Finite-state methods

Slides by James Martin, adapted by Diana Inkpen
for CSI 5386 @ uOttawa

Regular Expressions and Text Searching

- Regular expressions are a compact textual representation of a set of strings that constitute a language
  - In the simplest case, regular expressions describe regular languages
    - Here, a language means a set of strings given some alphabet.
- Extremely versatile and widely used technology
  - Emacs, vi, perl, grep, etc.

Example

- Find all the instances of the word "the" in a text.
  - `/the/`
  - `/[tT]he/`
  - `/\b[tT]he\b/`

Errors

- The process we just went through was based on two fixing kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors

- We'll be telling the same story with respect to evaluation for many tasks. Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy, or precision, (minimizing false positives)
  - Increasing coverage, or recall, (minimizing false negatives).

3 Formalisms

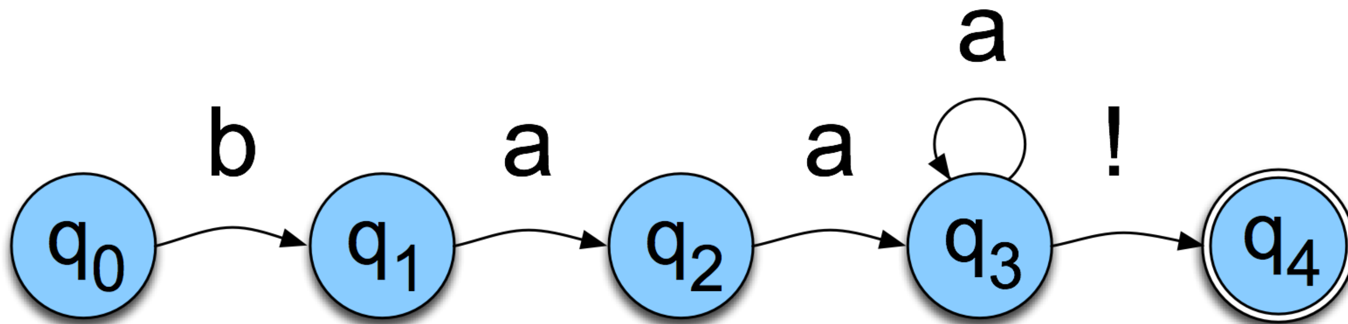
- Regular expressions describe languages (sets of strings)
- Turns out that there are 3 formalisms for capturing such languages, each with their own motivation and history
  - Regular expressions
    - Compact textual strings
      - Perfect for specifying patterns in programs or command-lines
  - Finite state automata
    - Graphs
  - Regular grammars
    - Rules

3 Formalisms

- These three approaches are all equivalent in terms of their ability to capture regular languages. But, as we'll see, they do inspire different algorithms and frameworks

