AI6122 Text Data Management & Analysis

Topic: Finite-state-automata

Topics to be covered

- Regular expression
- How to implement regular expression?
- Finite-state-automata (FSA)
  - Deterministic FSA
  - Non-deterministic FSA

Regular expression

- Regular expression is a **compact textual representation** of a set of strings representing a language
  - In the simplest case, regular expressions describe regular languages

- Have you tried text search with
  - Programming language: perl, python, C#, java….
  - Linux less, emacs, vi, grep, etc..

REs in programming languages

- Java – Pattern
  - // Extract the text between the two title elements
  - pattern = "(?i)(<title.*?>)(.+?)(</title>)";
  - String updated = EXAMPLE_TEST.replaceAll(pattern, "$2");
  - Reference: https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

- Python – re
  - >>> import re
  - >>> m = re.search('(?<=abc)def', 'abcdef')
  - >>> m.group(0)
  - 'def'

Basic regular expression

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,”
/DOROTHY/	“SURRENDER DOROTHY”
/!/	“You’ve left the burglar behind again!” said Nori

- The simplest RE is a sequence of characters
- / / are NOT part of the RE, but a notation used in Perl language
- The first instance of each match to the RE is underlined
  - an application might return more than just the first one.

A bit more regular expression

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"

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

- Square brackets([ ]) matches **any single character** from within the class
  - [a-zA-Z] matches any letter,
  - [0-9] matches any number;
  - dash (-) specifies a range

- \b matches a **word boundary**
  - /\bdog\b/ The doggie plays in the yard. The dog is cute

A bit more

- Dot (.): matches **any single character** (\. will match the dot only)
  - /3.14159/ vs. /3\.14159/ matches 3.14159 only
- Star (*): matches **zero or more** of preceding item
  - /fred*/ matches fre, fred, fredddd.
- Plus(+): matches **one or more** of preceding item
  - /fred+/ matches fred, fredddd but not fre
- Question mark (?) matches **zero or one** of preceding item

Always match the largest string they can
e.g. /fred*/
String: <u>freddd</u>fff

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"<u>woodchuck</u>"
`colou?r`	color or colour	"<u>colour</u>"

Applications of RE

- Recognize phone numbers
  - e.g. 6790 6300

- Syntax highlighting
  - e.g. html tags, programming language syntax

- Data validation
  - e.g. input for postal code

- Morphological analysis (next chapter)
- And many more …

Example: When RE is used in real applications

- Find all the instances of the word “the” in a text.
  - /the/ How about: The?
  - /[tT]he/ How about: They they other?
  - /\b[tT]he\b/ How about: the_ the25

- The process we just went through was to fix two kinds of errors
  - Matching strings that we <u>should not</u> have matched (there, then, other)
    - False positives
  - <u>Not</u> matching things that we <u>should have</u> matched (The)
    - False negatives

Exercise

- Task is to match "the"
  - 5 words: "the they theu The teo"
  - A solution: `/the/`
  - If the matches are "the they theu The teo"

- What are the false positives?
- What are the false negatives?

Errors

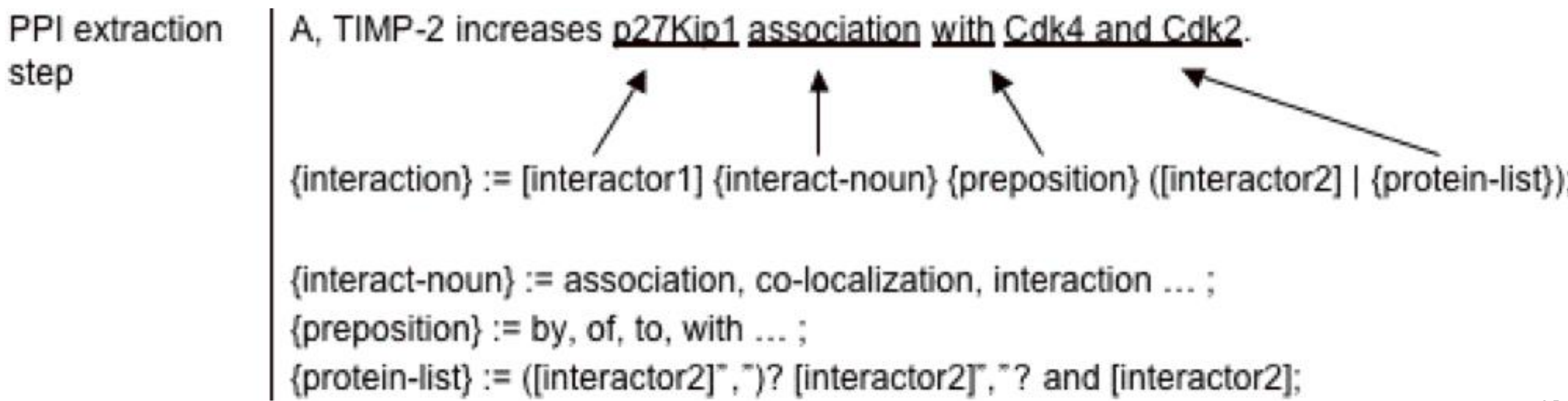
- Reducing the error rate for an application often involves two efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).

