


# Langage naturel problèmes et solutions informatiques



Béatrice Daille

Université de Nantes

inspiré de J. Jurasky, L. Danlos et d'autres  
collègues

# 2001, L'Odyssée de l'espace

L'ordinateur HAL comprend l'homme, dialogue avec lui dans sa langue, exécute ses commandes, et ressent des émotions.



# 2001, L'Odyssée de l'espace

L'ordinateur HAL comprend l'homme, dialogue avec lui dans sa langue, exécute ses commandes et ressent des émotions.

**A 12 ans de cette échéance  
en est-on là ?**



# Applications du TALN

- ❑ Les voitures parlent
- ❑ Les machines à café comprennent votre commande
- ❑ Dictée automatique
- ❑ Correcteurs orthographiques et grammaticaux dans les traitements de textes
- ❑ Systèmes de traduction automatique
- ❑ Photocopieurs ou téléphones qui traduisent



# Applications du TALN

- Moteurs de recherche du Web
- Assistante personnelle virtuelle
- Scanner (OCR)
- Saisie de textes sur des claviers de taille réduite
- Routage de documents
- Analyse des opinions



# Plan de l'exposé

1. ***Les bases***
2. Application bluffante : ELIZA (1966)
3. ***Recherche de motifs***
  - Expressions rationnelles***
  - Automates à états finis***



# Les bases

- Caractères et codes
- Chaînes de caractères et mots
- Phrases
- Textes, tris, loi de Zipf

# Caractère (au sens informatique)

Pour l'ordinateur, chaque caractère correspond à un code, généralement un nombre entier, même les caractères invisibles.

Un code : une table de correspondance qui associe à une donnée numérique un symbole graphique (*glyphes*)

1	2	3	4	5	...	23	24	25	26
A	B	C	D	E	...	W	X	Y	Z





# Critères des codes

**Nom** : un code est toujours nommé

**Taille** : Indication du nombre de bits nécessaires pour coder les symboles

**Symboles** (alphabet latin) :

- les 10 chiffres,
- les 26 lettres de l'alphabet
- les signes de ponctuation ou des opérateurs
- les *caractères de contrôle*

**Traitement** :

tri des caractères



# Codes normalisés (1)

**ASCII** – Norme ISO 646 en 1963, puis en 1988

- 7 bits (128 caractères)
- universel : inclus dans les autres codes utilisés
- Codage :
  - 0 à 31 : caractères de contrôle
  - 32 à 47 : signes de ponctuation et opérateurs
  - 48 à 57 : chiffres 0 à 9
  - 65 à 90 : lettres majuscules (sans accent)
  - 97 à 122 : lettres minuscules (sans accent)
- Textes de la langue anglaise uniquement

# Code ASCII – Table

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Caractère : C

Nom : c majuscule

Décimal : 67

Hexadécimal : 43

Octal : 0103

Binaire : 010000011



# Codes normalisés (2)

## **ISO Latin n (1987- )**

- 8 bits (256 caractères)
- 128 premiers caractères : caractères ASCII
- 10 groupes : 128 autres caractères
- ISO-8859-1 ou ISO-LATIN-1 ou LATIN-1
- Langues de l'Europe de l'Ouest : allemand, anglais, danois, espagnol, féroïen, finnois, français, islandais, italien, néerlandais, norvégien, portugais, suédois.



