IFT-7022
Traitement de la langue naturelle

Expressions régulières

Tiré de Jurafsky et Martin (2e et 3e édition)

Chapitre 2 – *Speech and language processing*

Expressions régulières

- Utiles pour la recherche dans un texte
- On décrit ce que l'on cherche à l'aide d'un patron (*pattern*)
  - Patron: une représentation compacte d'un ensemble de chaînes de caractères.
  - Recherche: une fonction qui retourne les textes qui correspondent au patron.
- Disponibles dans la plupart des langages
  - Perl, grep, Java, Python, Emacs, etc..
- Fréquemment utilisé en NLP

Exemple

- Trouver toutes les instances des mots *le*, *la et l'* dans le texte.
  - La première dame américaine Melania Trump s'est exprimée lundi matin sur la lutte contre le harcèlement sur l'internet, l'une de ses priorités, son mari débutant lui sa journée par une salve habituelle de tweets acerbes. «Reconnaissons-le: la plupart des enfants ont davantage conscience des avantages et des pièges des réseaux sociaux que certains adultes», a déclaré Melania Trump à l'occasion d'un colloque sur le cyberharcèlement dans le Maryland.

Outil en ligne : http://gskinner.com/RegExr/

Exemple

- Trouver toutes les instances des mots *le*, *la et l'* dans le texte.
  - La première dame américaine Melania Trump s'est exprimée lundi matin sur la lutte contre le harcèlement sur l'internet, l'une de ses priorités, son mari débutant lui sa journée par une salve habituelle de tweets acerbes. «Reconnaissons-le: la plupart des enfants ont davantage conscience des avantages et des pièges des réseaux sociaux que certains adultes», a déclaré Melania Trump à l'occasion d'un colloque sur le cyberharcèlement dans le Maryland.
- Expressions :
  - `/\b[lL][ae']\b/`

Outil en ligne : http://gskinner.com/RegExr/

Expressions régulières

- Littéral
- Classe de caractères
- Registre de valeur (*range*)
- Négation
- Optionnel
- N'importe quel caractère
- Alias
- Compte

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/Claire_says,/	"Dagmar, my gift please," Claire says,"
/song/	"all our pretty songs"
/!/	"You've left the burglar behind again!" said Nori

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

Figure 2.1 The use of the brackets [] to specify a disjunction of characters.

RE	Match	Example Patterns Matched
/[A-Z]/	an uppercase letter	"we should call it 'Drenched Blossoms'"
/[a-z]/	a lowercase letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Figure 2.2 The use of the brackets [] plus the dash - to specify a range.

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	"Oyfn pripetchik"
[^Ss]	neither 'S' nor 's'	"I have no exquisite reason for't"
[^\.]	not a period	"our resident Djinn"
[e^]	either 'e' or '^'	"look up ^ now"
a^b	the pattern 'a^b'	"look up a^b now"

Figure 2.3 Uses of the caret ^ for negation or just to mean ^ .

Expressions régulières

- Littéral
- Classe de caractères
- Registre de valeur (*range*)
- Négation
- Optionnel
- N'importe quel caractère
- Alias
- Compte

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"woodchuck"
`colou?r`	color or colour	"colour"

Figure 2.4 The question-mark ? marks optionality of the previous expression.

RE	Match	Example Patterns
`/beg.n/`	any character between *beg* and *n*	begin, beg'n, begun

Figure 2.5 The use of the period . to specify any character.

RE	Expansion	Match	Example Patterns
`\d`	`[0-9]`	any digit	Party␣of␣5
`\D`	`[^0-9]`	any non-digit	Blue␣moon
`\w`	`[a-zA-Z0-9_]`	any alphanumeric or underscore	Daiyu
`\W`	`[^\w]`	a non-alphanumeric	!!!!
`\s`	`[␣\r\t\n\f]`	whitespace (space, tab)	
`\S`	`[^\s]`	Non-whitespace	in␣Concord

Figure 2.6 Aliases for common sets of characters.

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression

Figure 2.7 Regular expression operators for counting.