  - Example: “They The the they”
    - `/the/` (They The the they)
    - `/[tT]he/` (They The the they)

- We’ll be telling the same story for many tasks, in the whole semester

RE application examples

- XML tags
  - Pattern.compile("<MeshHeadingList>.*</MeshHeadingList>", Pattern.DOTALL)
  - Pattern.compile("<DescriptorName[^<>]*>[^<>]*</DescriptorName>")
- Time (e.g. 09:30): ^[0-2]?[0-3]:[0-5][0-9]$



Finite state automata (FSA)

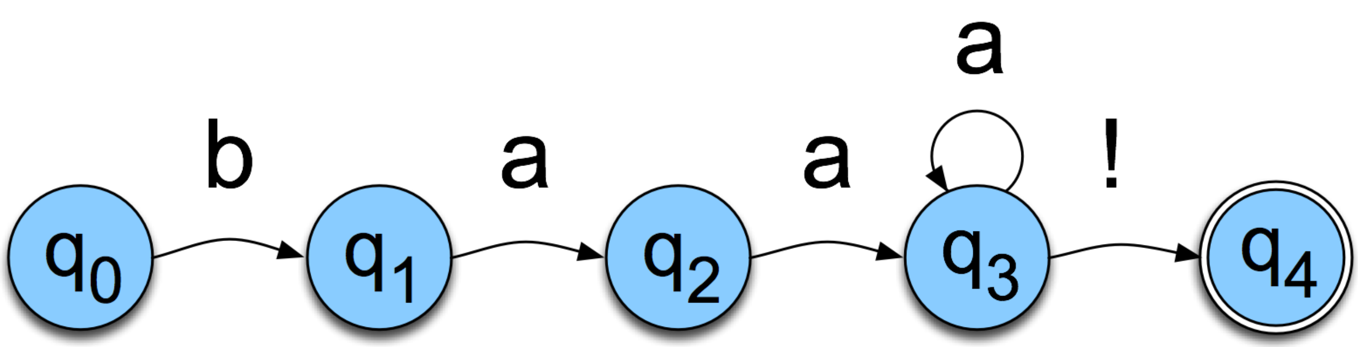
- Regular expressions
  - Compact textual strings (e.g. “/[tT]he/”)
    - Perfect for specifying patterns in programs or command-lines
  - Can be implemented as a FSA

- Finite state automata
  - Graphs
  - Can be described with a regular expression
    - A textual way of specifying the structure of FSA
  - FSA has a wide range of uses

FSA as Graphs

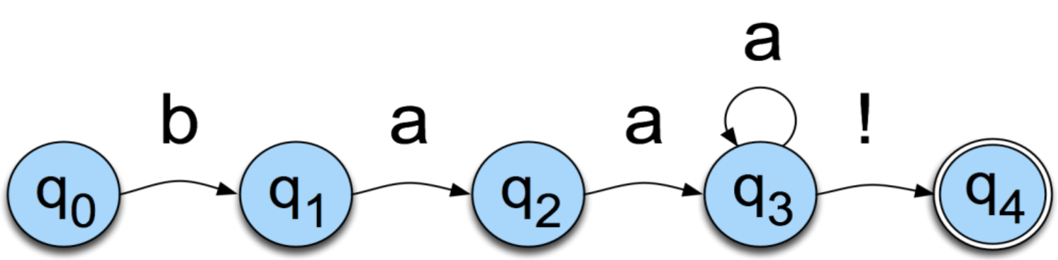
- Let's start with the sheep language from the text
  - /baa+!/

baa! baaa!
baaaa! …



Sheep language FSA

- We can say the following things about this machine
  - It has 5 states
  - **b**, **a**, and **!** are in its alphabet
  - q_0 is the start state
  - q_4 is an accept state
  - It has 5 transitions