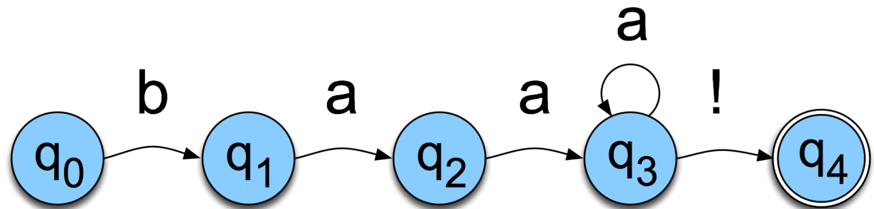
FSAs as Graphs

- Let' s start with the sheep language from Chapter 2
  - /baa+!/



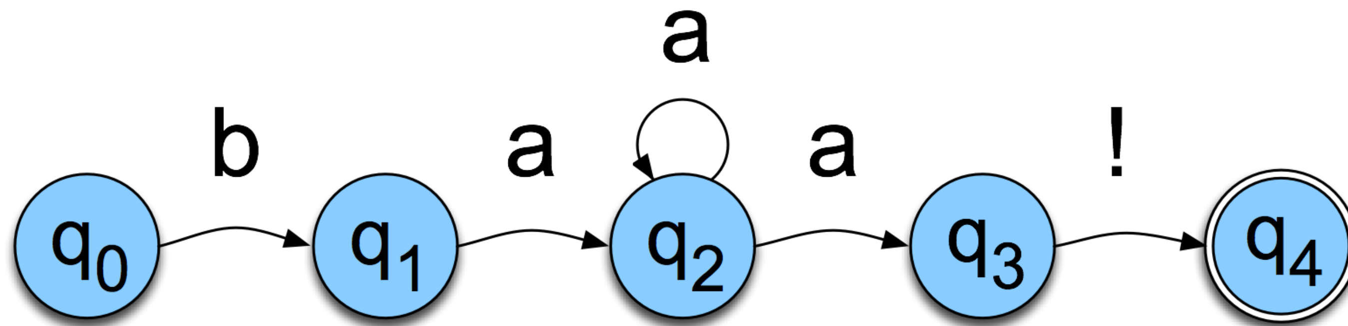
Sheep FSA

- 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



But Note

- There are other machines that correspond to this same language



- More on this one later

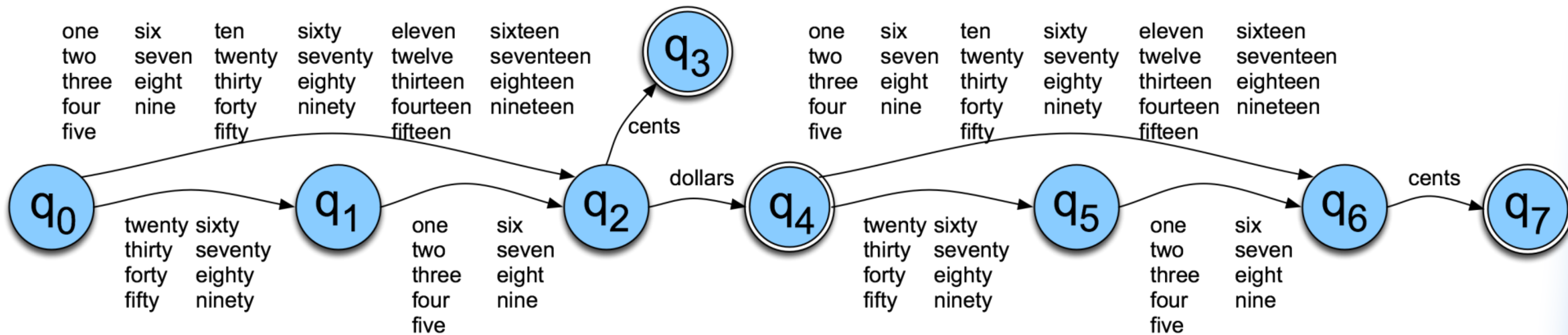
More Formally

- 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 states
  - A transition function that maps QxΣ to Q

About Alphabets

- Don't take term *alphabet* word too narrowly; it just means we need a finite set of symbols in the input.
- These symbols can and will stand for bigger objects that may in turn have internal structure
  - Such as another FSA

Dollars and Cents

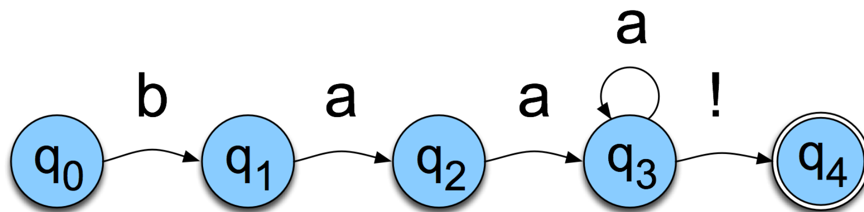


Yet Another View

- The guts of an FSA can ultimately be represented as a table

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

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

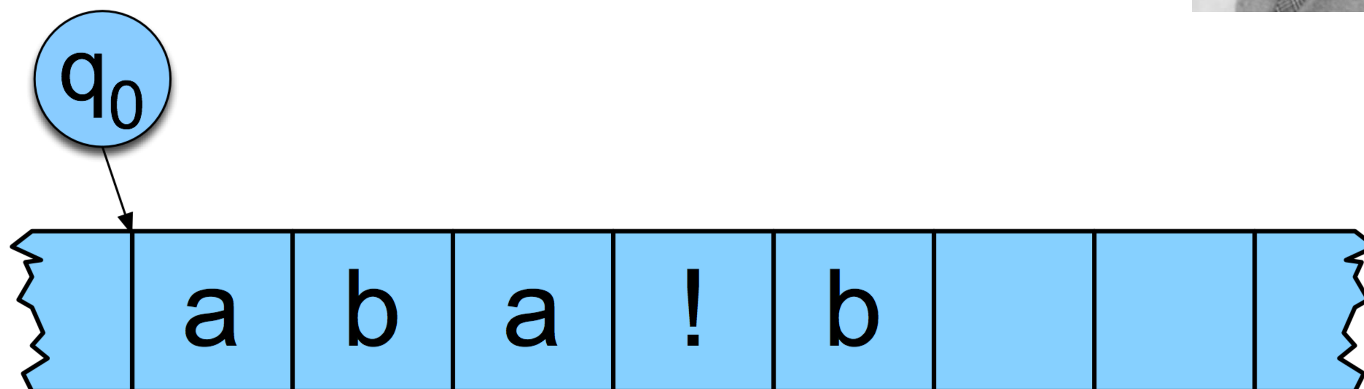


Recognition

- 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
- Those all amount the same thing in the end

Recognition

- Traditionally, (Turing's notion) this process is depicted with an input string written on a tape.



Recognition

- 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.

