $\mathcal{Q}_A = \mathcal{Q}_1 \cup \{new\_ini\}$
- $Ini_A = \{new\_ini\}$
- $\mathcal{F}_A = \mathcal{F}_1 \cup \{new\_ini\}$
- $\delta_A(q,x) = \begin{cases} \delta_1(q,x) & \forall q \in \mathcal{Q}_1 - \mathcal{F}_1 \\ \delta_1(q,x) \cup \delta_1(ini_1,x) & \forall q \in \mathcal{F}_1 \\ \delta_1(ini_1,x) & si\ q = new\_ini \end{cases}$

reconnait le langage L_1^*
Donc $L_1^* \in REC$.

Stabilité de REC par $\cup, ., ^*$

Théorème

$RAT \subseteq REC$

- REC est close par union, concaténation et étoile de Kleene.
- REC contient $\emptyset, \{\varepsilon\}, \{x\}$ $(\forall x \in X)$
- RAT est la plus petite famille close par union, concaténation et étoile de Kleene contenant $\emptyset, \{\varepsilon\}, \{x\}$ $(\forall x \in X)$
- $RAT \subseteq REC$

Tout langage rationnel peut être reconnu par un automate.

Équations et langages

Une équation à une inconnue : L

Si A et B sont deux langages fixés, peut-on trouver un langage L vérifiant

$$L = A.L \cup B \quad ?$$

En existe-til plusieurs ? un seul ? aucun ?
Peut-on l(es) exprimer en fonction de A et B ?

Lemme d'Arden

Si A et B sont deux langages et si $\varepsilon \notin A$, alors
$L = A^*B$ est **le seul** langage vérifiant $L = A.L \cup B$

Corollaire

Si A et B sont 2 langages **réguliers** et si $\varepsilon \notin A$,
$A^*.B$, unique solution de $L = A.L \cup B$ est aussi un langage **régulier**

Équations et langages

2 équations et 2 inconnues

Si $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}, B_1, B_2$ sont des langages réguliers et si aucun des $A_{i,j}$ ne contient ε, alors le système d'équations à 2 inconnues L_1, L_2 :

$$\begin{array}{rcccccc} L_1 & = & A_{1,1}.L_1 & \cup & A_{1,2}.L_2 & \cup & B_1 \\ L_2 & = & A_{2,1}.L_1 & \cup & A_{2,2}.L_2 & \cup & B_2 \end{array}$$

admet une solution unique et les langages L_1 et L_2 sont réguliers.

preuve

$L_1 = A_{1,1}^*(A_{1,2}.L_2 \cup B_1)$ (lemme d'Arden)
$L_2 = A_{2,1}.A_{1,1}{}^*(A_{1,2}.L_2 \cup B_1) \cup A_{2,2}.L_2 \cup B_2$
$L_2 = (A_{2,1}.A_{1,1}{}^*A_{1,2} \cup A_{2,2}).L_2 \cup A_{2,1}.A_{1,1}^*.B_1 \cup B_2$
$L_2 = (A_{2,1}.A_{1,1}{}^*A_{1,2} \cup A_{2,2})^*(A_{2,1}.A_{1,1}^*.B_1 \cup B_2)$ (Arden)

Équations et langages

n équations et n inconnues

Le lemme d'Arden se généralise aux systèmes de n équations et n inconnues.

Si les langages $A_{i,j}$ et B_i sont réguliers et si aucun des $A_{i,j}$ ne contient ε, alors les L_i forment une solution unique et sont réguliers.