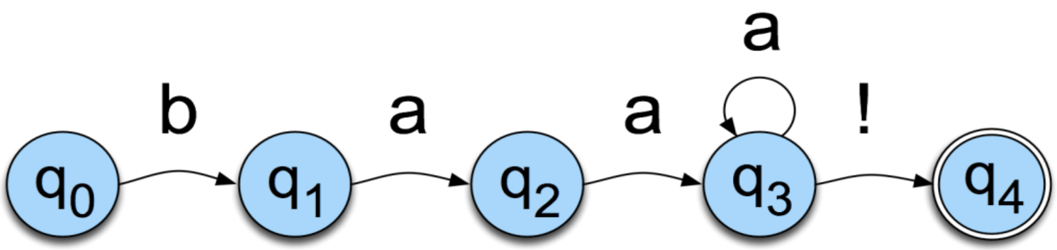
More Formally

- You can specify an FSA by enumerating the following things:
  - The set of states: Q
  - A finite alphabet: Σ
  - A start state
  - A set of accept/final states
  - A transition function that maps $Q \times \Sigma \rightarrow Q$

- Don't take term **alphabet** word too narrowly;
  - it just means we need **a finite set of symbols** in the input.

Yet Another View

- An FSA can ultimately be represented as tables



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

If you're in state 1 and you're looking at an input ***a***, then go to state 2

Recognition

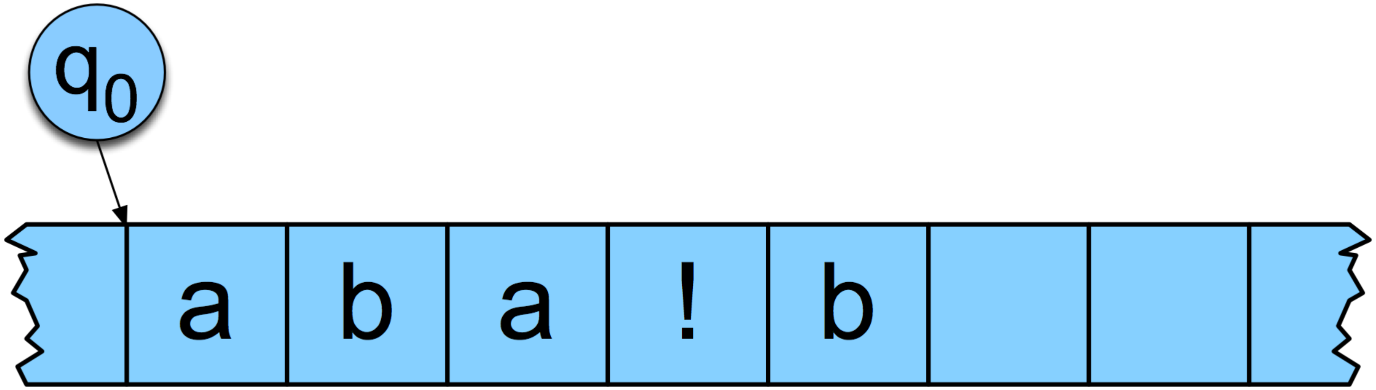
- Recognition is the process of **determining if a string should be accepted** by a machine

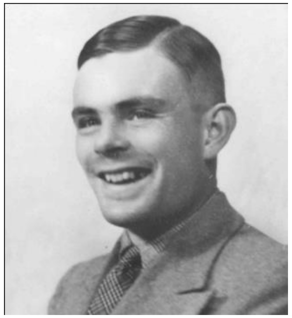
- Or… it's the process of determining if a string is in the language we're defining with the machine

- Or… it's the process of **determining if a regular expression matches a string**