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
```

Key Points

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

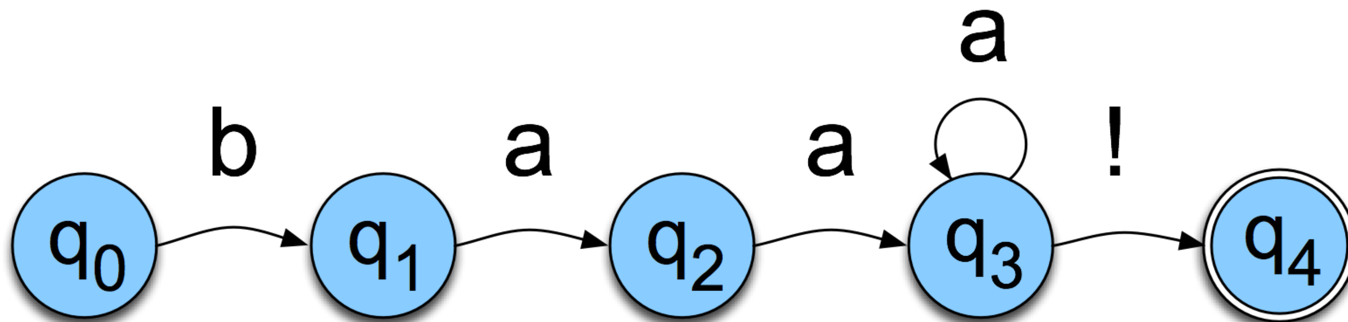
Key Points

- Crudely therefore… matching strings with regular expressions (ala 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 implements D-recognize (or something like it)

Recognition as Search

- You can view this algorithm as a trivial kind of *state-space search*