# Codes normalisés (3)

## UNICODE 1993

<http://www.unicode.org>

- consortium Unicode (1989) : ISO + constructeurs ordinateurs + ... (Apple, IBM, Microsoft, etc.)
- 10 principes (universalité, efficacité, ...)
- 16 bits (65 536 caractères)

## ISO/IEC 10646

- 32 bits (4 milliards de caractères)
- UTF8 : Unicode sur 1 à 4 octets



# UTF8

Implémentation de Unicode 5.0 (2006) : codage de taille variable

- 1 octet (8 bits) : caractères ASCII
- 2 octets (16 bits) : ISO-8859 (Latin1) + caractères alphabétiques
- 3 octets (24 bits) : Chinois, Japonais, Coréen
- 4 octets : tous les autres caractères

# UTF8 – Exemple

Codage des caractères d'Unicode sous forme de séquences de un à quatre octets

Character	UTF-16	UTF-8	UCS-2
A	0041	41	0041
c	0063	63	0063
Ö	00F6	C3 B6	00F6
☹	4E9C	E4 BA 9C	4E9C
§	D834 DD1E	F0 9D 84 9E	N/A



# Commandes LINUX

- **Détection**

```
echo -n é | wc -c
```

```
file -i fichier.txt
```

```
utrac -p fichier.txt
```

- **Conversion**

```
iconv -f iso-8859-1 -t utf-8 entree.txt -o  
sortie.txt
```

```
iconv -f utf-8 -t iso-8859-1 -o entree.txt  
sortie.txt
```





# Chaîne de caractères / Mot (au sens informatique)

- Une **chaîne de caractères** est une **séquence** de caractères
- Un **séparateur de mot** est un caractère particulier de l'alphabet qui permet de délimiter le mot
- Un **mot** est une séquence de lettres comprise entre 2 séparateurs de mots consécutifs



# Phrase (au sens informatique)

- Un **délimiteur de phrase** est un caractère particulier de l'alphabet qui permet de délimiter la phrase
- Une **phrase** est une séquence de caractères comprise entre 2 séparateurs de phrases consécutifs



# Texte et Tris

- Occurrence : l'apparition d'une chaîne de caractères dans un texte
- Fréquence : nombre d'occurrences
- Hapax : fréquence 1
- Tris
  - liste de fréquence : listes des occurrences accompagnées de leur fréquence
  - Autres listes : liste alphabétique, liste de première apparition