Recognition

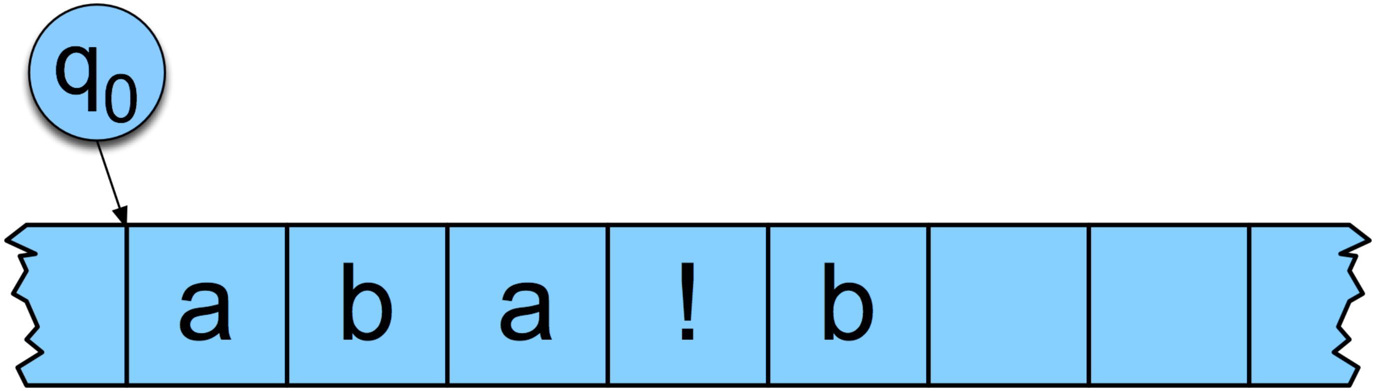
- Traditionally, (Turing's notion) this process is depicted with a tape.





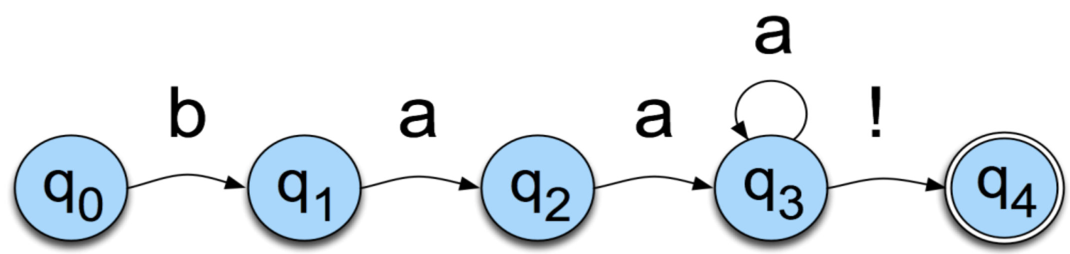
Recognition (D-Recognize)

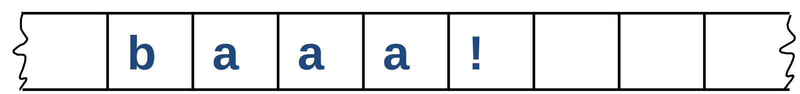
- Simply a process of starting in the **start state**
- Examining the **current input**
- Consulting **the table**
- Going **to a new state** and updating the tape pointer.
- Until you run **out of tape**.
- Accept?

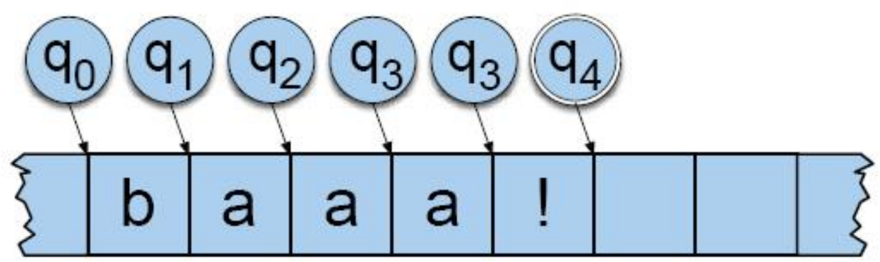


Example

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







D-Recognize algorithm

```
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
```

Key Points

- Deterministic means that at each point in processing there is always one **unique** thing to do (no choices).

- D-recognize is a simple table-driven interpreter
- The algorithm is universal for all **unambiguous** regular languages.
  - To change the machine, you simply change the table.