- Search states are pairings of tape positions and state numbers
- Operators are compiled into the table
- Goal state is a pairing with the end of tape position and a final accept state
- Why is it trivial?

Non-Determinism



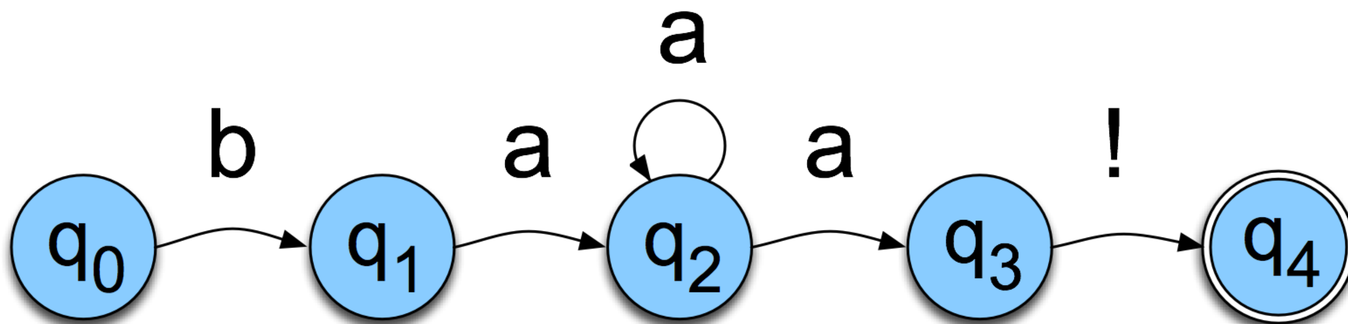
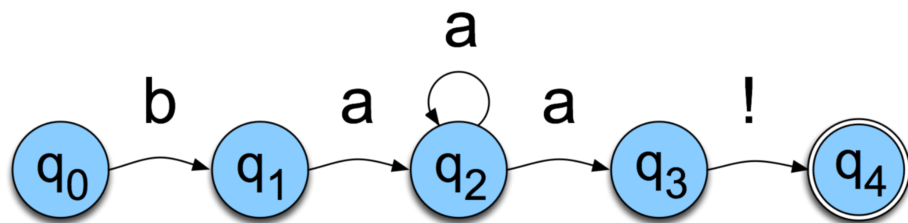


Table View

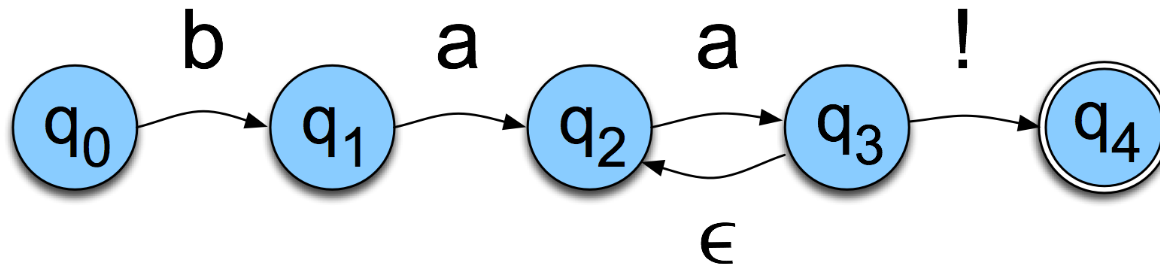
Allow multiple entries in the table to capture non-determinism

	b	a	!	
0	1			
1		2		
2		2,3		
3			4	
4				



Non-Determinism cont.