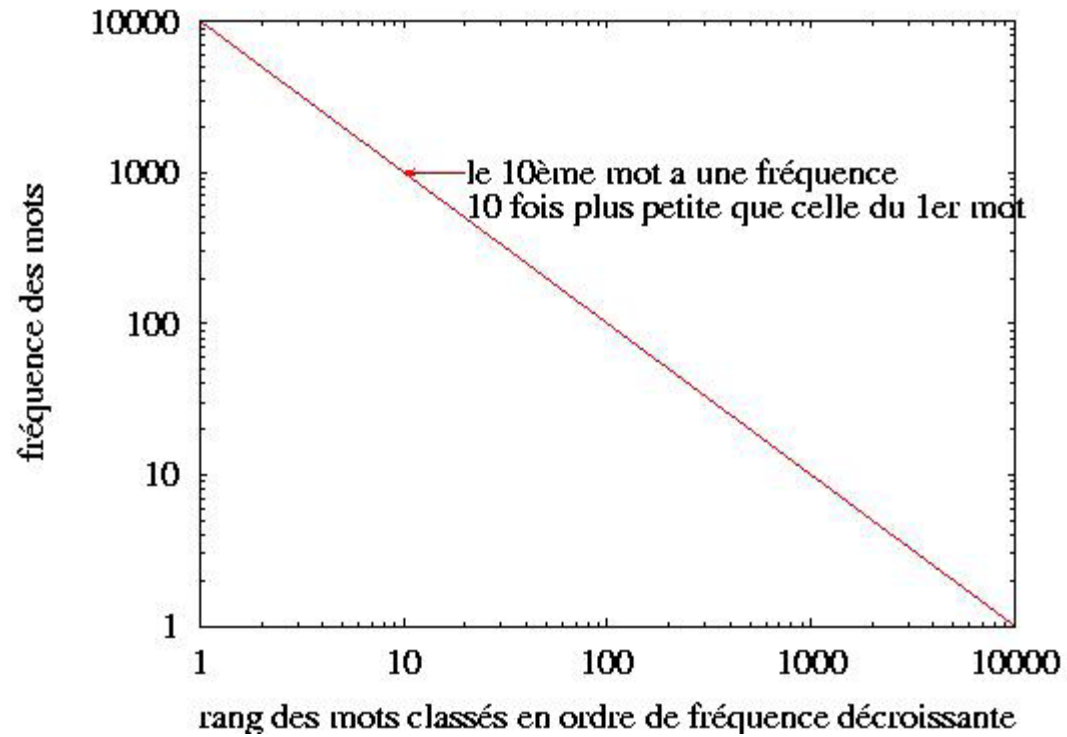


# Loi de Zipf

Dans les années 30, un scientifique de l'université de Harvard, **G.K. Zipf**, a montré qu'en classant les mots d'un texte par fréquence décroissante, on observe que la fréquence d'utilisation d'un mot est inversement proportionnelle à son rang,  $r$ . Cette loi peut s'exprimer de la manière suivante :

Fréquence d'un mot de rang  $r = (\text{Fréquence du mot de rang } 1) / r$

# Loi de Zipf



# Liste de fréquence des mots

## Le Cid

Rang 1	429	<i>de</i>	2	264	<i>l'</i>	3	259	<i>?</i>	
	4	<u>258</u>	<i>et</i>	5	245	<i>un</i>	6	230	<i>en</i>
	7	229	<i>le</i>	8	220	<i>que</i>	9	201	<i>mon</i>
	10	198	<i>est</i>	<u>11</u>	<u>191</u>	<i>Et</i>	12	189	<i>d'</i>
	13	187	<i>je</i>	14	177	<i>la</i>	15	167	<i>il ....</i>
	16	155	<i>vous</i>	17	151	<i>qu'</i>	<b>40</b>	<b>67</b>	<b>Rodrigue</b>

## Le journal Le Monde (10 ans)

Rang 1	16 702 498	<i>de</i>
Rang 2	9 239 401	<i>la</i>
Rang 3	7 048 266	<i>le</i>



# Plan de l'exposé

1. ***Les bases***
2. Application bluffante : ELIZA (1966)
3. ***Recherche de motifs***
  - Expressions rationnelles***
  - Automates à états finis***



# Eliza

(J. Weizenbaum, MIT, 1966)

Simulation de dialogue écrit entre  
un psychothérapeute et un patient

Le psychothérapeute est un ordinateur





# Eliza : exemples

- Bonjour, je suis votre psychothérapeute. Parlez moi de vos problèmes.
- Je suis très angoissé.
- Pourquoi dites-vous que vous êtes très angoissé ?
- C'est à cause de ma femme.
- Parlez-moi de votre famille.



# Eliza : exemples

- J'adore mes enfants.
- J'aimerais que vous me parliez de vos relations avec vos enfants.
- Personne ne m'aime.
- Est-ce que ça vous ennuie que personne ne vous aime ?



# Eliza : exemples

- Oui, je me sens seul.
- Votre vie sexuelle y est peut-être pour quelque chose. Continuez sans crainte.
- ...

# Eliza : Analyse de l'exemple

- Bonjour, je suis votre psychothérapeute.  
Parlez moi de vos problèmes.
  - Texte figé qui commence tout dialogue
- Je suis très angoissé.
  - Réponse libre du patient
  - Stockée dans une variable R1
- Pourquoi dites-vous que vous êtes très angoissé ?
  - Pourquoi dites-vous que R1 ?
  - transformation : *je suis* → *vous êtes*



# Eliza : Analyse de l'exemple

- C'est à cause de ma femme.
  - Réponse libre du patient qui contient le mot « femme »
- Parlez-moi de votre famille.
  - Ordre systématique du psy dès que la réponse précédente contient un mot de la liste : *femme, mari, mère, père, etc.*

# Eliza : Analyse de l'exemple

- **Oui, je me sens seul.**
  - Réponse libre du patient qui ne contient rien de spécial
- **Votre vie sexuelle y est peut-être pour quelque chose. Continuez sans crainte.**
  - Réponse du psy quand il ne sait plus trop quoi dire