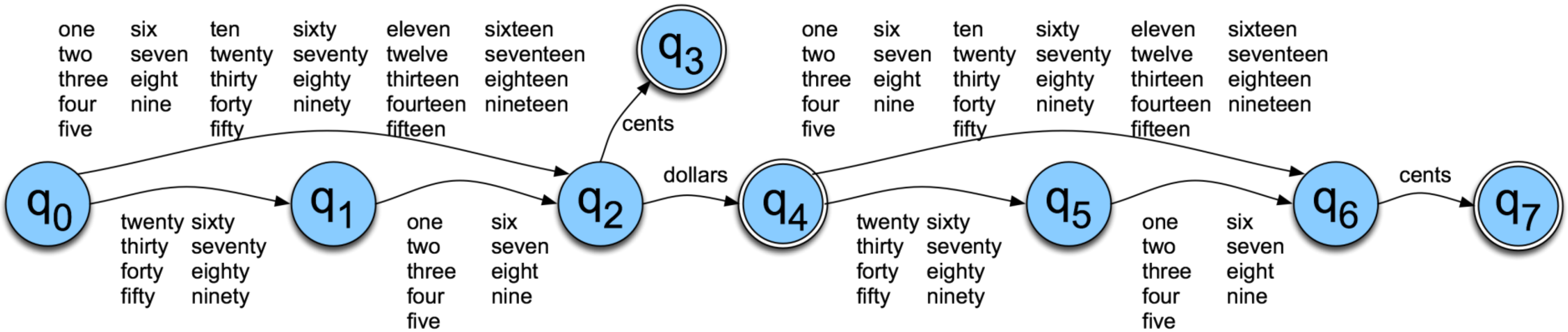
- Matching strings with regular expressions (e.g., Perl, grep, etc.) is a matter of
  - translating the regular expression into a machine (a table) and
  - passing the table and the string to an interpreter that implement D-recognize

A short summary

- Regular expression
  - Very basic one /the/
  - More notations [], ?, *, .
- Two types of errors
  - false positives
  - false negatives
- Finite state automata
  - Representation
  - D-recognize algorithm to implement regular expression

Dollars and Cents

- Don't take term ***alphabet*** too narrowly; it just means we need a finite set of symbols in the input.

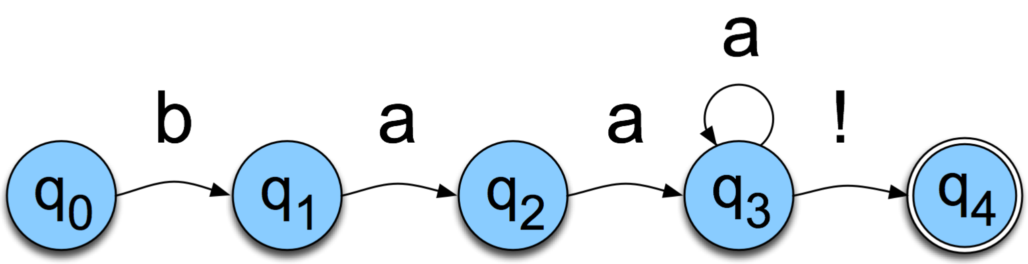


- Exercise: How to represent the expressions for phone numbers in Singapore with FSA/RE?

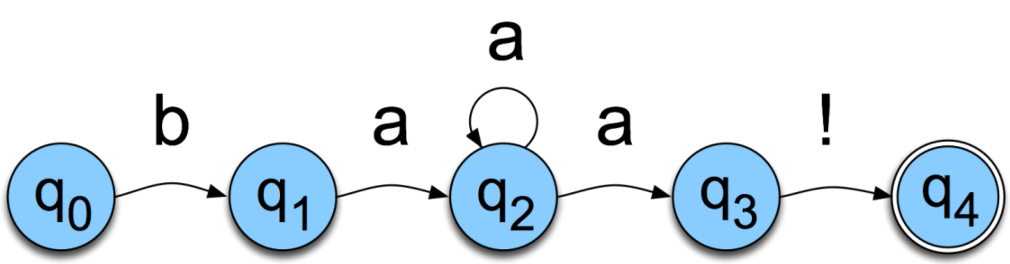
Non-Deterministic (NFSA)