- Yet another technique
  - Epsilon transitions
  - Key point: 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

ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
  1. Either take a ND machine and convert it to a D machine and then do recognition with that.
  2. Or explicitly manage the process of recognition as a state-space search (leaving the machine/table as is).

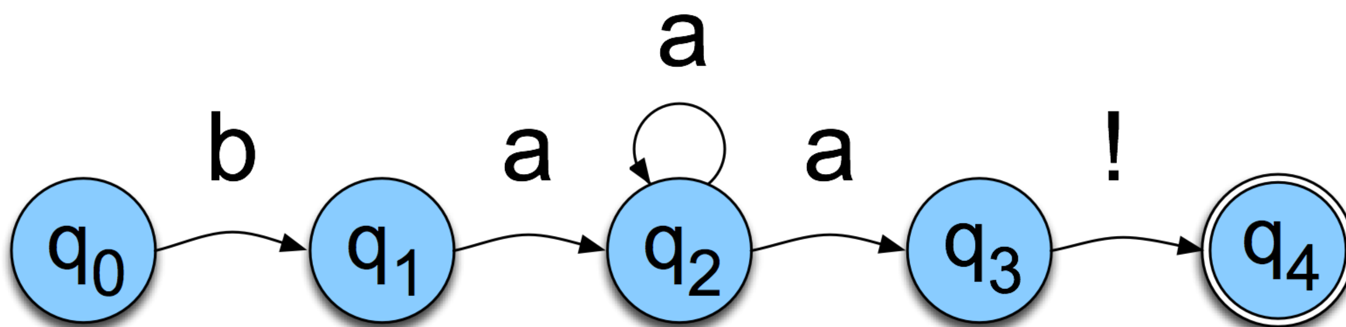
Non-Deterministic Recognition: Search

