Natural Language Processing

Lecture 4

Today's Outline

Review REs, FSAs, and Deterministic Recognition

Non-Determinism

Recognition as Search

ND-Recognize

Fun with FSAs

First Homework

Review

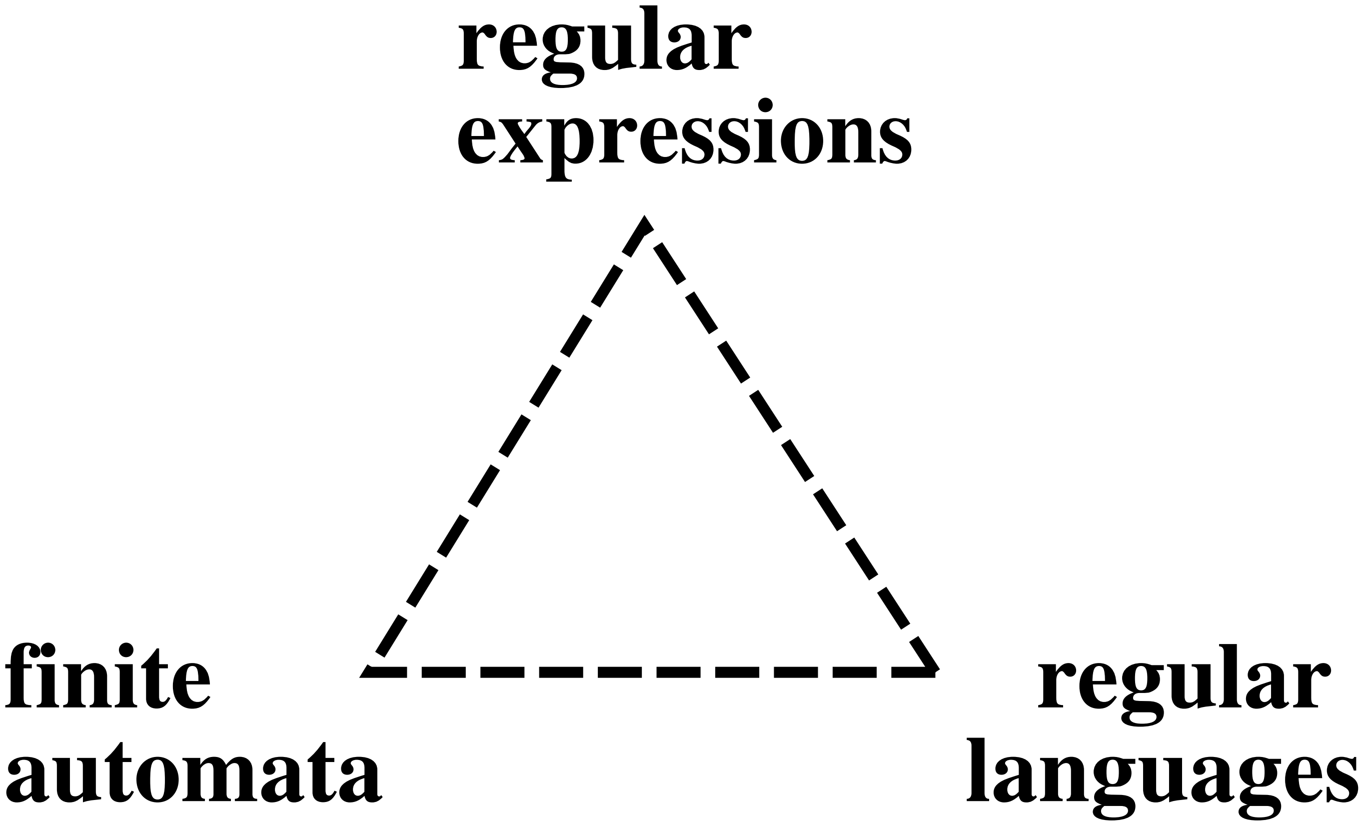
REs are compact textual representations of FSAs, themselves compact graphical representations of formal languages (sets of strings).

Recognition is the process of determining if an input is in the language accepted/generated/represented by a given FSA.

Recognition in deterministic machines is easy (table-driven).

Three Views

There are three formally equivalent ways of (not including tables) looking at what we're doing.



Well Four

We'll talk later about representing regular languages with rules like these...