Erreurs de repérage

- Il y a deux types d'erreurs possibles:
  - Repérer les chaînes qui n'auraient pas dû correspondre au patron
    - par ex. *les*, *habituelle*
    - Faux positifs
  - Ne pas repérer celles qui auraient dû correspondre au patron
    - par ex. *La*
    - Faux négatifs

Erreurs de repérage

- Fréquentes dans les différentes applications langagières.
- Réduire le taux d'erreur revient souvent à:
  - Augmenter la précision (minimiser les faux positifs)
  - Augmenter le rappel (*recall*) (minimiser les faux négatifs).

http://regex.alf.nu

Regex en Python

- *Module re*
- *compile* – fonction qui génère une version compilée d'une regex
- *match/search* – fait le *matching* entre une regex et une chaîne
- *sub* – remplace la première occurrence d'un patron par une chaîne

```
import re

example_string = "this is a test"
p = re.compile(r"\w+")

m = p.search(example_string)
print("Start index:", m.start())    #  Start index: 0
print(" End index:", m.end())   #  End index: 4
print(m.group())  #  this

print(p.findall(example_string))  # ['this', 'is', 'a', 'test']

p = re.compile("is")
print(p.match(example_string))  # None

print(re.sub(r"(this) (is) (.+)", "\\2 \\1 \\3", example_string))  #  'is this a test'
```

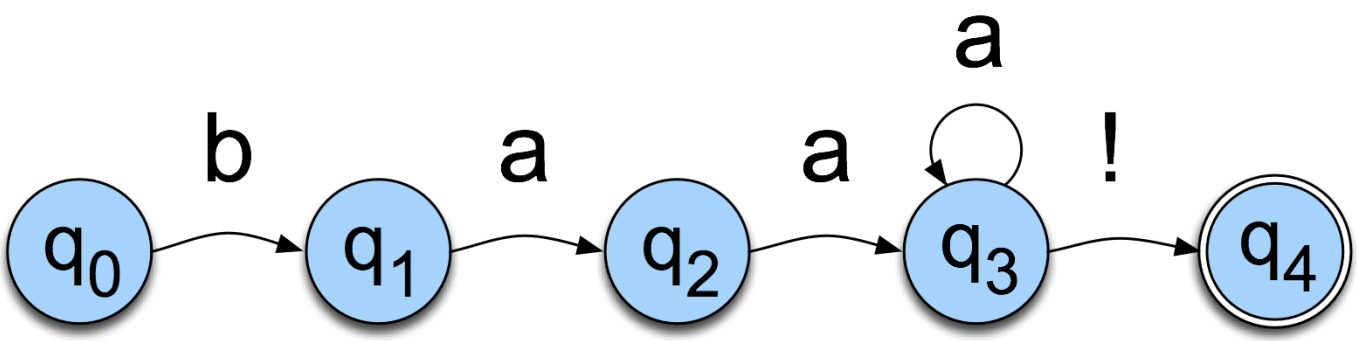
- Nouveau module - regex

Automate à états finis

- Expressions régulières
  - Une notation
  - Pour des raisons d'efficacité, on utilise des versions "compilées"
  - Peuvent être implémentées par un automate à états finis (FSA)
- Automates (déterministes ou probabilistes)
  - Utilisé pour plusieurs techniques NLP.
    - Par ex. la morphologie, une partie de la syntaxe.