- The sheep language from the text: **/baa+!/**

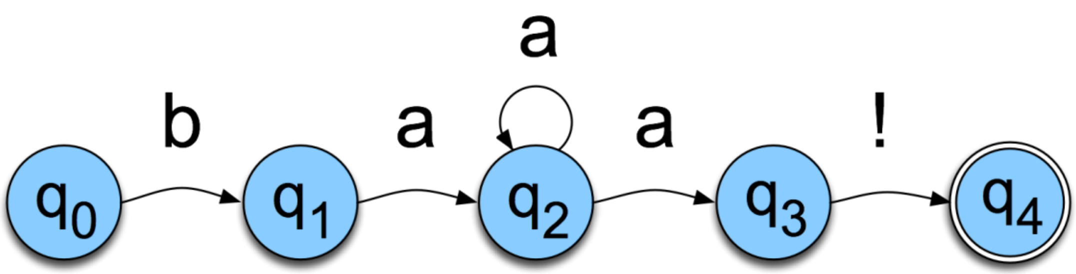
baa!   baaa!   baaaa! …



- There are other machines that correspond to this same language



Yet Another View

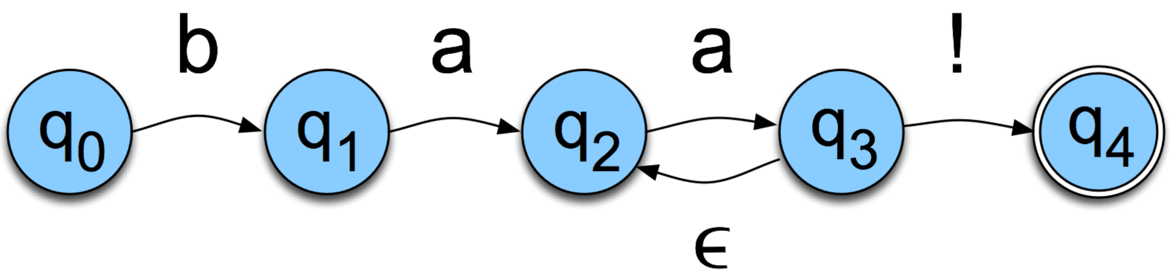


	Input			
State	b	a	!	e
0	1			
1		2		
2		2, 3		
3			4	
4:				

If you're in state 2 and you're looking at an input ***a***, then go to state 2 or 3

Non-Deterministic (NFSA)

- Yet another technique
  - **Epsilon** transitions
  - These transitions do not examine or advance the tape during recognition



Equivalence

- Non-deterministic machines **can be converted to deterministic** ones with a fairly simple construction

- That means that they have the same power;
  - non-deterministic machines **are not more powerful than** deterministic ones in terms of the languages they can and can't characterize

- Non-determinism doesn't get us more formal power and it causes headaches so why bother?
  - More natural (understandable) solutions
  - Not always obvious to users whether or not the regex that they've produced is non-deterministic or not,
    - better to not make them worry about it

Non-Deterministic Recognition: Search

- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But **not all paths** directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

Non-Deterministic Recognition

- Success in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.

- Failure occurs when **all** of the possible paths for a given string lead to failure.

A summary

- Regular expression
  - – Very basic regular expression /the/
  - – More notations [], ?, *, .
- Two types of errors: false positive, false negative
- Finite state automata
  - – Deterministic (FSA)
    - D-recognize algorithm to implement regular expression
  - – Non-Deterministic (NFSA)
    - Two approaches to implementing regular expression

Readings

- Java.util.regex API
  - http://docs.oracle.com/javase/8/docs/api/java/util/regex/package-summary.html

- Java regexps tutorial
  - http://docs.oracle.com/javase/tutorial/essential/regex/

- RegExr: an online tool to learn, build, & test Regular Expressions
  - http://regexr.com/

Morphology and FSAs

- We'd like to use the machinery provided by FSAs to capture these facts about morphology
  - Accept strings that are in the language
  - Reject strings that are not
  - Determine whether an input string of letters make up a legitimate English words

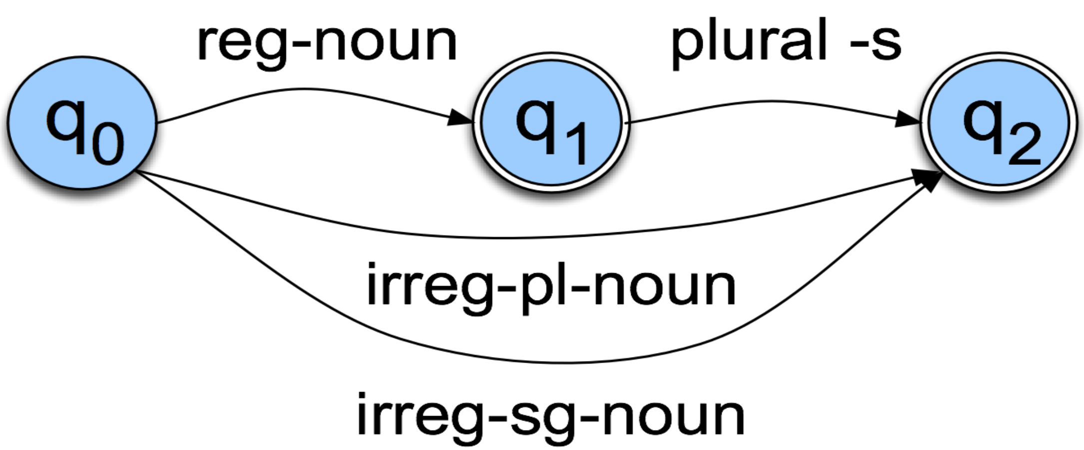
- So that we do not have to list all the words in the language

Spell checker?