Automates et équations de langages

Langages L_q, définition

Soit un aut. déterministe $\mathcal{A} = (X, \mathcal{Q}, q_{ini}, \mathcal{F}, \delta)$ et un état $q \in Q$,

$$L_q = \{w \in X^* | \hat{\delta}(q,w) \in \mathcal{F}\}$$

L_q est l'ensemble des mots qui, lus en partant de q, atteignent un état acceptant

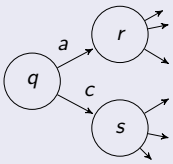
Propriété (pour un automate déterministe)

$$\mathcal{L}(\mathcal{A}) = L_{q_{ini}}$$

Automates et équations de langages

équation associée à un automate, principe

Exemple : alphabet $X = \{a, b, c\}$ et des états q, r, s tels que q non acceptant et $\delta(q, a) = r,\ \delta(q, c) = s,\ \delta(q, b)$ indéfini
Quels mots atteignent un état acceptant en partant de q ?



- mot vide : non
- commence par b : non
- un a suivi d'un mot allant de r jusqu'à un état acceptant
- un c suivi d'un mot allant de s jusqu'à un état acceptant

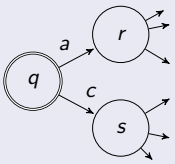
Donc

$$L_q = \{a\}.L_r \cup \{c\}.L_s$$

Automates et équations de langages

équation associée à un automate, principe

Exemple : alphabet $X = \{a, b, c\}$ et des états q, r, s tels que q **acceptant** et $\delta(q, a) = r,\ \delta(q, c) = s,\ \delta(q, b)$ indéfini
Quels mots atteignent un état acceptant en partant de q ?



- mot vide : oui
- commence par b : non
- un a suivi d'un mot allant de r jusqu'à un état acceptant
- un c suivi d'un mot allant de s jusqu'à un état acceptant

Donc

$$L_q = \{a\}.L_r \cup \{c\}.L_s \cup \{\varepsilon\}$$

Automates et équations de langages

On procède de même pour chaque état de l'automate

- une équation pour chaque L_q
- pour n états on obtient n equations à n inconnues.
- les facteurs des L_q sont des singletons $\{x\},\ x \in X$
  - ne contiennent pas le mot vide
  - sont des langages réguliers
- le lemme d'Arden s'applique : il existe une solution unique pour les L_q.
- les L_q sont réguliers
- $\mathcal{L}(A) = L_{q_{ini}}$ est régulier

Le langage reconnu par un automate est un langage régulier

Théorême de Kleene : $REC = RAT$

$REC \subseteq RAT$

Pour tout automate déterministe A

- Chaque L_q admet une solution unique, qui est un langage régulier (rationnel).
- $\mathcal{L}(A) = L_{q_{ini}}$ est un langage régulier

Donc $REC \subseteq RAT$

Théorème de Kleene

$$RAT = REC$$

Calcul de l'expression à partir d'un automate

Équations d'expressions régulières

- on sait maintenant que les langages L_q sont réguliers
- on établit une équation d'expressions régulières
  Par exemple :

$$L_q = a.L_r + b.L_s + \varepsilon$$

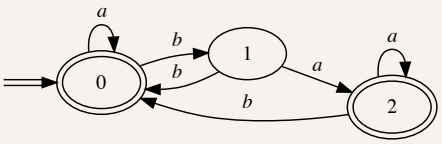
  au lieu de

$$L_q = \{a\}.L_r \cup \{c\}.L_s \cup \{\varepsilon\}$$

  pour calculer les expressions régulières, on peut :
  - dans l'équation $L_1 = ...$, éliminer L_1 en partie droite, en utilisant Arden si nécessaire.
  - dans l'équation $L_2 = ...$, éliminer L_1 et L_2 en partie droite, en utilisant Arden si nécessaire.
  - et ainsi de suite
  - on obtient une expression pour L_n.
  - puis pour chacun des L_i

Calcul de l'expression à partir d'un automate

Équations d'expressions régulières



$$\begin{array}{lll} (0) & L_0 & = aL_0 + bL_1 + \varepsilon \\ (1) & L_1 & = bL_0 + aL_2 \\ (2) & L_2 & = bL_0 + aL_2 + \varepsilon \end{array}$$