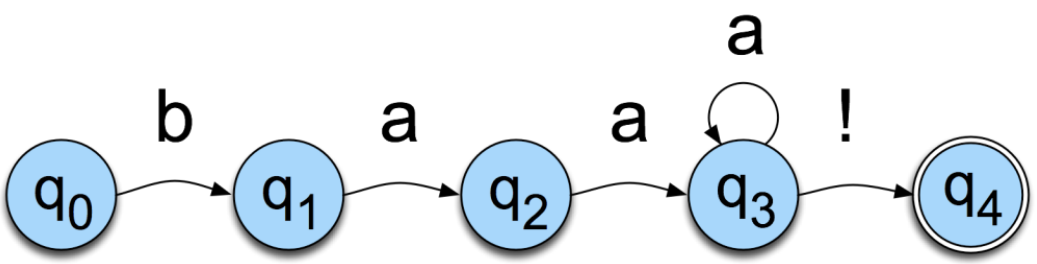
FSAs comme des graphes

- Considérons le langage des moutons du chapitre 2.
  - `/baa+!/`



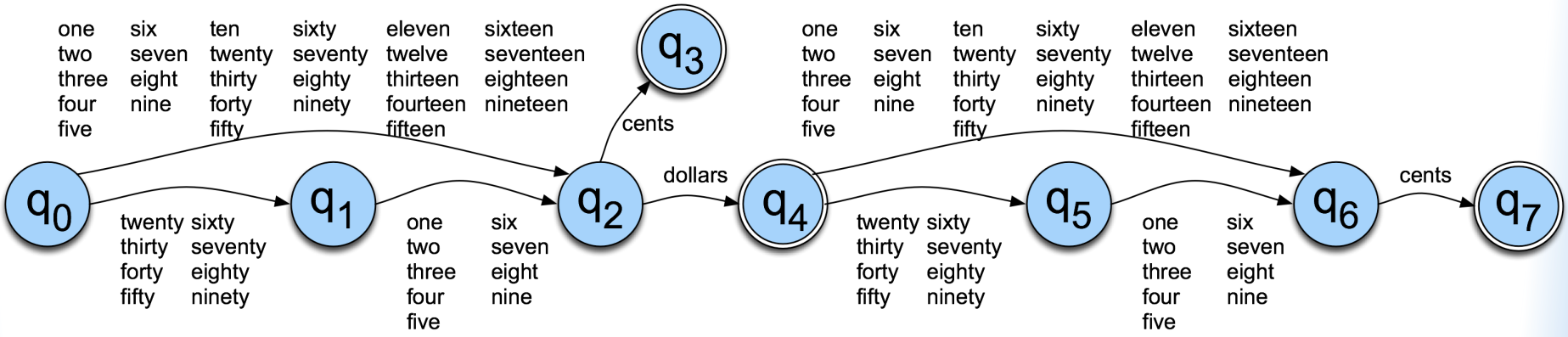
Exemple de FSA

- On peut observer à partir de cet automate:
  - Il a 5 états.
  - b, a, et ! constitue son alphabet.
  - q_0 est l'état de départ.
  - q_4 est un état terminal.
  - Il y a 5 transitions.



À propos des alphabets

- On fait seulement référence à un ensemble fini de symboles en entrée.
- Ces symboles peuvent désigner de plus gros objets.

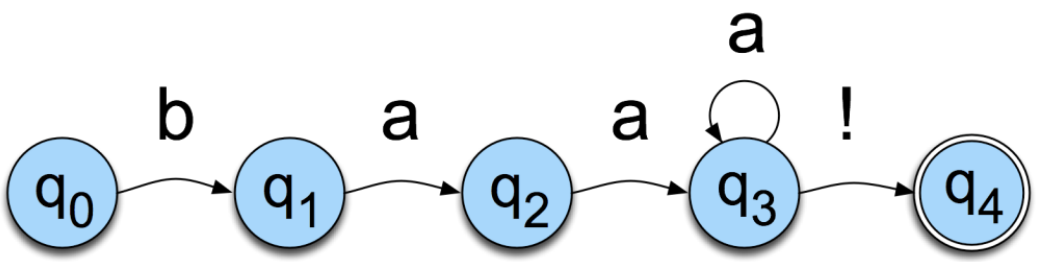


Une autre perspective

- Les FSAs peuvent être représentés par des tableaux.
  - La fonction de transition

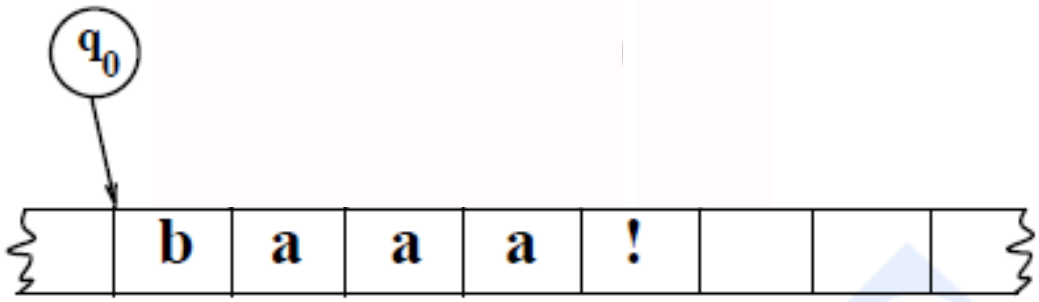
	b	a	!	e
0	1			
1		2		
2		3		
3		3	4	
4:				

Si on est dans l'état 1 et qu'on observe le symbole *a*, alors on va vers l'état 2.