# Bilan sur Eliza

- Application bluffante
- L'ordinateur ne comprend **RIEN** aux interventions du patient.
- Ses réponses : activation d'une des centaines ou milliers de réponses pré-enregistrées  
par la technique de recherche de motifs



# Recherche de motifs

- Expressions régulières/rationnelles
- Automates à états finis





# Expressions régulières

- Un langage formel pour décrire les chaînes de caractères
- Recherche/Remplacement de chaînes de caractères
- Notions
  - Caractère littéral/Méta-caractères
  - Classes de caractères
  - Quantificateurs
  - Ancres
  - Alternative (ou)

# Expressions régulières

- Délimiteur d'expression régulière `/ /`
- Syntaxe Perl (compatible JavaScript)
- Classe de caractères `[ ]`

Regexp	Correspondance	Formes
<code>/[vV]alise/</code>	Valise ou valise	« <u>Valise</u> »
<code>/[abc]/</code>	a ou b ou c	« <u>valise</u> »
<code>/[1234567890]/</code>	N'importe quel chiffre	« le <u>1</u> <sup>er</sup> janvier »

# Expressions régulières

□ Classe de caractères

expression d'un intervalle [ - ]

Regexp	Correspondance	Formes
/[a-z]/	une lettre minuscule	« l <u>a</u> valise »
/[A-Z]/	une lettre majuscule	« la <u>V</u> alise »
/[0-9]/	un chiffre	« le <u>1</u> <sup>er</sup> janvier »

# Expressions régulières

## □ Négateur [ ^ ]

Regexp	Correspondance	Formes
/[^A-Z]/	pas une lettre majuscule	« L <u>a</u> valise »
/[^Ss]/	ni S ni s	« S <u>a</u> lut »
/[^\.]/	pas un point	« <u>n</u> otre ami »
/[e^]/	soit e soit ^	« grand <u>^</u> »
/a^b/	la chaîne a^b	« ah <u>a^b</u> aah »

# Expressions régulières

- Quantificateurs

?            0 ou 1 fois

/voi?le/        ⇒    vole ou voile

\*            0 ou n fois

/oo\*h/            ⇒ oh ou ooh ou ooooh ou oooooh ...

+            1 ou n fois

/oo+h/            ⇒ ooh ou ooooh ou oooooh

# Expressions régulières

- Quantificateurs

$\{m,n\}$  au minimum  $m$  occurrences et au maximum  $n$  occurrences

$/me\{2,3\}uh/ \Rightarrow$  meeuh **ou** meeeuh

$\{m,\}$  au minimum  $m$  occurrences

$/me\{2,\}uh/ \Rightarrow$  meeuh **ou** meeeuh **ou** meeeeeuh

$\{m\}$  exactement  $m$  occurrences

$/me\{2\}uh/ \Rightarrow$  meeuh

# Expressions régulières

- N'importe quel caractère sauf fin de ligne `.`

`/b./` ⇒ `bol` ou `bel` ou `b8l` ou ...

`/a.c./` ⇒ `a1c1` ou `abcd` ou `aacc` ou ...

- Ancres `^` `$`

- `/^[A-Z]/` → "Ramallah, Palestine"

- `/^[^A-Z]/` → "¿verdad?" "vraiment ?"

- `/\.$/` → "La fin."

- `/.$/` → ?