Calcul de l'expression à partir d'un automate

Exemple de résolution

$$
\begin{array}{lll}
(0) & L_0 = aL_0 + bL_1 + \varepsilon & \\
(1) & L_1 = bL_0 + aL_2 & \\
(2) & L_2 = bL_0 + aL_2 + \varepsilon & \\
 & & \\
(0') & L_0 = a^*(bL_1 + \varepsilon) & (Arden) \\
 & L_1 = ba^*(bL_1 + \varepsilon) + aL_2 & (1)\ et\ (0') \\
 & \quad = ba^*bL_1 + ba^* + aL_2 & \\
(1') & L_1 = (ba^*b)^*(ba^* + aL_2) & (Arden) \\
 & L_2 = L_1 + \varepsilon & (1)\ et\ (2) \\
 & \quad = (ba^*b)^*(ba^* + aL_2) + \varepsilon & (1') \\
 & \quad = (ba^*b)^*aL_2 + (ba^*b)^*ba^* + \varepsilon & \\
(2') & L_2 = ((ba^*b)^*a)^*((ba^*b)^*ba^* + \varepsilon) & (Arden) \\
 & L_1 = \ldots & \\
 & L_0 = \ldots &
\end{array}
$$

Calcul de l'expression à partir d'un automate

résolution : autre façon

$$
\begin{array}{lll}
(0) & L_0 = aL_0 + bL_1 + \varepsilon & \\
(1) & L_1 = bL_0 + aL_2 & \\
(2) & L_2 = bL_0 + aL_2 + \varepsilon & \\
 & & \\
 & L_2 = L_1 + \varepsilon & (1)\ et\ (2) \\
 & L_1 = bL_0 + a(L_1 + \varepsilon) & \\
 & \quad = aL_1 + bL_0 + a & \\
(1') & L_1 = a^*(bL_0 + a) & (Arden) \\
 & L_0 = aL_0 + ba^*(bL_0 + a) + \varepsilon & (0)\ et\ (1') \\
 & \quad = (a + ba^*b)L_0 + ba^*a + \varepsilon & \\
 & L_0 = (a + ba^*b)^*(ba^*a + \varepsilon) & (Arden)
\end{array}
$$

Résiduels

Langage résiduel : définition

Pour tout langage $L \subseteq X^*$ et tout mot $u \in X^*$ on appelle **langage résiduel de L par u** ou encore **quotient à gauche** de L par $u$le langage

$$u^{-1}L = \{v \in X^* |\ u.v \in L\}$$

Ce sont donc tous les mots qui, en leur ajoutant le préfixe u forment un mot de L

Exemple : $L = \{a, abc, abaa, bca, ab\}$

L	$(ab)^{-1}L$
a	
$\underline{ab}c$	c
$\underline{ab}aa$	aa
bca	
$\underline{ab}$	ε

L	$b^{-1}L$
a	
abc	
$abaa$	
$\underline{b}ca$	ca
ab	

$(ab)^{-1}L = \{c, aa, \varepsilon\}$
$b^{-1}L = \{ca\}$

$a^{-1}L = \{\varepsilon, bc, baa, b\}$
$(bb)^{-1}L = \emptyset$

Résiduels

Propriétés « constructives » des résiduels

$\forall u \in X^*, \forall L \subseteq X^*$,

$$\varepsilon^{-1}L = L \quad (1)$$
$$u \neq \varepsilon \Rightarrow u^{-1}\{\varepsilon\} = \emptyset \quad (2)$$
$$(x.u)^{-1}L = u^{-1}(x^{-1}L) \quad (3)$$
$$u^{-1}(L_1 \cup L_2) = u^{-1}L_1 \cup u^{-1}L_2 \quad (4)$$

De plus, pour le quotients à gauche par une lettre :

$$\varepsilon \notin L_1 \Rightarrow x^{-1}(L_1.L_2) = (x^{-1}L_1).L_2 \quad (5)$$
$$\varepsilon \in L_1 \Rightarrow x^{-1}(L_1.L_2) = (x^{-1}L_1).L_2 \cup (x^{-1})L_2 \quad (6)$$
$$x^{-1}L^* = (x^{-1}L).L^* \quad (7)$$