Reconnaissance

- Le processus qui
  - Démarre dans l'état initial;
  - Examine le symbole en entrée;
  - Consulte la table;
  - Sélectionne le nouvel état et met à jour le ruban;
  - Continue jusqu'à ce qu'on soit rendu au bout du ruban.



	b	a	!	e
0	1			
1		2		
2		3		
3		3	4	
4:				

D-Recognize

```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
   elsif transition-table[current-state,tape[index]] is empty then
     return reject
   else
     current-state ← transition-table[current-state,tape[index]]
     index ← index + 1
  end
```

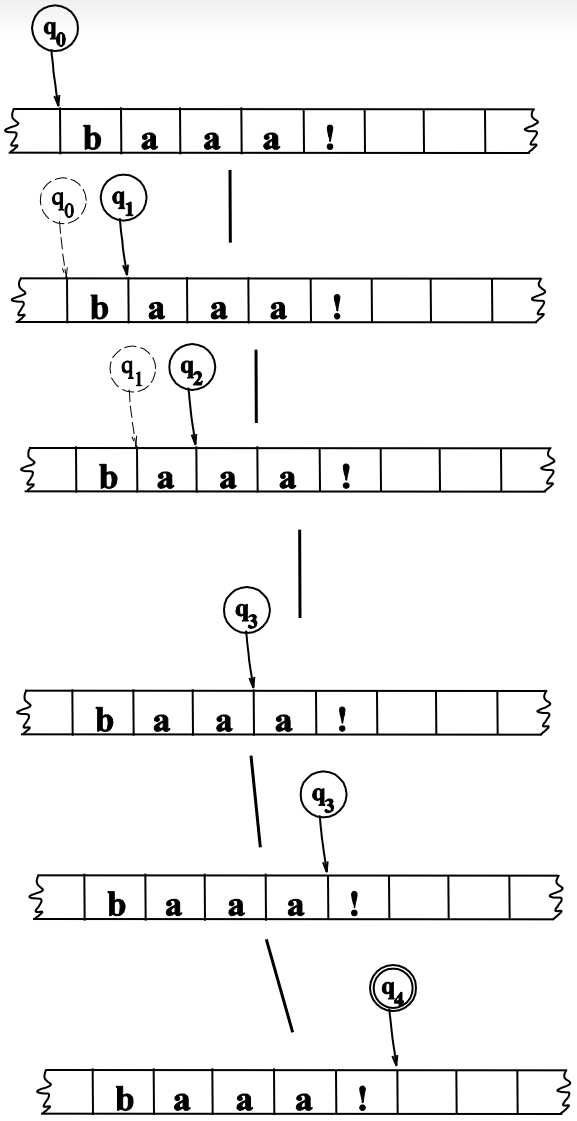
Points importants

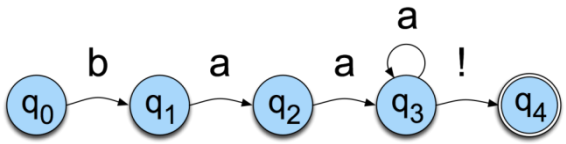
- D-recognize → algorithme déterministe
  - À chaque point de traitement, il n'y qu'une seule transition possible (pas de choix).
  - Tout simplement un interpréteur dirigé par une table de transition.
- Donc le *matching* d'expressions régulières revient à :
  - Traduire l'expression en table de transition;
  - Passer la table et la chaîne de caractères à un interpréteur.

Reconnaissance et exploration

- On peut voir cet algorithme comme une forme triviale d'exploration dans un espace d'états.
  - Les états sont décrits par les positions sur le ruban et le numéro des états.
  - Les opérateurs sont compilés dans une table.
  - L'état but est la combinaison de la fin du ruban et d'un état terminal.
- Cette formulation est triviale parce que…?