S $\rightarrow$ b a a A

A $\rightarrow$ a A

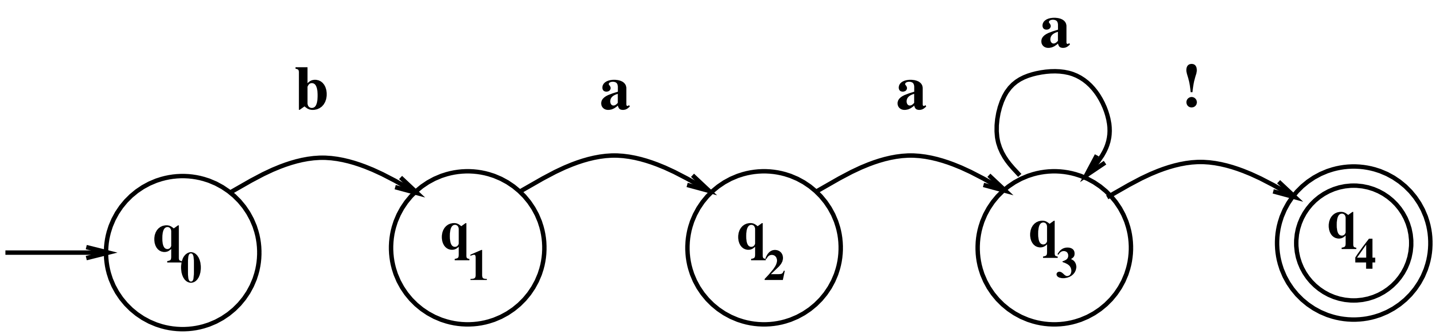
A $\rightarrow$!