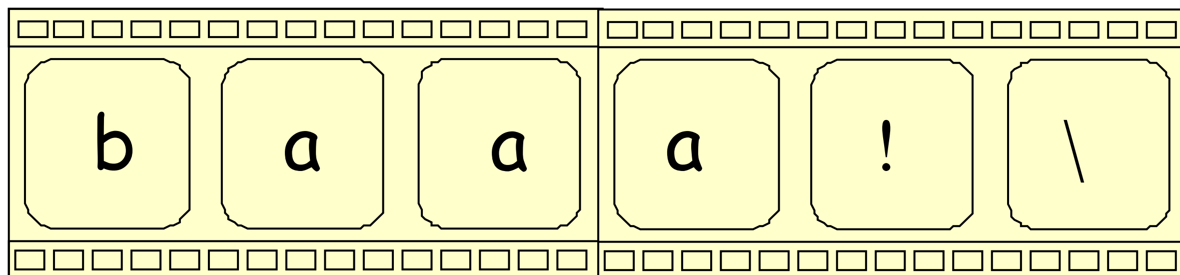
- In a ND FSA there exists at least one path through the machine for a string that is in the language defined by the machine.
- But not all paths directed through the machine for an accept string lead to an accept state.
- No paths through the machine lead to an accept state for a string not in the language.

Non-Deterministic Recognition

- So 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.

Example

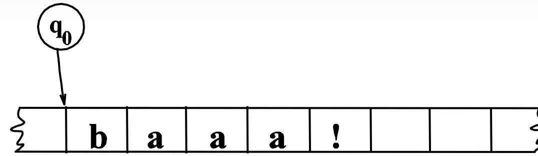


