Regular Expressions

- using REs to find patterns
- implementing REs using finite state automata

REs and FSAs

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata
- Finite-state automata are a way of implementing regular expressions

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions for Textual Searches

Who does it?

Everybody:

- Web search engines, CGI scripts
- Information retrieval
- Word processing (Emacs, vi, MSWord)
- Linux tools (sed, awk, grep)
- Computation of frequencies from corpora
- Perl

http://xkcd.com/

Regular Expression

- **Regular expression:** formula in algebraic notation for specifying a set of strings
- **String:** any sequence of alphanumeric characters
  - letters, numbers, spaces, tabs, punctuation marks
- **Regular expression search**
  - **pattern:** specifying the set of strings we want to search for
  - **corpus:** the texts we want to search through

Basic Regular Expression Patterns

- Case sensitive: d is not the same as D
- Disjunctions: **`[dD]`** **`[0123456789]`**
- Ranges: **`[0-9]`** **`[A-Z]`**
- Negations: **`[^Ss]`** *(only when ^ occurs immediately after [ )*
- Optional characters: **`?`** and **`*`**
- Wild : **`.`**
- Anchors: **`^`** and **`$`**, also **\b** and **\B**
- Disjunction, grouping, and precedence: **|** **(pipe)**

Caret for negation, ^ , or anchor

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"
^T	T at the beginning of a line	"The Dow Jones closed up one"

Optionality and Counters

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"The woodchuck hid"
`colou?r`	color or colour	"comes in three colours"
`(he){3}`	exactly 3 "he"s	"and he said hehehe."

? zero or one occurrences of previous char or expression

* zero or more occurrences of previous char or expression

+ one or more occurrences of previous char or expression

{n} exactly n occurrences of previous char or expression

{n, m} between n to m occurrences

{n, } at least n occurrences

Wild card ‘ .’

RE	Match	Example Patterns Matched
`beg.n`	any char between *beg* and *n*	begin, beg’n, begun
`big.*dog`	find lines where big and dog occur	the big dog bit the little the big black dog bit the

Operator Precedence Hierarchy

1.	Parenthesis	( )
2.	Counters	* + ? { }
3.	Sequences and Anchors	the ^my end$
4.	Disjunction	|

Examples:

`/moo+/`

`/try|ies/`

`/and|or/`

Example

- Find all instances of the word “the” in a text.

Example

- Find all instances of the word "the" in a text.
  - `/the/`

Example

- Find all instances of the word "the" in a text.
  - `/the/`

    `Misses capitalized examples`

Example

- Find all instances of the word "the" in a text.
  - `/the/`

    `Misses capitalized examples`
  - `/[tT]he/`

Example

- Find all instances of the word “the” in a text.
  - `/the/`
    - `Misses capitalized examples`
  - `/[tT]he/`
    - `Finds other or theology`

Example

- Find all instances of the word "the" in a text.
  - /the/
    Misses capitalized examples
  - /[tT]he/
    - Finds other or theology
  - /\b[tT]he\b/

Example

- Find all instances of the word "the" in a text.
  - `/the/`
    Misses capitalized examples
  - `/[tT]he/`
    - Finds other or theology
  - `/\b[tT]he\b/`
  - `/[^a-zA-Z][tT]he[^a-zA-Z]/`

Example

- Find all instances of the word “the” in a text.
  - /the/

    Misses capitalized examples
  - /[tT]he/
    - Finds other or theology
  - /\b[tT]he\b/
  - /[^a-zA-Z][tT]he[^a-zA-Z]/
    - Misses sentence-initial “the”

Example

- Find all instances of the word “the” in a text.
  - /the/
    - Misses capitalized examples
  - /[tT]he/
    - Finds other or theology
  - /\b[tT]he\b/
  - /[^a-zA-Z][tT]he[^a-zA-Z]/
    - Misses sentence-initial “the”
  - /(^|[^a-zA-Z])[tT]he[^a-zA-Z]/

Errors

- The process we just went through was based on fixing two 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)

A more complex example

Write a RE that will match “*any PC with more than 500MHz and 32 Gb of disk space for less than $1000”.*

- First a RE for prices

```
/$[0-9]+/                              # whole dollars
/$[0-9]+\.[0-9][0-9]/                  # dollars and cents
/$[0-9]+(\.[0-9][0-9])?/               #cents optional
/\b$[0-9]+(\.[0-9][0-9])?\b/           #word boundaries
```

A more complex example

Write a RE that will match “*any PC with more than 500MHz and 32 Gb of disk space for less than $1000”.*

- First a RE for prices

```
/$[0-9]+/                          # whole dollars
/$[0-9]+\.[0-9][0-9]/              # dollars and cents
/$[0-9]+(\.[0-9][0-9])?/           #cents optional
/\b$[0-9]+(\.[0-9][0-9])?\b/       #word boundaries
```

Continued

- Specifications for processor speed

```
/\b[0-9]+ *(MHz|[Mm]egahertz|Ghz|[Gg]igahertz)\b/
```

- Memory size

```
/\b[0-9]+ *(Mb|[Mm]egabytes?)\b/
/\b[0-9](\.[0-9]+) *(Gb|[Gg]igabytes?)\b/
```