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
```

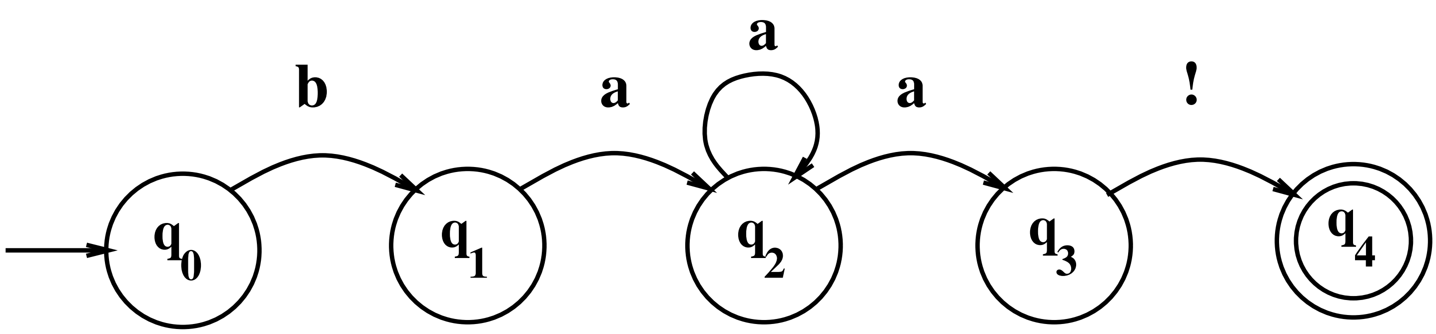
Non-Deterministic FSAs (NFSAs)

DFSA



- behavior is fully *determined* by current state and input

NFSA1

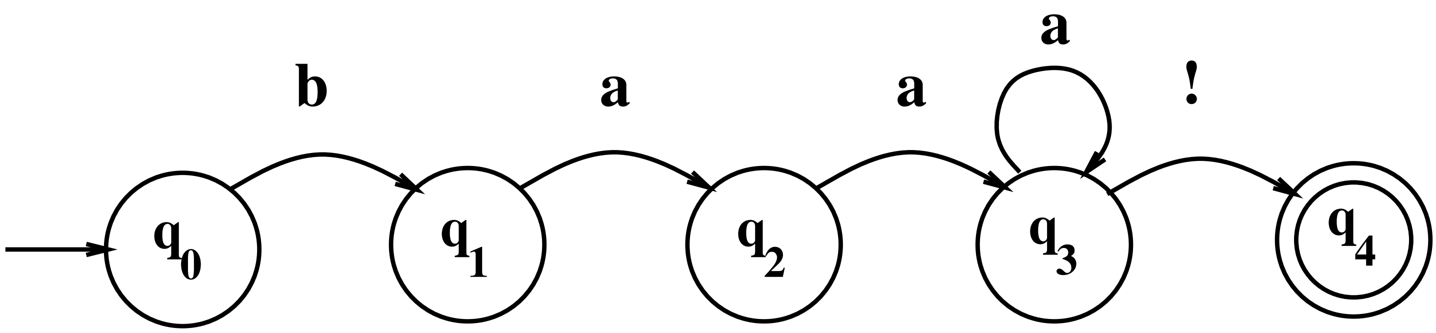


- an automata with *decision points*

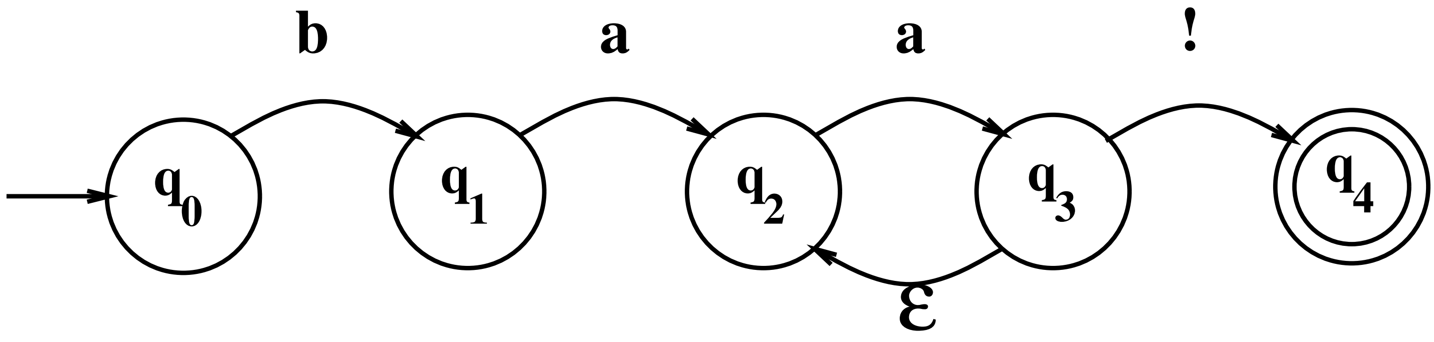
Non-Determinism (continued)

Yet another way of introducing non-determinism is to use ϵ-transitions (arcs with no symbols)

DFSA:

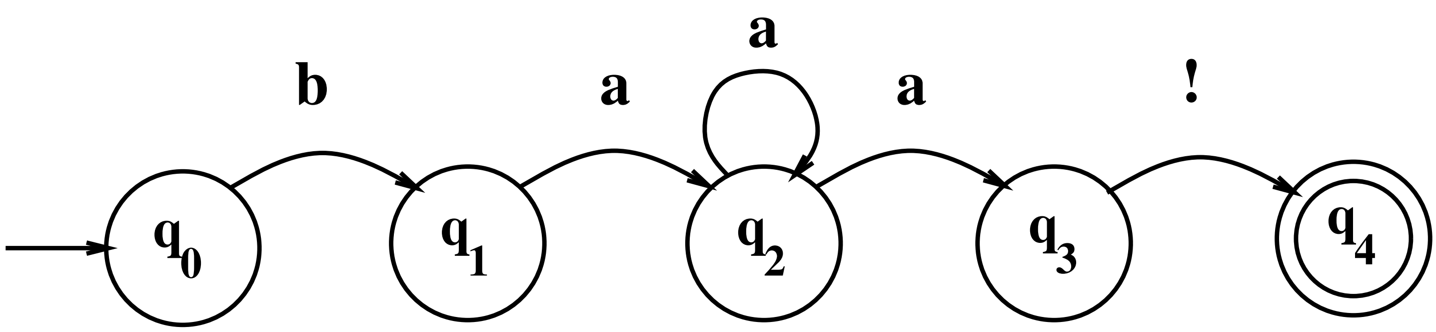


NFSA2:



Key Point: ϵ-transitions do not examine / advance the tape

NFSA Transition Tables



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

- epsilon input column
- cells can be *lists* of nodes

Equivalence of D and ND Machines

Any ND machine can be converted to a D machine by a fairly simple construction (e.g., Hopcroft and Ullman 1979).

This means that formally they have the same power - they can recognize exactly the same classes of language. Non-determinism does not add formal power to FSAs.

It also means that one way of doing recognition would be to convert an ND-machine to a D-machine and then do recognition with D-Recognize.