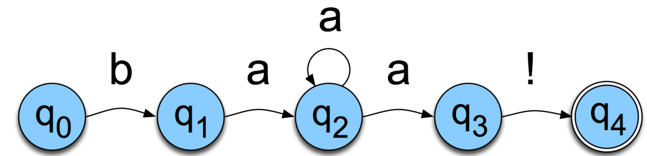


q_0 q_1 q_2 q_2 q_3 q_4

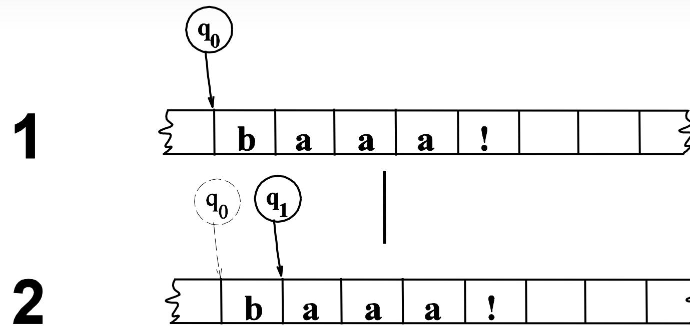
Example

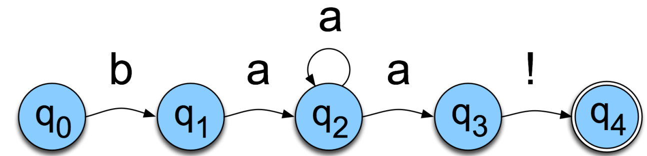
1



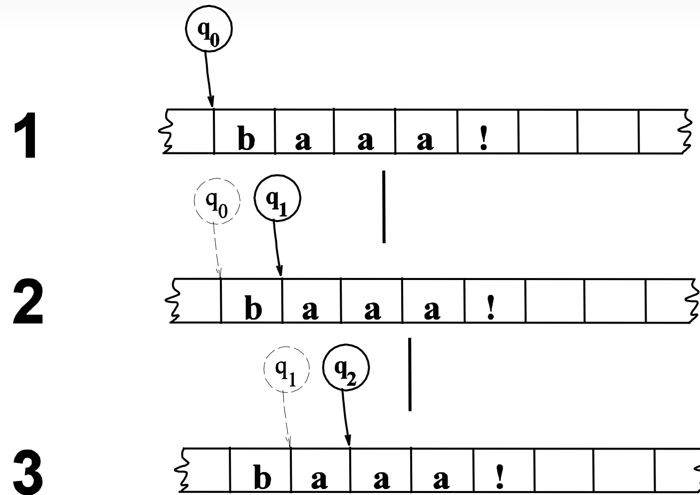


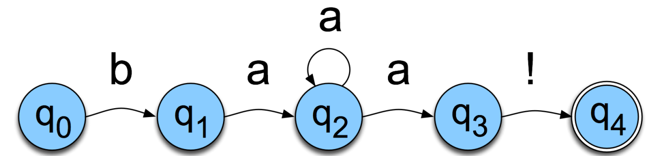
Example



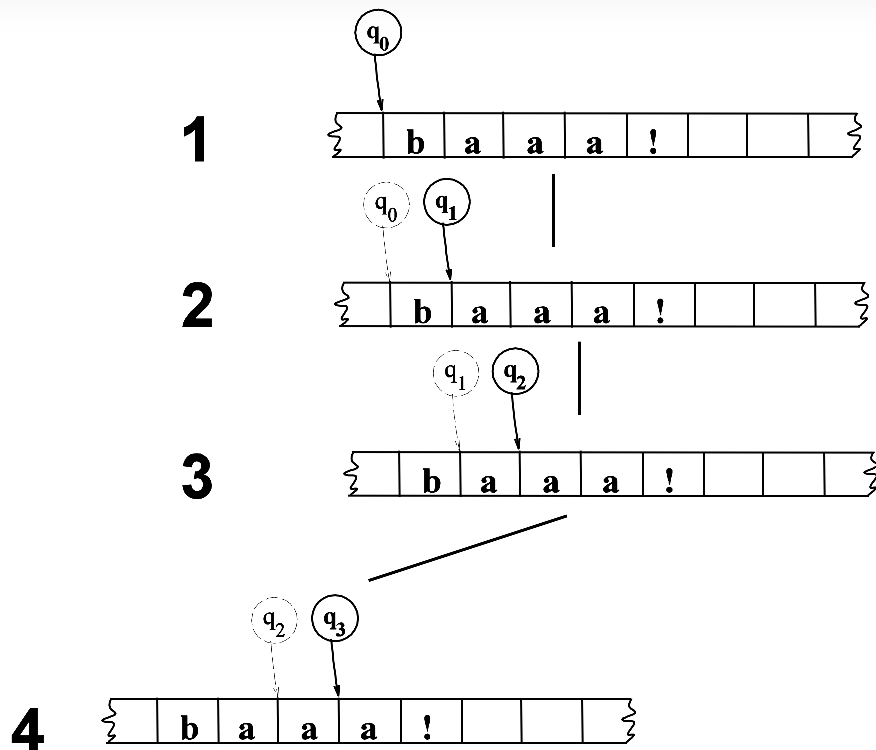


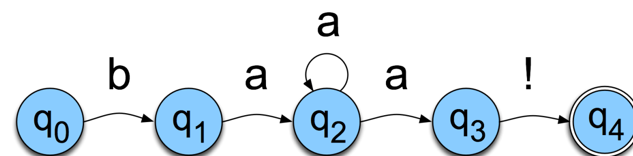
Example



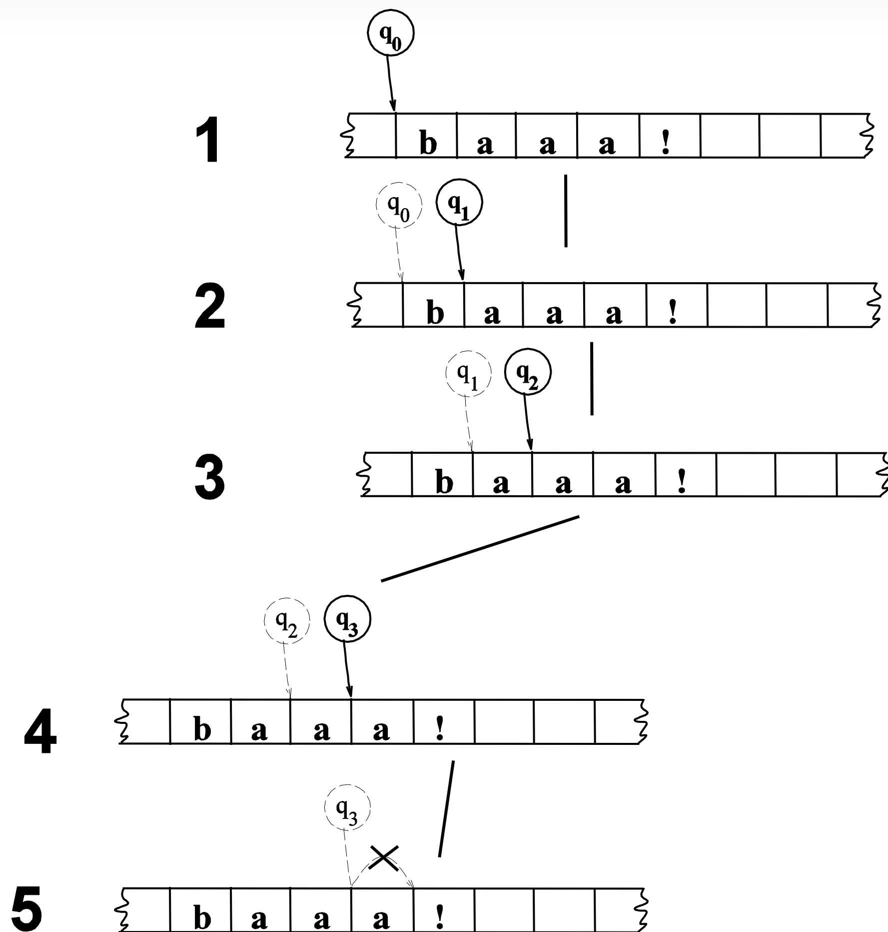


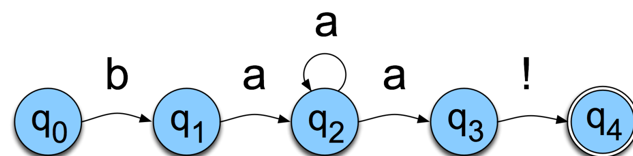
Example