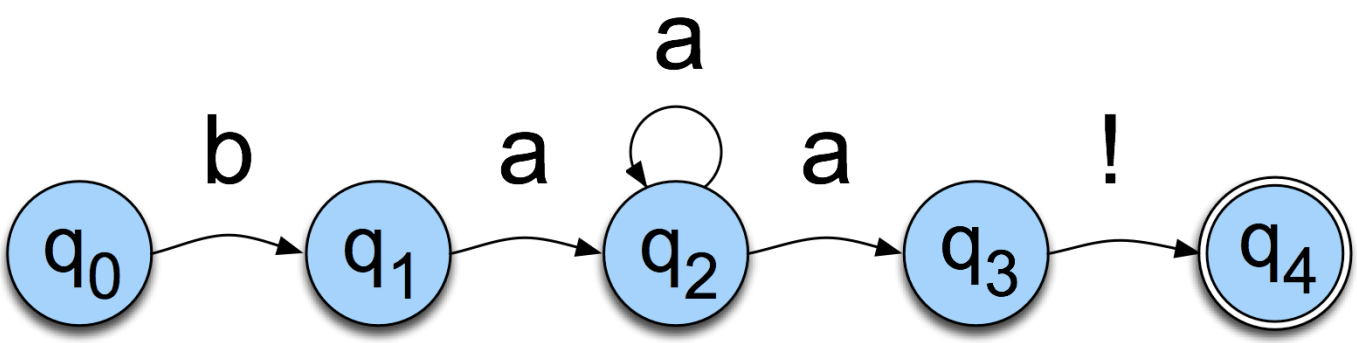
Exemple


q0
b a a a !
q0 q1
b a a a !
q1 q2
b a a a !
q3
b a a a !
q3
b a a a !
q4
b a a a !


a
b a a !
q0 q1 q2 q3 q4

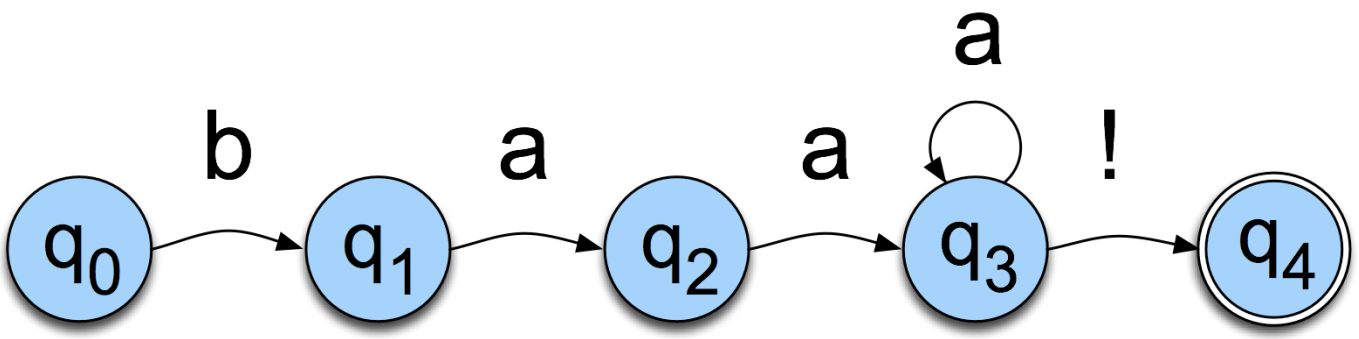
Exemple de FSA

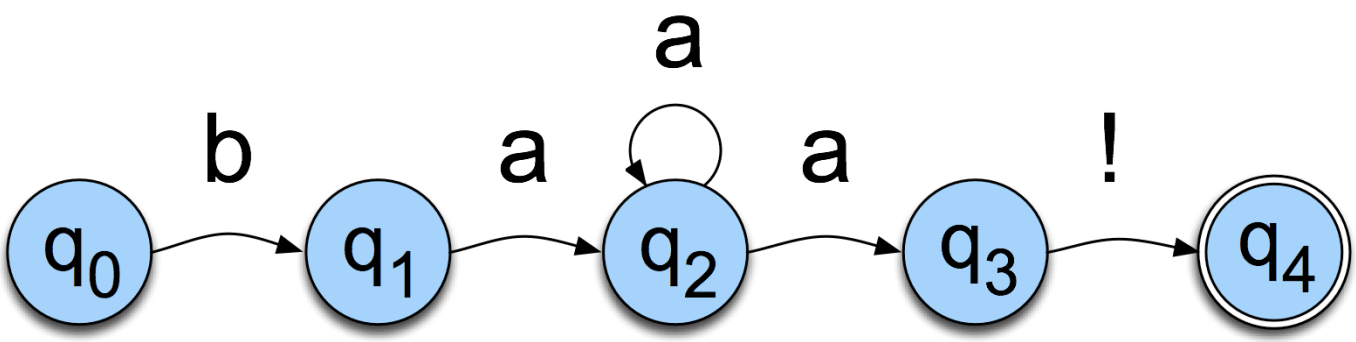
- D'autres automates peuvent correspondent au même langage.
  - `/baa+!/`