Non-Deterministic Recognition

Back to recognition . . .

In a non-deterministic machine, there exists at least one path directed through the machine by a string in the language that leads to an accept condition.

There may exist paths driven by strings in the language that do no lead to an accept condition. Doesn't matter if some wrong choices lead to a reject condition.

In a non-deterministic machine no path directed through the machine by a string outside the language leads to an accept condition.

Non-Determinism (continued)

So... non-deterministic recognition succeeds whenever it is driven by an input to any accept condition.

However, being driven to a reject condition by an input does not imply it should be rejected.

To reject a string it has to be sure that there are no paths that can lead to an accept.

That means that the problem of non-deterministic recognition can be thought of as a <u>search</u> problem.

The Problem of Choice

Choice in non-deterministic models comes up again and again in NLP

Several standard solutions

- backup (search, this chapter)
  - mark input/state at choice points
  - if wrong choice, use marker to backup and try another choice
- lookahead
  - look ahead in the input to help make choice
- parallelism
  - look at all choices in parallel

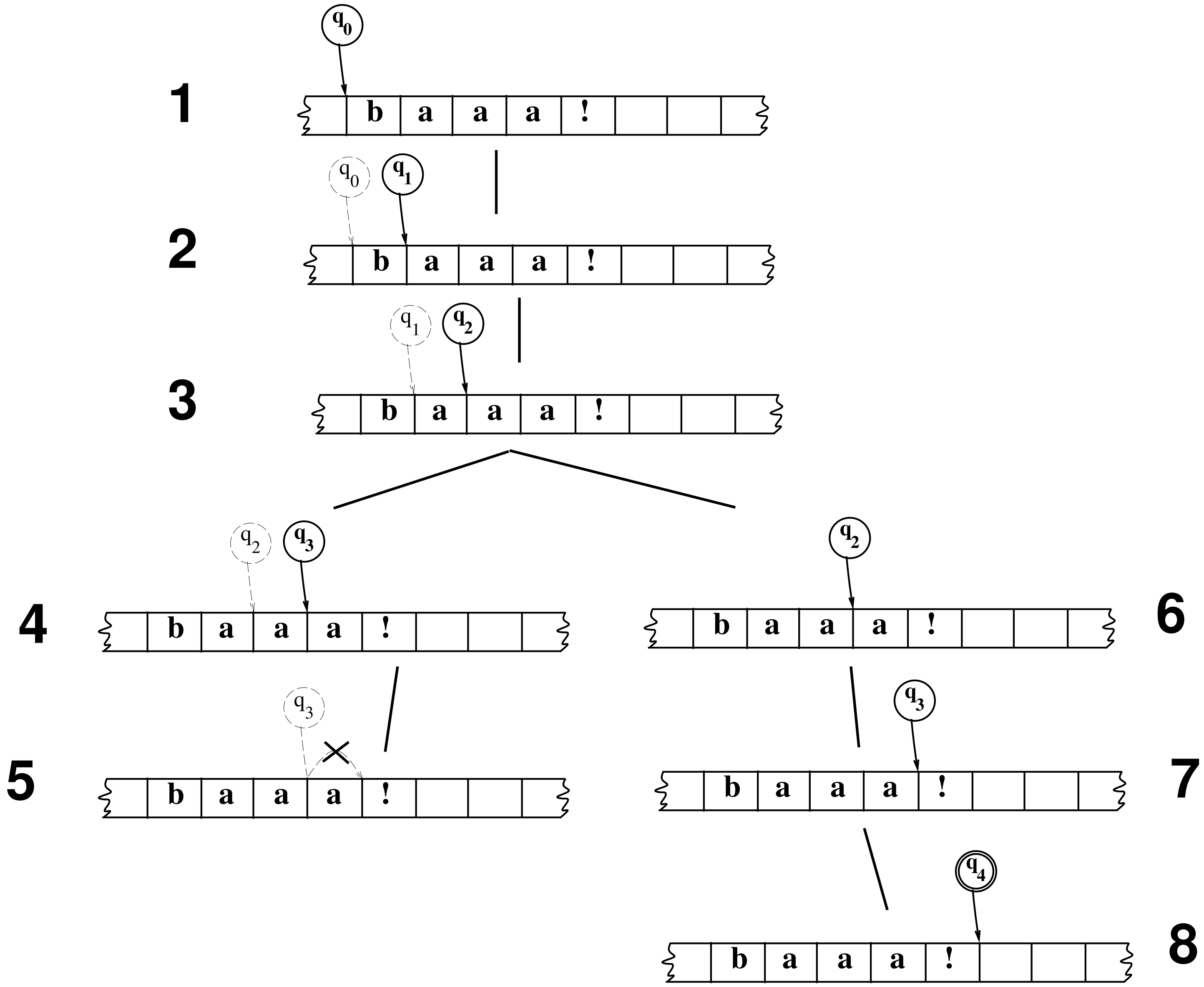
The Backup Approach

After a wrong choice leads to a dead-end (no input left, no legal transitions), return to a previous choice point and pursue another *unexplored* choice.

Thus, at each choice point, the search process needs to remember the search state:

- tape position
- FSA node (or *machine* state)

Backup Example



Key Points

A search-state is a pairing of a single machine-state with a position on the input tape.

By keeping track of not yet explored search-states, a recognizer can systematically explore all possible paths through a machine given some input.

ND-Recognize Code

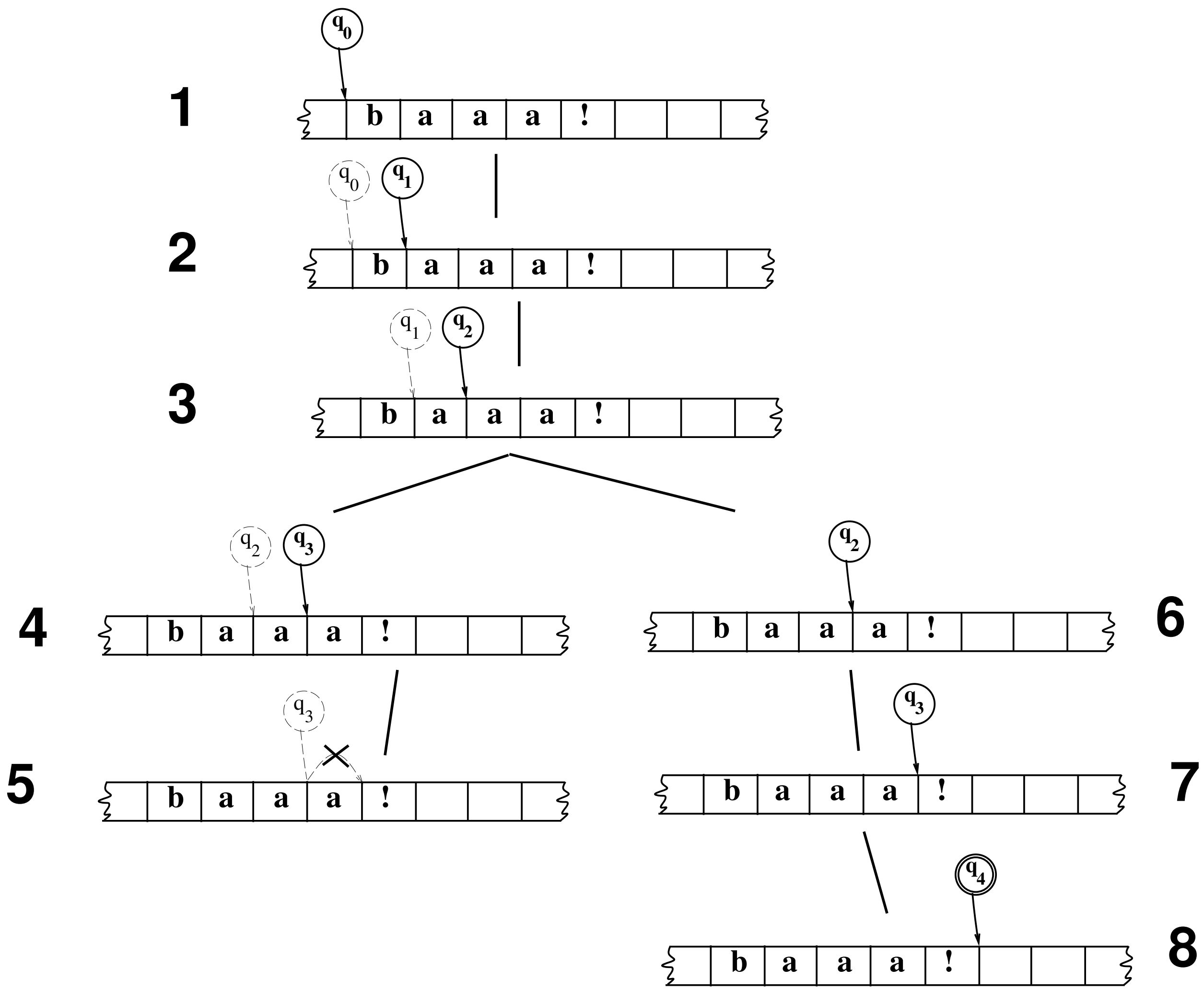
```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 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
```

Why Bother