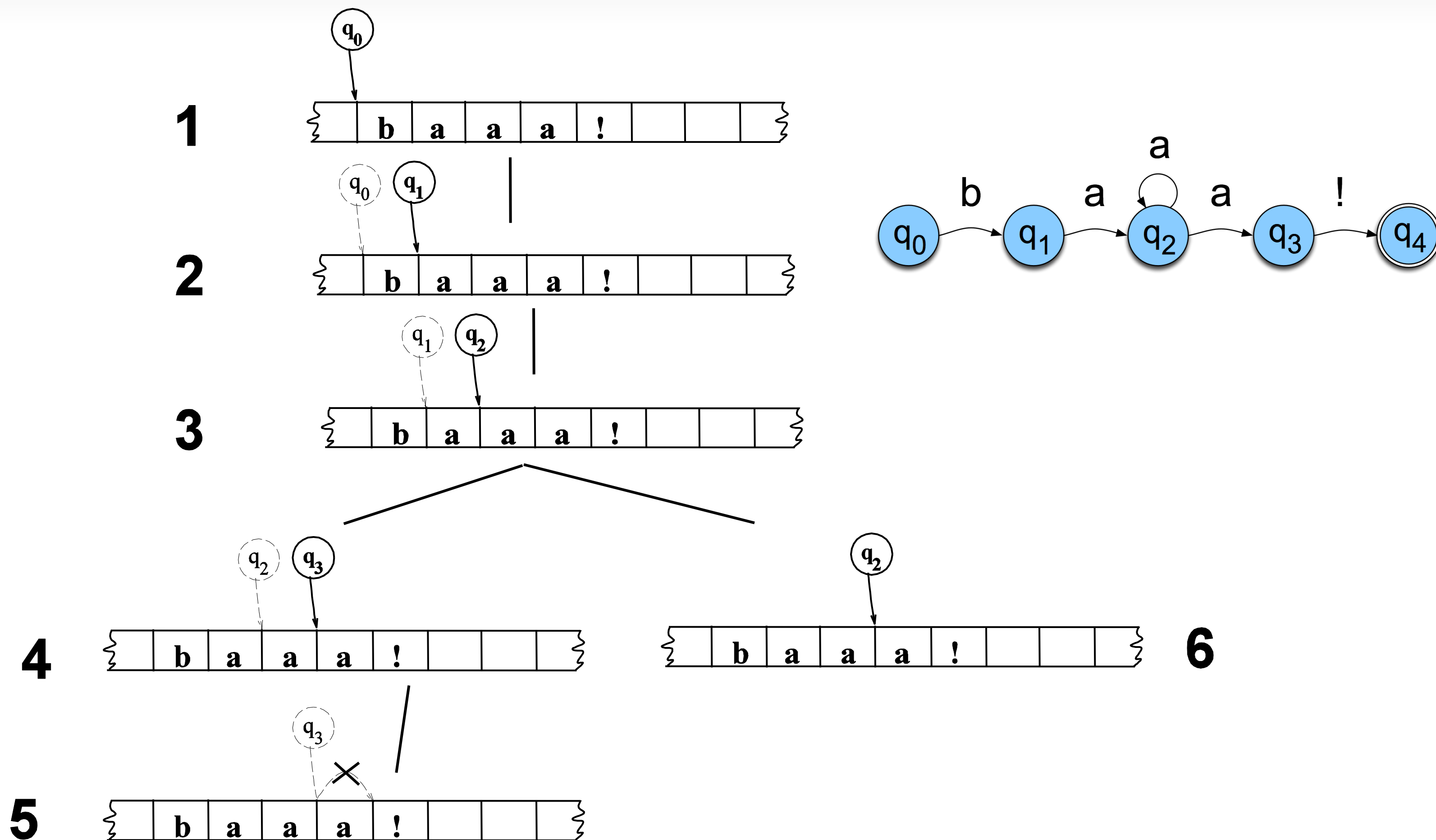


Example

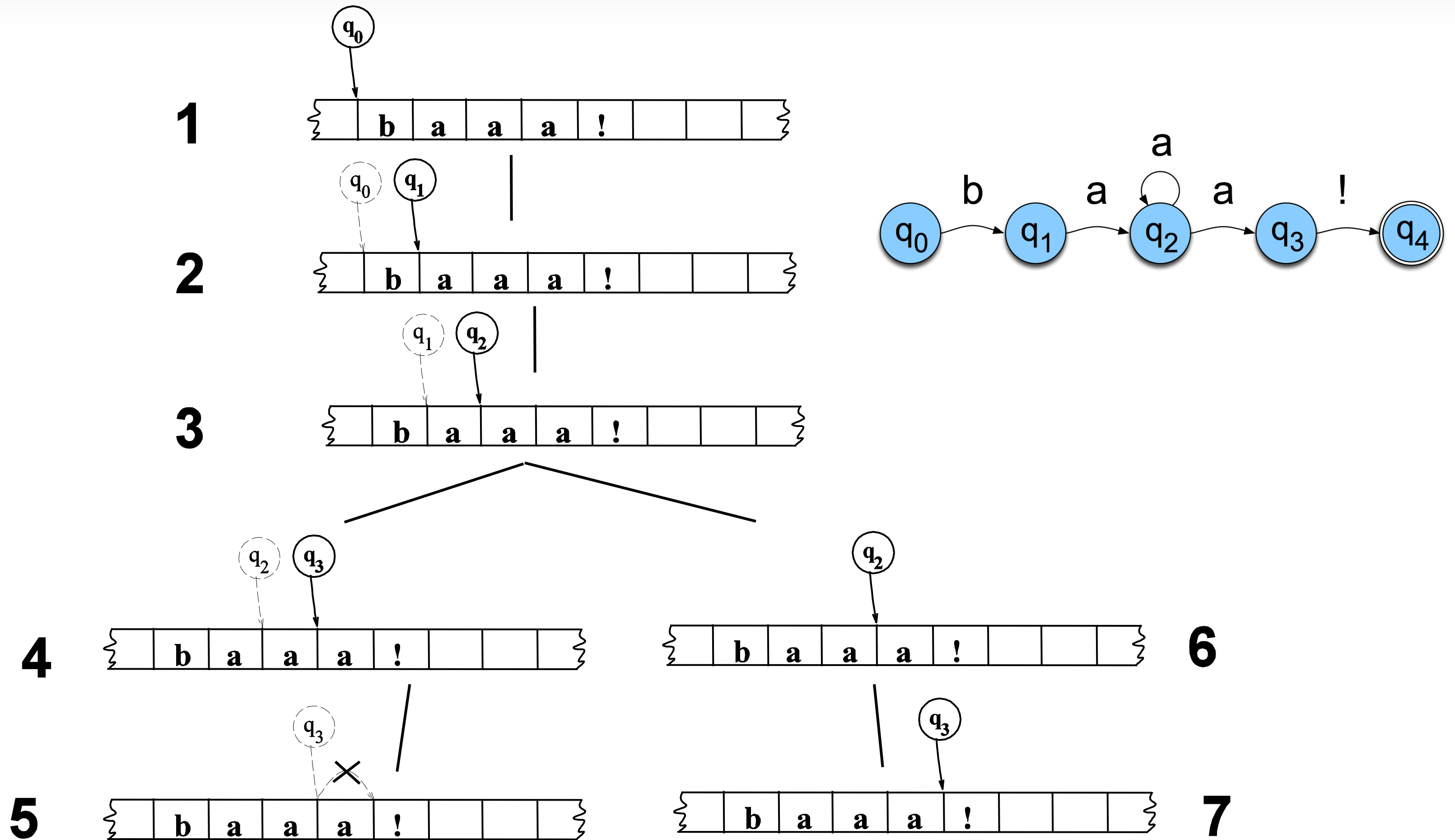




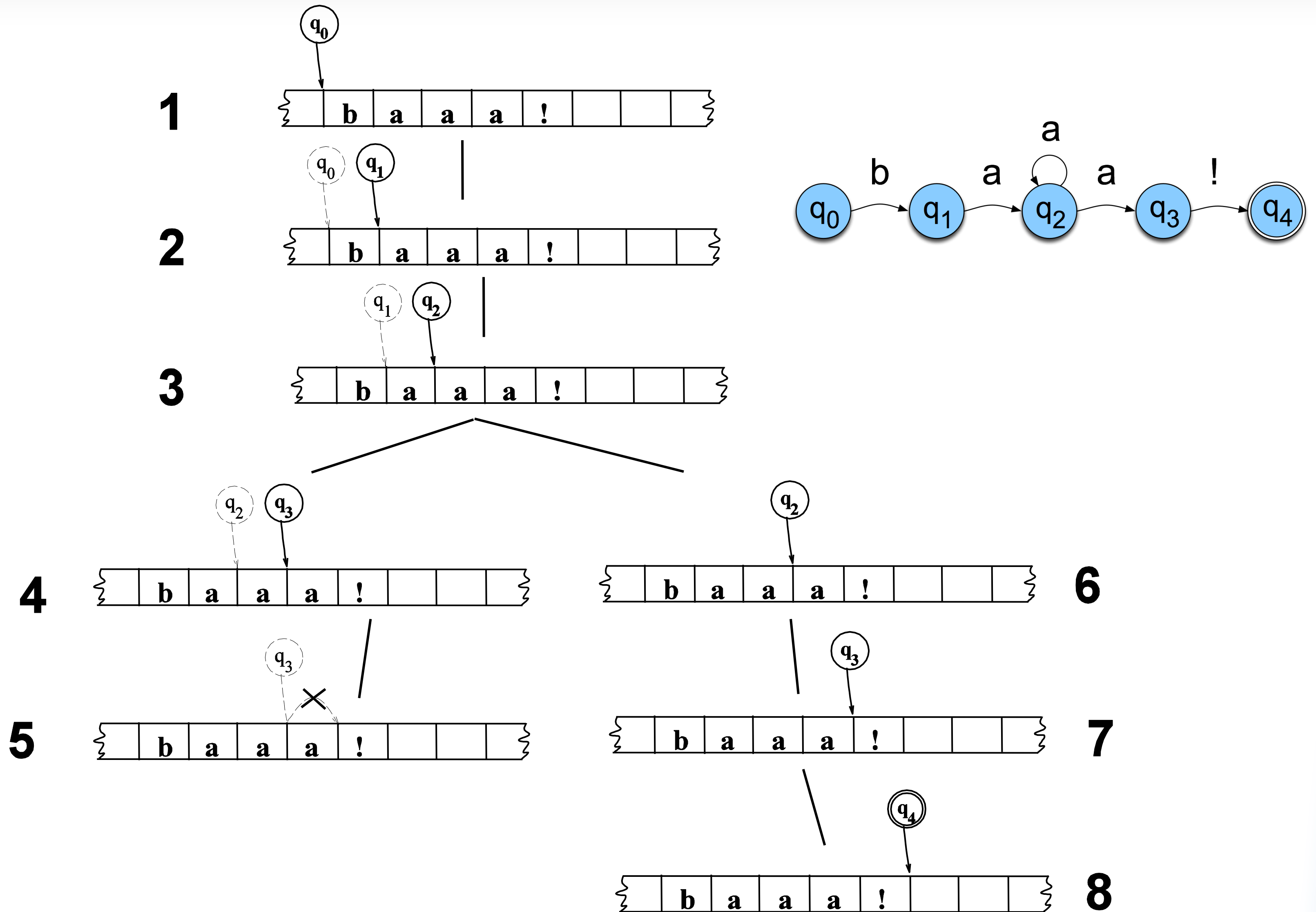
Example



Example



Example



Key Points

- States in the search space are pairings of tape positions and states in the machine.
- By keeping track of as yet unexplored states, a recognizer can systematically explore all the paths through the machine given an input.

Why Bother?

- 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