Quotient à gauche par une lettre et expressions régulières

Si x est une lettre

- $x^{-1}(e_1 + e_2) = x^{-1}(e_1) + x^{-1}(e_2)$
- $x^{-1}(e_1).(e_2) = (x^{-1}(e_1)).e_2$ quand $\varepsilon \notin \mathcal{L}(e_1)$
- $x^{-1}(e_1).(e_2) = (x^{-1}(e_1)).(e_2) + x^{-1}(e_2)$ quand $\varepsilon \in \mathcal{L}(e_1)$
- $x^{-1}(e)^* = (x^{-1}(e)).e^*$
- $x^{-1}x = \varepsilon$
- $x^{-1}y = \emptyset$ si y est une lettre, $y \neq x$
- $x^{-1}\varepsilon = \emptyset$
- $x^{-1}\emptyset = \emptyset$

Quotient à gauche par une lettre et expressions régulières

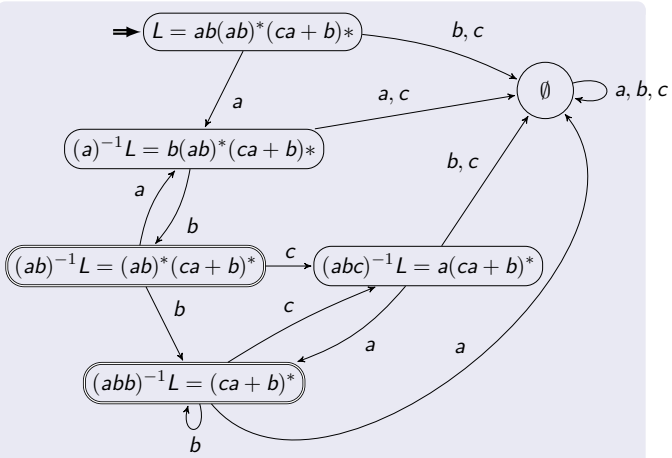
$e = (ac)^* + a^*b + c(a + b)$

$$
\begin{array}{ll}
x^{-1}(e) & = x^{-1}(ac)^* + x^{-1}a^*b + x^{-1}c(a + b) \\
 & = (x^{-1}ac)(ac)^* + (x^{-1}a^*)b + x^{-1}b + (x^{-1}c)(a + b) \\
 & = (x^{-1}ac)(ac)^* + (x^{-1}a)a^*b + x^{-1}b + (x^{-1}c)(a + b) \\
 & \\
a^{-1}(e) & = (a^{-1}ac)(ac)^* + (a^{-1}a)a^*b + a^{-1}b + (a^{-1}c)(a + b) \\
a^{-1}(e) & = c(ac)^* + a^*b + \emptyset + \emptyset \\
 & \\
b^{-1}(e) & = (b^{-1}ac)(ac)^* + (b^{-1}a)a^*b + b^{-1}b + (b^{-1}c)(a + b) \\
b^{-1}(e) & = \emptyset + \emptyset + \varepsilon + \emptyset \\
b^{-1}(e) & = \varepsilon \\
 & \\
c^{-1}(e) & = (c^{-1}ac)(ac)^* + (c^{-1}a)a^*b + c^{-1}b + (c^{-1}c)(a + b) \\
c^{-1}(e) & = \emptyset + \emptyset + \emptyset + \varepsilon(a + b) \\
c^{-1}(e) & = a + b
\end{array}
$$