Given all of the above, non-determinism seems to be more trouble that its worth. Why introduce it at all?

- more intuitive to solve certain problems
- often lots smaller (n versus possibly 2^n)

Example Revisited



- each # is the current search state
- transition table for $(q_2, \text{ a})$ examined twice, but tape pointer different

Recognition as Search

State-Space Search: Algorithms such as ND-Recognize, which systemically search for solutions

- problem definition creates a space of possible solutions
- goal is to explore the space
  - return answer when found
  - reject when space has been exhaustively explored

ND-Recognize as State-Space Search

Search states: pairings of machine states with tape position

State-space: all possible pairings given the machine

Search goal: find an accept state/end of tape pair

Ordering the Search

Note that NEXT is undefined in ND-Recognize

Depth-first search

- most recently created states considered first
- agenda is a stack
- last in first out (LIFO)
- previous example trace

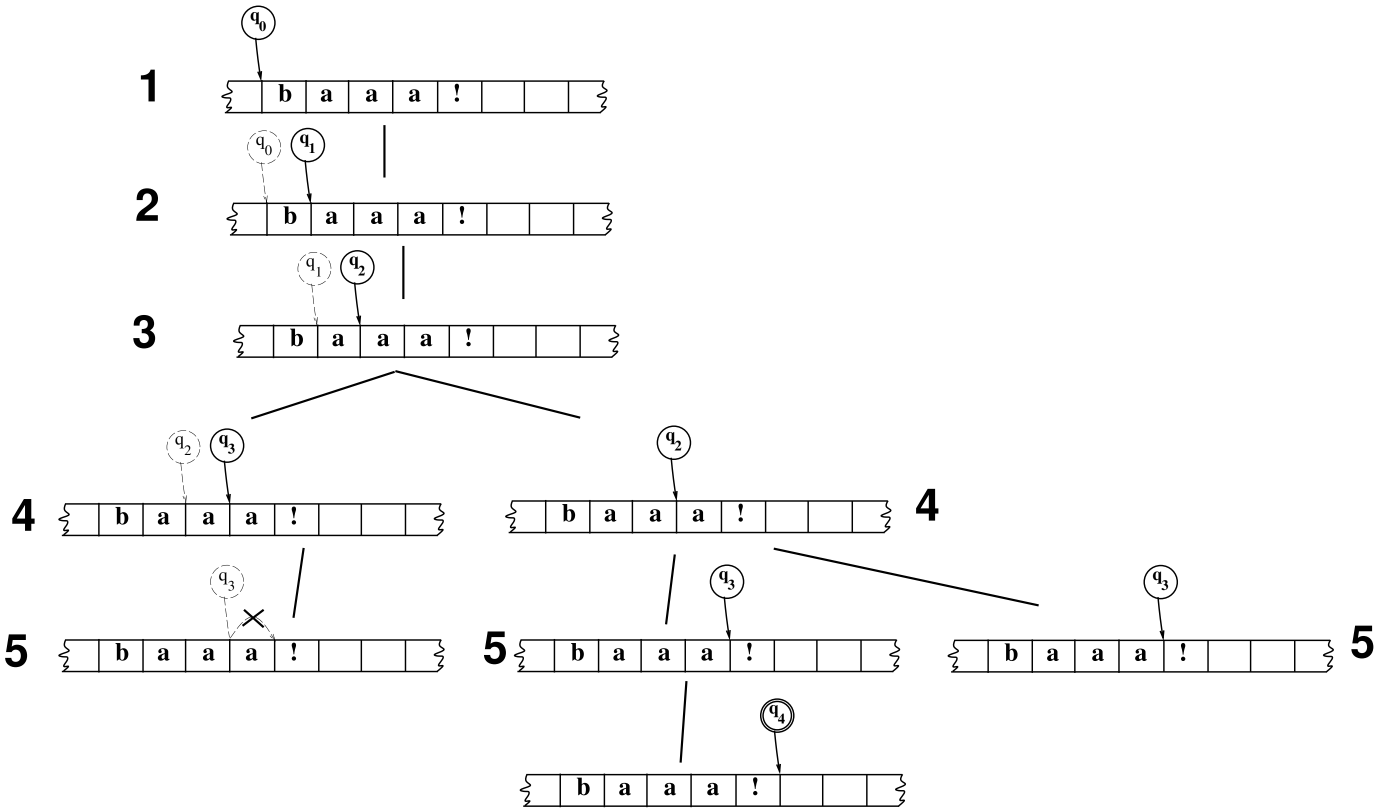
Breadth-first search

- states considered in creation order