Start Simple

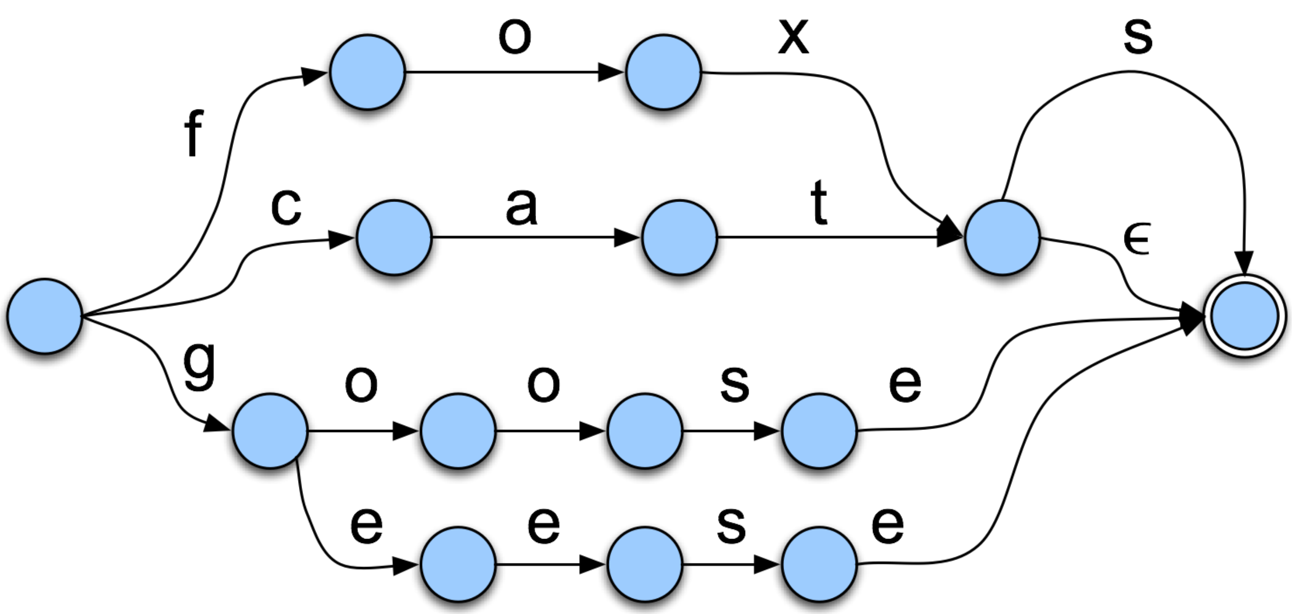
- Regular singular nouns are ok
  - Regular plural nouns have an -s on the end
- Irregulars are ok as is