- alternative `|`

- `/vous|moi/` → "c'est soit vous soit moi"

→ "c'est soit moi soit vous"

- `/(lu|ma|me|je|ve|sa|di)/`

# Exemple

- Trouver toutes les occurrences du mot “les”
  - `/les/`  
Pas de prise en compte des majuscules
  - `/[lL]es/`  
Trouve aussi lessive, faiblesse
  - `/^[lL]es$/`
  - `/[^a-zA-Z][lL]es[^a-zA-Z]/`
  - `/(^|[^a-zA-Z])[lL]es[^a-zA-Z]/`





# Erreurs

- 2 types d'erreurs
  - Chaînes trouvées que l'on n'aurait pas dû trouver (**lessive**, **lest**, faibl**esse**)
    - Faux positifs
  - Chaînes non trouvées que l'on aurait dû trouver
    - Faux négatifs



# Réduire les erreurs

- Quelle que soit l'application, réduire son taux d'erreur prend en compte deux aspects antagonistes
  - Améliorer l'exactitude (réduire les faux positifs)
  - Améliorer la couverture (réduire les faux négatifs).

# Remplacement et mémorisation

remplacement

`s/feuille/feuille/`  
`s/feuille/feuille/g`  
`s/feuille/feuille/i`

Remplacement de  
toutes les occurrences

Remplacement  
quelque soit la casse

Mémorisation à l'aide des variables : \1, \2, etc.

`s/une petite (.+)/une  
\1tte/`

une petite fille → une fillette



# Eliza : exemples

- Je suis très angoissé.
- Pourquoi dites-vous que vous êtes très angoissé ?
- C'est à cause de ma femme.
- Parlez-moi de votre famille.

# Eliza : expressions régulières

Étape 1 : remplacer les pronoms à la 1<sup>ère</sup> personne du singulier par les pronoms à la 2<sup>ème</sup> personne du pluriel

```
s/\bje suis\b/vous êtes/g
```

```
s/\bma\b /votre/g
```

Étape 2: utiliser des regexp pour générer les réponses

```
s/.* (vous êtes (très)? (angoissé|triste)) .*/Pourquoi  
dites-vous que \1/
```

Étape 3: utilisation de probabilités pour choisir parmi plusieurs remplacements possibles



# Bilan sur les expressions régulières

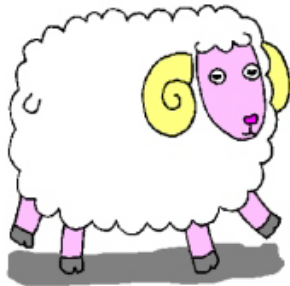
- Les expressions régulières constituent un outil des plus utiles pour manipuler du texte
- Eliza : vous pouvez déjà faire beaucoup en n'utilisant que des expressions régulières



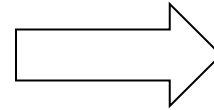
# Automates à états finis

- Les expressions régulières sont une des manières d'exprimer un automate à états finis
- Les automates à états finis sont au cœur de nombreux algorithmes du traitement automatique du langage naturel