- agenda is a queue
- first in first out (FIFO)

Dynamic programming / A*

- will be discussed later in the course

Breadth-First Trace



- instead of picking one choice and following it, examine all possible choices
- typically less efficient with respect to memory

Regular Languages

Equivalent

- the class of languages that are definable by regular expressions
- the class of languages that are characterizable by FSAs (either deterministic or not)

Formal Definition

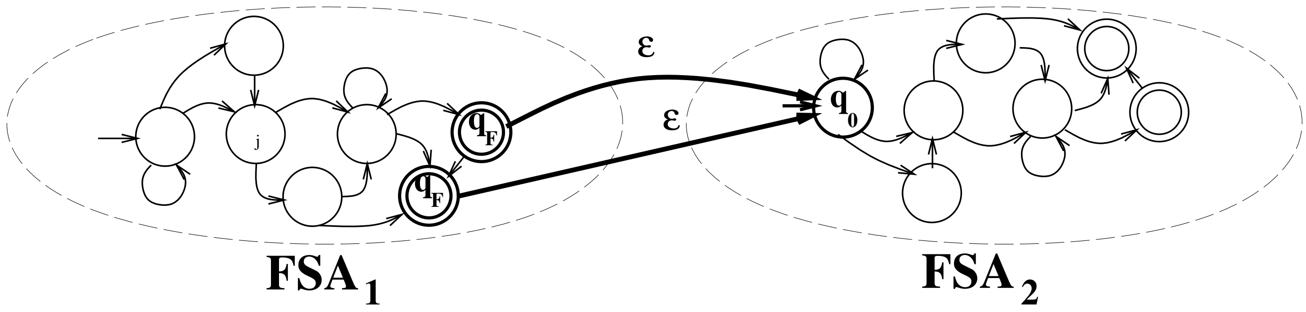
- if L_1 and L_2 are regular languages, so are the concatenation of L_1 and L_2, the union or disjunction of L_1 and L_2, and the Kleene closure of L_1
- thus, all the regular expression operators introduced in J&M Chapter 2 (except memory) can be implemented by these three operators

Fun With Automata

It turns out to be interesting (and useful) to consider the implications of various operations that combine regular languages into new languages.

In particular, we may want to know if the resulting combined language is still regular. Why?

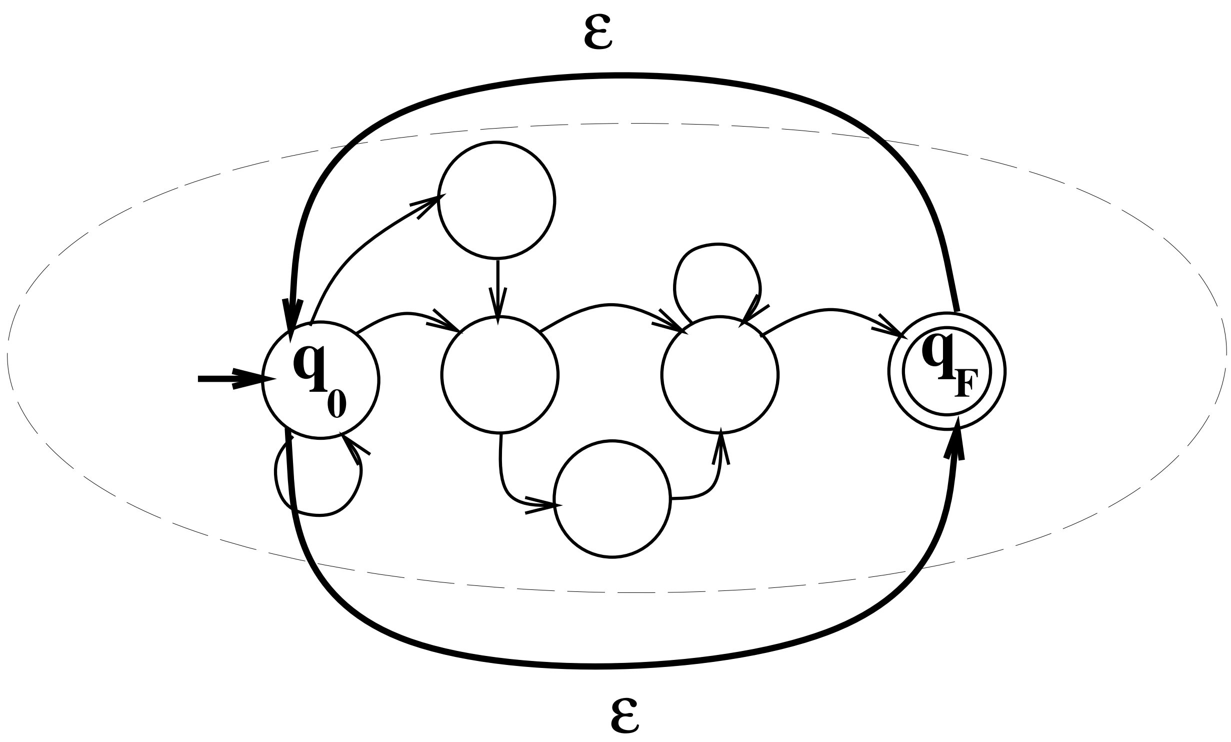