Exemple de calcul des résiduels

$$
\begin{aligned}
L &= ab(ab)^*(ca+b)* \\
a^{-1}L &= b(ab)^*(ca+b)* \\
(aa)^{-1}L &= \emptyset \\
(ab)^{-1}L &= (ab)^*(ca+b)^* \\
(ac)^{-1}L &= \emptyset \\
(aba)^{-1}L &= b(ab)^*(ca+b)^* && = (a)^{-1}L \\
(abb)^{-1}L &= (ca+b)^* \\
(abc)^{-1}L &= a(ca+b)^* \\
(abba)^{-1}L &= \emptyset \\
(abbb)^{-1}L &= (ca+b)^* && = (abb)^{-1}L \\
(abbc)^{-1}L &= a(ca+b)^* && = (abc)^{-1}L \\
(abca)^{-1}L &= (ca+b)^* && = (abb)^{-1}L \\
(abcb)^{-1}L &= \emptyset
\end{aligned}
$$

Automate des résiduels



Automate des résiduels

Automate des résiduels

Soit un langage M possédant un nombre fini de résiduels, on définit son **automate des résiduels** (automate déterministe)

- $\mathcal{Q} = \{q_R | R \text{ est un résiduel de } M\}$
- $q_{ini} = q_M$
- $\mathcal{F} = \{q_R,\ \varepsilon \in R\}$
- $\forall q_R \in \mathcal{Q},\ \forall x \in X,\ \delta(q_R, x) = q_{x^{-1}R}$

Propriété

L'automate des résiduels de M reconnaît le langage M.

Propriété

Tout langage possédant un nombre fini de résiduels est reconnaissable.

Résiduels et automates

Propriété

Soit un automate **déterministe** A,
pour tout état $q \in Q$ et tout mot $u \in X^*$ tels que $q_{ini} \xrightarrow[*]{u} q$

$$L_q = u^{-1}\mathcal{L}(A)$$

Propriété

Pour un automate **déterministe et accessible** reconnaissant L, chaque langage L_q est un résiduel de L.

Propriété

Dans un automate **déterministe complet** reconnaissant L, pour tout résiduel R de L, il existe au moins un état q tel que $L_q = R$.

Propriété

Tout automate **déterministe complet** possède une nombre d'états **au moins égal au nombre de résiduels du langage reconnu**.

Résiduels et reconnaissables

Propriété

Tout langage reconnaissable admet un nombre fini de résiduels

Propriété

Un langage est reconnaissable si et seulement si il admet un nombre fini de résiduels

Propriété

Tout langage reconnaissable admet un automate déterministe complet minimal unique.
Cet automate est son automate des résiduels.

Langages non réguliers

Un langage non reconnaissable

Le langage $L = \{a^n b^n | n \geq 0\}$ ne possède pas un nombre fini de résiduels. Il n'est donc pas reconnaissable.

pour $k \geq 0$, $(a^k)^{-1}L = \{a^n b^{n+k} | n \geq 0\}$
b^k est le seul mot de b^* appartenant à $(a^k)^{-1}L$
$\Rightarrow$ pour $k' \neq k$, $b^k \notin (a^{k'})^{-1}L$
$\Rightarrow (a^k)^{-1}L \neq (a^{k'})^{-1}L$
$\Rightarrow \{(a^k)^{-1}L\}$ forme un ensemble non fini
$\Rightarrow L$ possède une infinité de résiduels

Théorême

Il existe des langages non réguliers

Minimalisation

Congruence de Nérode

Définie sur les états d'un automate déterministe et accessible par

$$p \cong q \iff L_p = L_q$$

C'est bien une congruence

- compatible avec δ : $\forall x \in X, p \cong q \iff \delta(p,x) \cong \delta(q,x)$
- sature $\mathcal{F}$: $p \cong q \Rightarrow (p \in \mathcal{F} \iff q \in \mathcal{F})$

Nombre de classes d'équivalence

Autant de classes d'équivalence que d'ensembles L_q distincts, donc autant que de langages résiduels

Conclusion

L'automate quotient est l'automate minimal

Minimalisation, méthode de Moore

Algorithme de Moore : calcul de la congruence de nérode

Calcul récurrent

1. $$p \cong_0 q \iff (p \in \mathcal{F} \iff q \in \mathcal{F})$$
   niveau 0 : tous deux acceptants ou tous deux non acceptants.
2. $$p \cong_{i+1} q \iff p \cong_i q \text{ et } \forall x \in X, \delta(p,x) \cong_i \delta(q,x)$$
   niveau $i+1$: équivalence au niveau i
   ET
   pour toute lettre x les états obtenus depuis ces 2 états doivent être équivalents au niveau i.