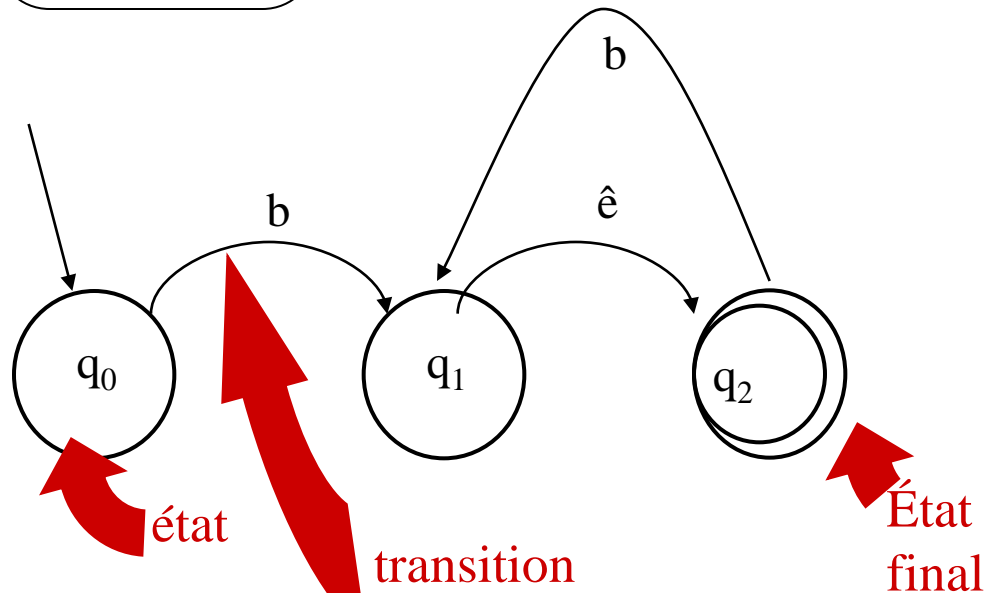
# Automates à états finis



bê  
bêbê  
bêbêbê  
...



$/\wedge(b\hat{e})+\$/$



Langage naturel





# Automate du mouton

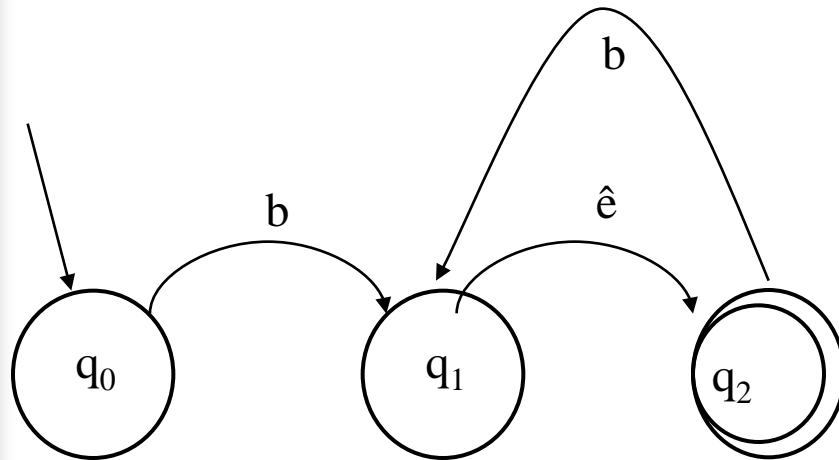
- caractéristiques
  - 3 états
  - b et ê appartiennent à l'alphabet
  - $q_0$  est l'état initial
  - $q_2$  est l'état final
  - 3 transitions



# Définir un automate à états finis

- Un ensemble d'états :  $Q$
- Un alphabet fini :  $\Sigma$
- Un état initial  $q_0$
- Un ensemble  $F$  d'états finaux  $F \subseteq Q$
- Une fonction de transition  $\delta(q, i)$  de  $Q \times \Sigma$  vers  $Q$