Concatenation



The primitive concatenation operation of a regular expression can be imitated by an automaton.

Use epsilon transitions to connect all final states of the first machine to the single initial state of the second.

Kleene *

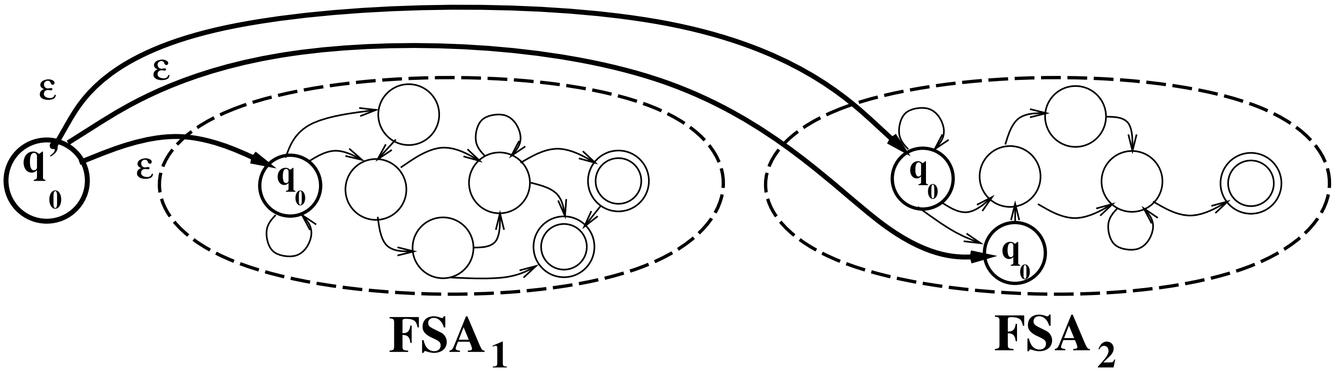


Connect all final states back to the initial state (repetition)

Directly link initial and final state (zero)

How would you implement Kleene + ?

Union



Add a new initial state with epsilon transitions to all the former initial states.

Closure (continued)

Regular languages are also closed under other operations (e.g., intersection) - see Jurafsky and Martin page 50.

Composing Automata

Consider the problem of recognizing dates like the following:

- January 27, 2000
- Wednesday January 27, 2000
- Monday February 29, 1900

One solution (a bad one) is to build one big complex regular expression.

What might a better solution be?

For Next Time

Chapter 3

Homework: look at syllabus for details

Acknowledgements

Some of this material was taken from the course web pages of James Martin.