- le calcul est itéré jusqu'à n tel que $\cong_n = \cong_{n-1}$

Minimalisation, méthode de Moore

Partition de l'ensemble d'états par raffinements successifs

Les états sont répartis en classes qui sont raffinées à chaque étape.
On note $[q]_i$ la classe de q à l'étape i.
On note $x_1, x_2, ...x_m$ les lettres de l'alphabet.

- au départ ($i = 0$), 2 classes : $\{\mathcal{F}, Q - \mathcal{F}\}$
- passer de l'étape i à $i+1$:
  Pour **chaque classe**, établir le « profil » de chaque état :

  $$profil_i(q) = ([\delta(q,x_1)]_i\ ,\ [\delta(q,x_2)]_i\ ,\ \ldots\ ,\ [\delta(q,x_m)]_i)$$

  si plusieurs profils distincts apparaissent, la classe est scindée : une partie par profil, regroupant les états de même profil.
- fin quand aucune partie n'a été scindée lors d'une étape.
- les classes obtenues sont les classes de l'équivalence de Nérode
- l'automate minimal est l'automate « quotient »

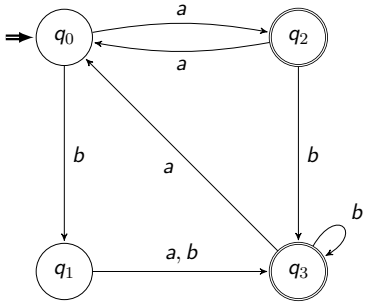
Minimalisation, méthode de Moore

Automate minimal

- $\mathcal{Q}^{min} = \{\ [q]\ , q \in Q\}$
- $q_{ini}{}^{min} = [q_{ini}]$
- $\mathcal{F}^{min} = \{\ [q]\ , [q] \subseteq \mathcal{F}\}$
- $\delta([q], x) = [\ \delta(q,x)\ ]$

Exemple de minimalisation (algo de Moore)

Automate initial
(déterministe complet et accessible)



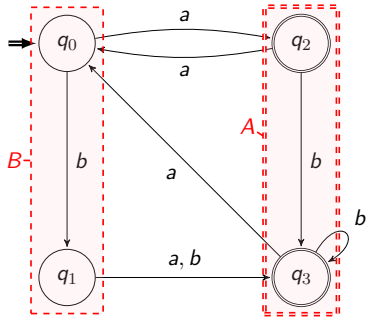
Étape 0

Créer 2 classes A et B

- une pour les états acceptants
- une pour tous les autres

Exemple de minimalisation (algo de Moore)

Étape 0 (résultat)
$A = \{q_2, q_3\} \subseteq \mathcal{F},\ B = \{q_0, q_1\}$



Étape 1

A	a	b
q_2	B	A
q_3	B	A

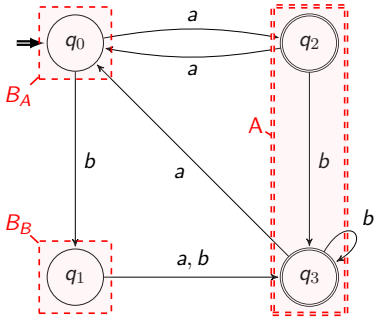
q_2 et q_3 ont même profil.
A n'est pas scindée.