# Une autre formulation



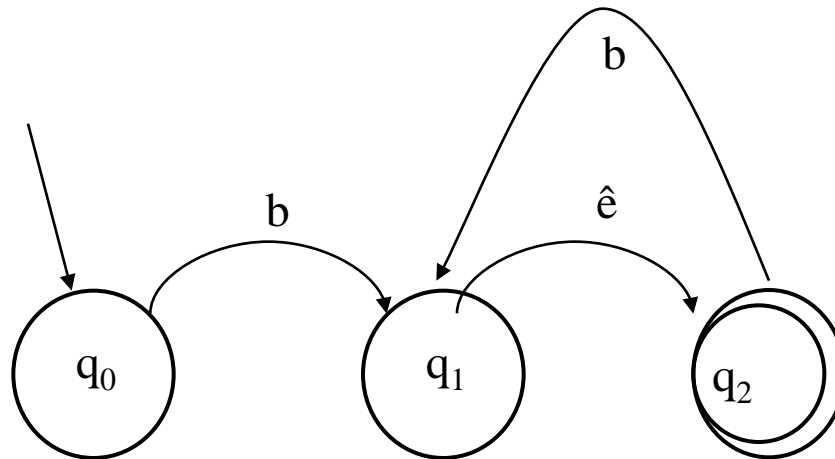
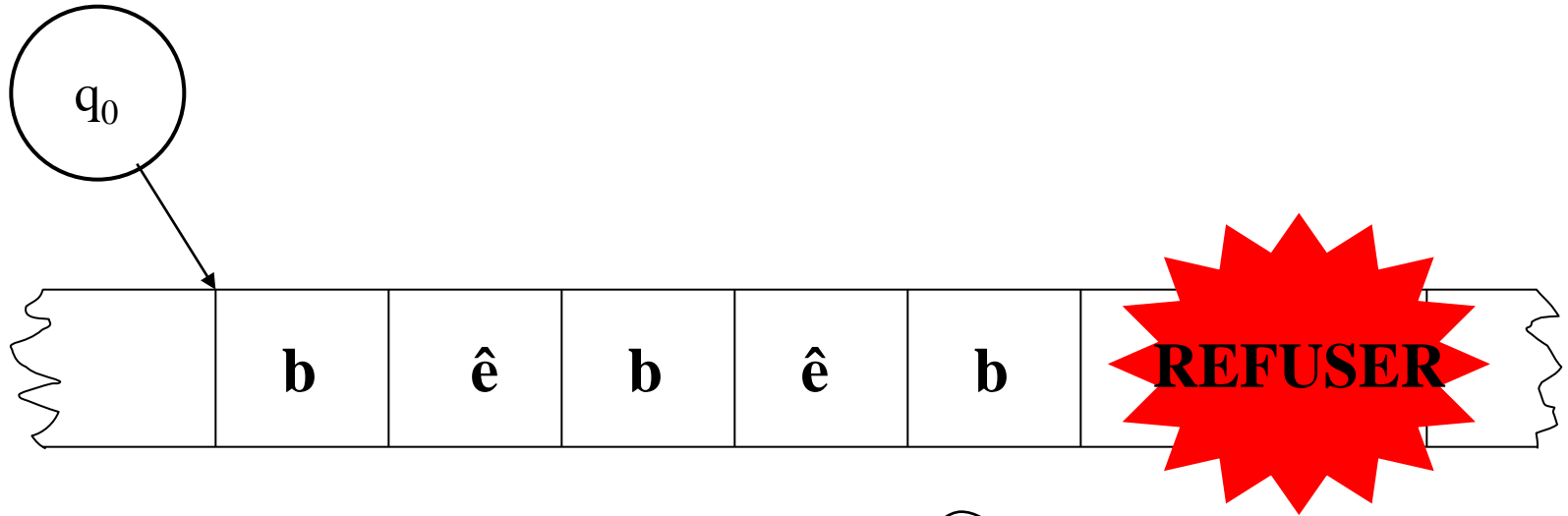
États	<b>b</b>	<b><math>\hat{e}</math></b>
$q_0$	$q_1$	0
$q_1$	0	$q_2$
$q_2$	$q_1$	0