Non-Déterminisme

`/baa+!/`



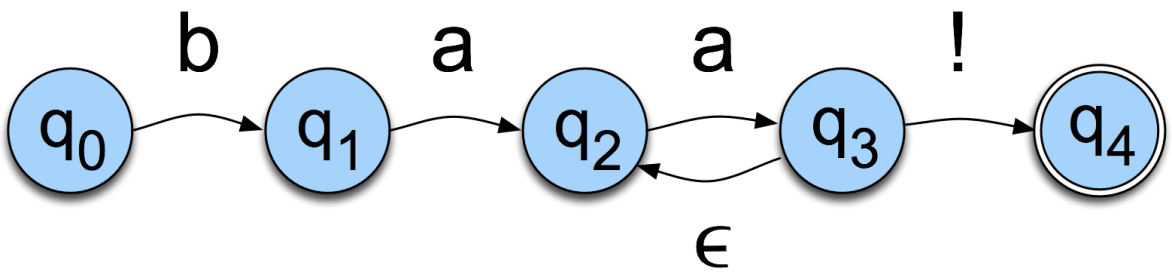


Reconnaissance non-déterministe

- Deux approches de base (voir Friedl 2006)
  1. Convertir un automate non déterministe en version déterministe.
     - Et faire la reconnaissance avec ce nouvel automate.
  2. Gérer les différentes options comme une exploration dans un espace d'états
     - Sans modifier l'automate.

Non-Déterminisme

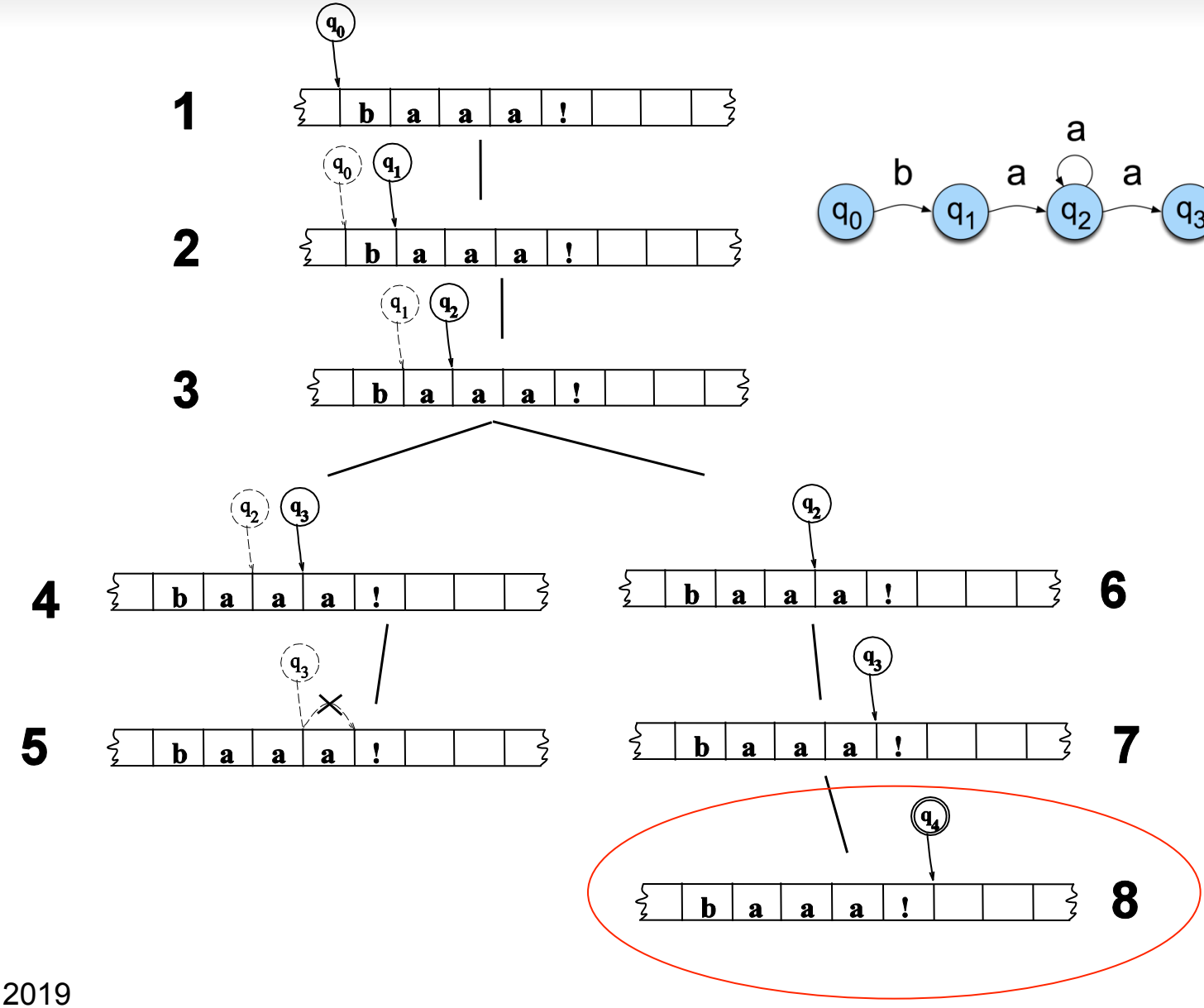
- Possible de faire la conversion
- Une technique - Transitions epsilon
  - Ces transitions n'avancent pas le ruban durant le processus de reconnaissance.