B	a	b
q_0	A	B
q_1	A	A

q_0 et q_1 ont des profils distincts (donc sont **non** équivalents).
On scinde B en 2 classes :
$B_A = \{q_0\},\ B_B = \{q_1\}$

Exemple de minimalisation (algo de Moore)

Étape 1 (résultat)
$A = \{q_2, q_3\} \subseteq \mathcal{F}$,
$B_A = \{q_0\},\ B_B = \{q_1\}$



Étape 2

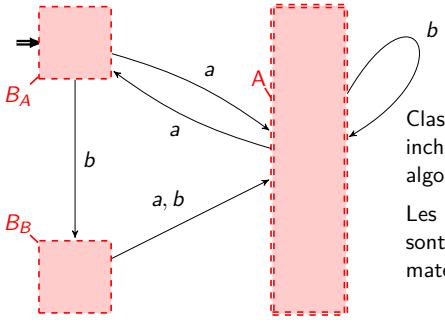
A	a	b
q_2	B_A	A
q_3	B_A	A

q_2 et q_3 sont équivalents.
A n'est pas scindée.

Les classes B_A et B_B ne peuvent pas être scindées (singletons).

Exemple de minimalisation (algo de Moore)

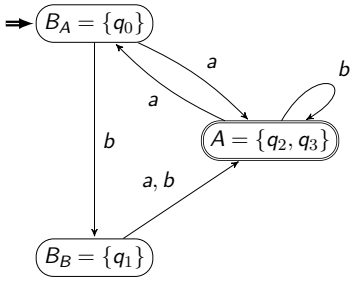
Étape 2 (résultat)
$A = \{q_2, q_3\} \subseteq \mathcal{F}$,
$B_A = \{q_0\},\ B_B = \{q_1\}$



Classes d'équivalence inchangées, donc algorithme terminé.

Les classes obtenues sont les états de l'automate minimal.

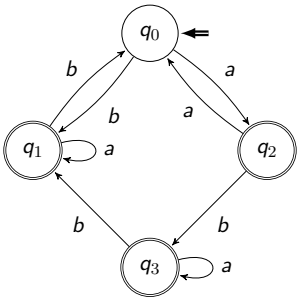
Exemple de minimalisation (algo de Moore)



Automate déterministe complet minimal

Exemple de minimalisation (algo de Moore)

Automate initial
(déterministe complet et accessible)



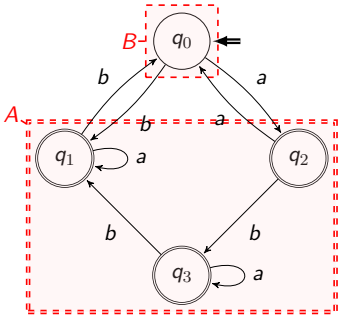
Étape 0

Créer 2 classes A et B

- une pour les états acceptants
- une pour tous les autres

Exemple de minimalisation (algo de Moore)

Étape 0 (résultat)
$A = \{q_1, q_2, q_3\} \subseteq \mathcal{F},\ B = \{q_0\}$



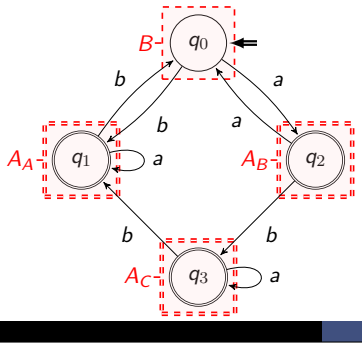
Étape 1

A	a	b
q_1	A	B
q_2	B	A
q_3	A	A

Les états présentent 3 profils distincts :
$(A, B),\ (B, A),\ (A, A)$
A est scindée en 3.

Exemple de minimalisation (algo de Moore)

Étape 1 (résultat)
$A_A = \{q_1\},\ A_B = \{q_2\},\ A_C = \{q_3\}$
$A_A \subseteq \mathcal{F},\ A_B \subseteq \mathcal{F},\ A_C \subseteq \mathcal{F}$
$B = \{q_0\}$



Étape 2

Plus aucun raffinement n'est possible..

NB : on est revenu à l'automate initial, qui était donc minimal