- Simple Rules



Now, Plug in the Words

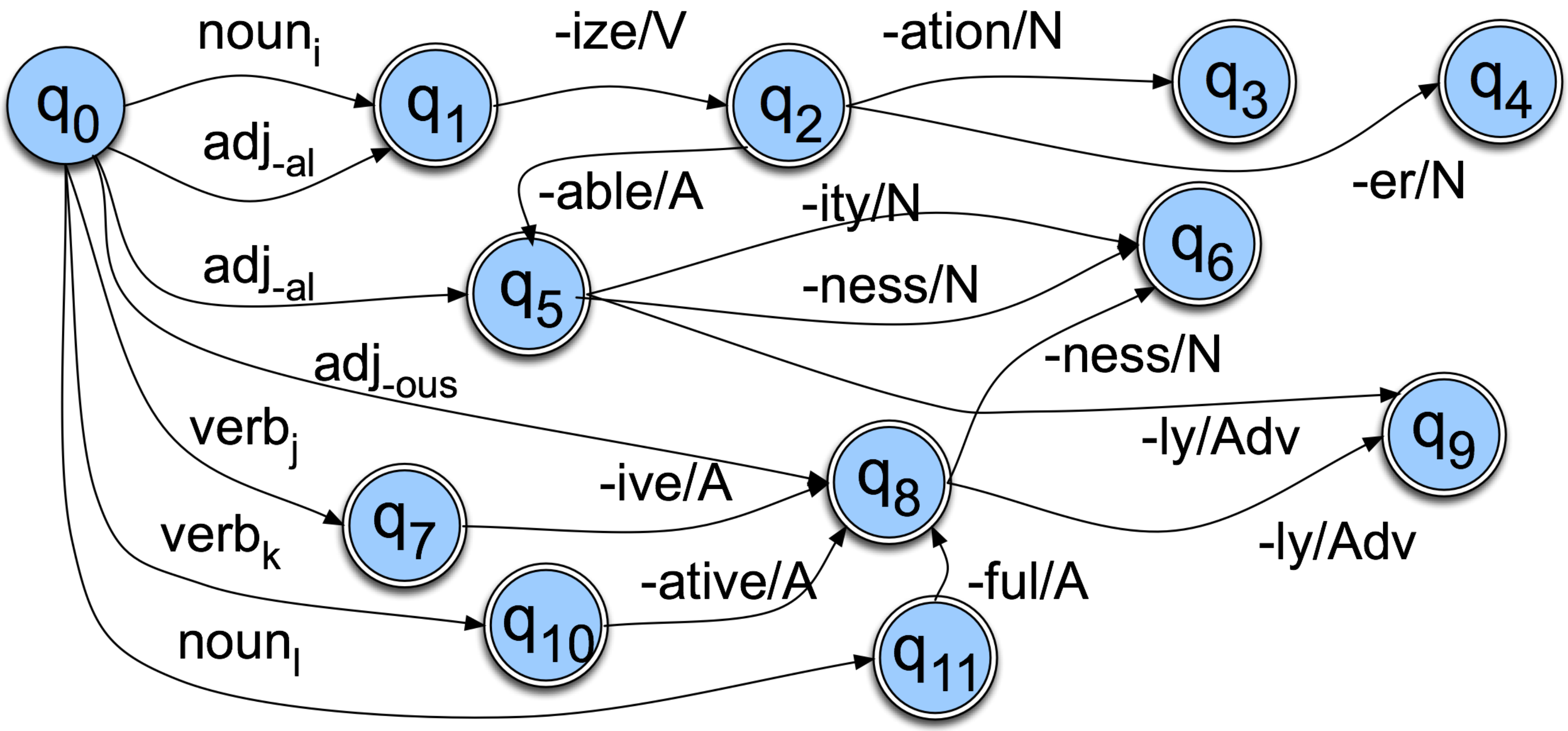
- Replace the class names like “reg-noun” with FSAs that recognize all the words in that class.



- Recognize strings, e.g. geese, goat, foxs
  - → more complicated solutions are needed

Derivational Rules

If everything is an accept state, how do things ever get rejected?



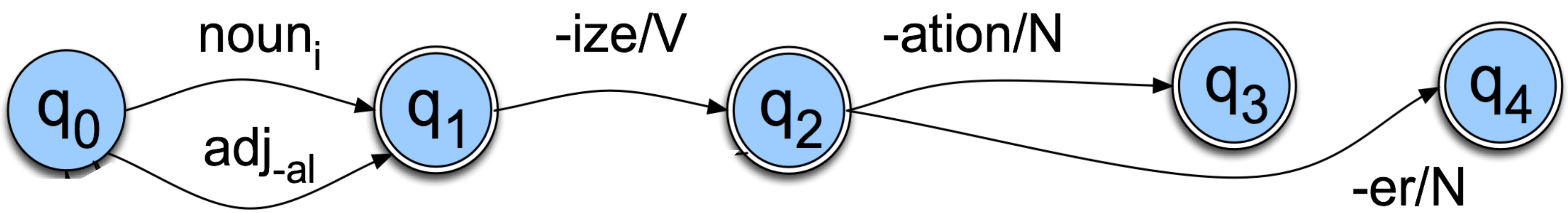
Notations:

noun$_i$: A subset of nouns that can accept *–ize*.

adj$_{al}$: Adjectives ending with *–al*

Exercise: Write a regular expression for the FSA

- A|B – A or B
- (ABC) – ABC as a component
- A? – A is optional



Parsing

- We can now run strings through these machines to recognize strings in the language
  - Spelling checking
- Often if we find some string in the language we might like to assign a structure to it (parsing)
  - From "cats" to "cat +N +PL"
  - From "caught" to "catch+V+past"
- The kind of parsing we're talking about is normally called morphological analysis

Applications

- Application 1: An important stand-alone component of many applications (spelling correction, information retrieval)

- Application 2: Simply a link in a chain of further linguistic analysis (e.g. parsing)

If no match, spelling error!

by using rules for different groups of words e.g. regular nouns