Reconnaissance ND: Exploration

- Dans un automate non déterministe :
  - Il existe au moins un chemin complet pour une chaîne définie par le langage de l'automate.
  - Mais pas tous les chemins d'un automate mène à un état final pour une chaîne.
  - Aucun chemin de l'automate ne mène à un état terminal pour une chaîne qui ne fait pas partie du langage.
- Succès de l'exploration
  - Trouver un chemin qui mène à un état final.
- Échec de l'exploration
  - Aucun chemin possible dans l'arbre.

Exemple



Reconnaisance non-déterministe

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

  agenda ← {(Initial state of machine, beginning of tape)}
  current-search-state ← NEXT(agenda)
  loop
    if ACCEPT-STATE?(current-search-state) returns true then
      return accept
    else
      agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
    if agenda is empty then
      return reject
    else
      current-search-state ← NEXT(agenda)
  end

function GENERATE-NEW-STATES(current-state) returns a set of search-states

  current-node ← the node the current search-state is in
  index ← the point on the tape the current search-state is looking at
  return a list of search states from transition table as follows:
    (transition-table[current-node,ε], index)
    ∪
    (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

  current-node ← the node search-state is in
  index ← the point on the tape search-state is looking at
  if index is at the end of the tape and current-node is an accept state of machine
  then
    return true
  else
    return false
```

Points importants

- Les états de l'espace sont des paires :
  - Position du ruban;
  - État de l'automate.
- En gardant une trace des états non explorés :
  - Un interpréteur peut systématiquement explorer tous les chemins possibles dans l'automate.
- Quel type d'exploration?

Mais pourquoi ?

- Le non-déterminisme ne nous donne pas plus de puissance de traitement.
- Et il est plus coûteux à traiter.
  - Voir *catastrophic backtracking*
- Mais les solutions sont plus faciles à comprendre pour un humain.

Conclusion

- Les expressions régulières sont des outils puissants.
- N'importe quelle expression régulière peut être implémentée par un automate (FSA).
- Les automates peuvent être :
  - Déterministes : définis par l'état courant.
  - Non déterministes : par exploration des différents chemins possibles.
- Toute expression régulière peut être compilée automatiquement :
  - en automate non déterministe;
  - qui peut être converti en automate déterministe.

Pour plus d'information

- Tutoriel Python
  - https://docs.python.org/3/library/re.html
- Livre de Friedl (2006)