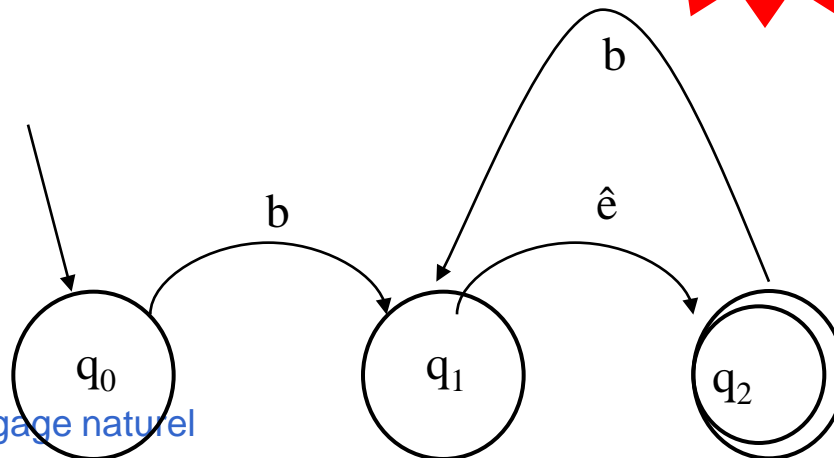
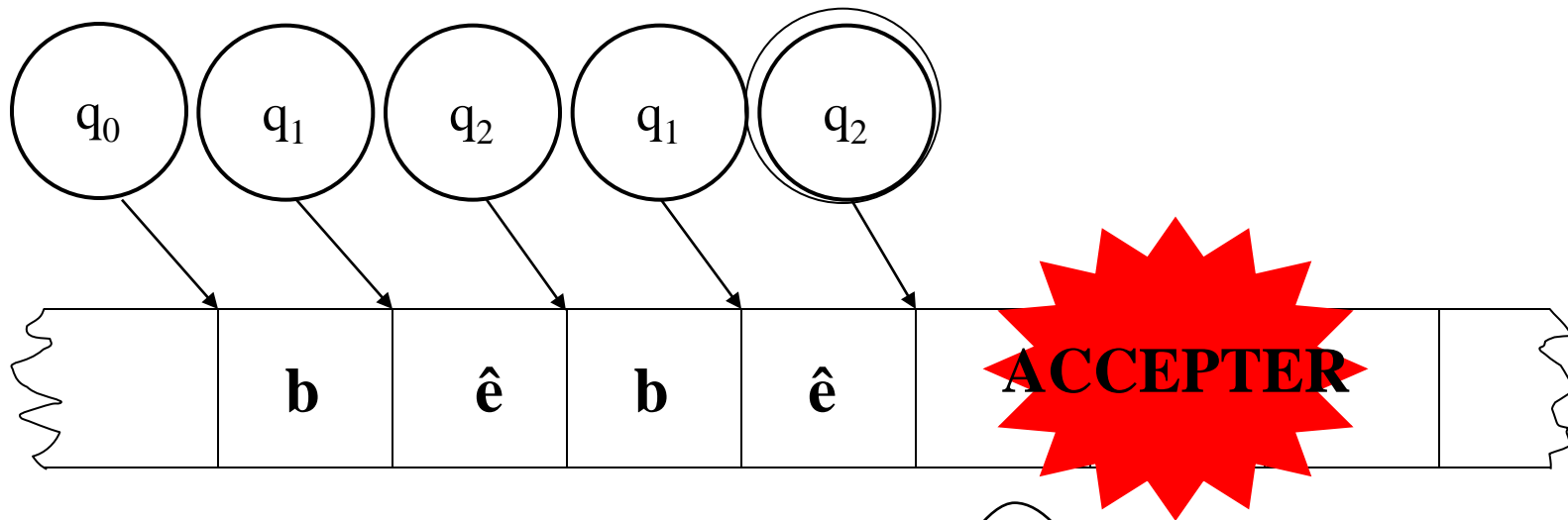
# Reconnaissance

- Une chaîne est reconnue si elle est acceptée par l'automate
  - Commencer à l'état initial
  - Examiner la chaîne en entrée
  - Consulter la table des transitions
  - Aller à l'état suivant et avancer le curseur sur la chaîne
  - Jusqu'à la fin de la chaîne

# Chaîne d'entrée



# Chaîne d'entrée



# Algorithme de reconnaissance

```
fonction Reconnaître (chaîne, automate) retourner accepter or  
refuser  
  index ← début de chaîne  
  état_courant ← état initial  
  répéter  
    Si fin de chaîne alors  
      si état_courant est un état final alors  
        retourner accepter  
      sinon  
        retourner refuser  
    sinon si transition [état_courant, chaîne[index]] == 0 alors  
      retourner refuser  
    sinon  
      état_courant ← transition [état_courant, chaîne[index]]  
      index ← index + 1  
  fin
```