- Vendors

```
/\b(Win(95|98|NT|dows *(NT|95|98|2000)?))\b/
/\b(Mac|Macintosh|Apple)\b/
```

Substitutions and Memory

- Substitutions: `s/regexp/pattern/)`

```
s/color/colour/
```

- Memory (`\1`, `\2`, etc. refer back to found matches) e.g., Put angle brackets around all integers in text

the 39 students ==> the <39> students

```
s/([0-9]+)/<\1>/
```

Using Backslash

RE	Match	Example Patterns Matched
`\*`	an asterisk "*"	"K*A*P*L*A*N"
`\.`	a period "."	"Dr. Livingston, I presume"
`\?`	a question mark	"Would you light my candle?"
`\n`	a newline	
`\t`	a tab	

Some Useful Aliases

RE	Expansion	Match	Example Patterns
\d	[0-9]	any digit	Party of 5
\D	[^0-9]	any non-digit	99p
\w	[a-zA-Z0-9_]	any alphanumeric or underscore	99p
\W	[^\w]	a non-alphanumeric	!!!!
\s	[\r\t\n\f]	whitespace (sp, tab)	
\S	[^\s]	Non-whitespace	in Concord

Substitutions and Memory

- Substitutions: `s/regexp/pattern/)`

```
s/color/colour/
```

- Memory (`\1`, `\2`, etc. refer back to found matches) e.g., Put angle brackets around all integers in text

  the 39 students ==> the <39> students

```
s/([0-9]+)/<\1>/
```

Example

Swap first two words of line

```
s/(\w+) +(\w+)/\2 \1/
```

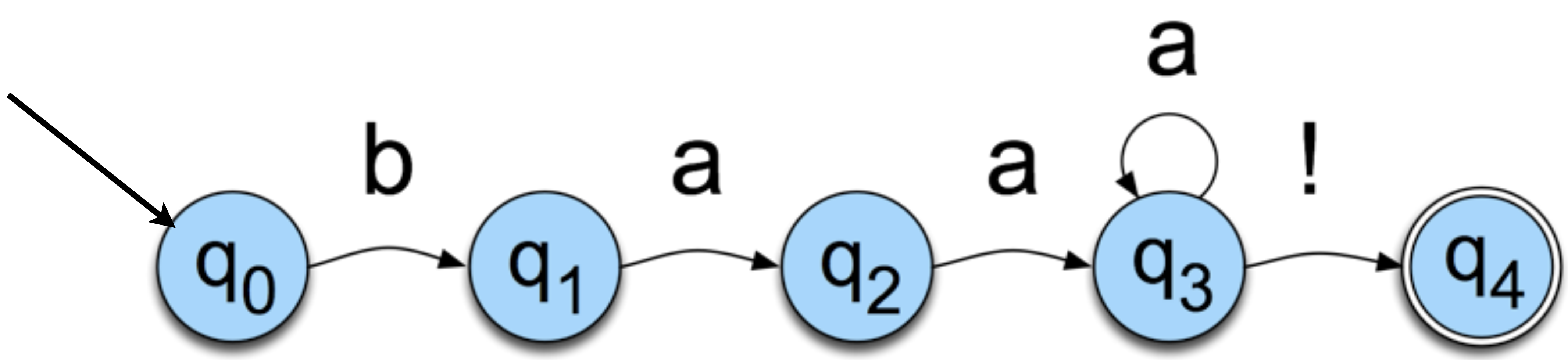
```
% perl -de 42
DB<1> $s = "DOES HE LIKE BEER";
DB<2> print $s;
DOES HE LIKE BEER
DB<3> $s =~ s/(\w+) +(\w+)/\2 \1/;
DB<4> print $s;
HE DOES LIKE BEER
```

Finite State Automata & Regular Expressions

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs and their probabilistic relatives are at the core of much of what we'll do this quarter

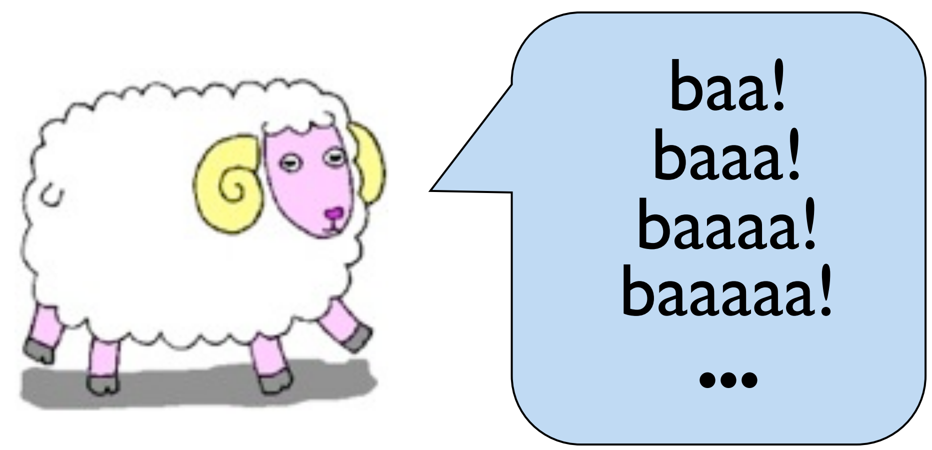
FSAs as Graphs

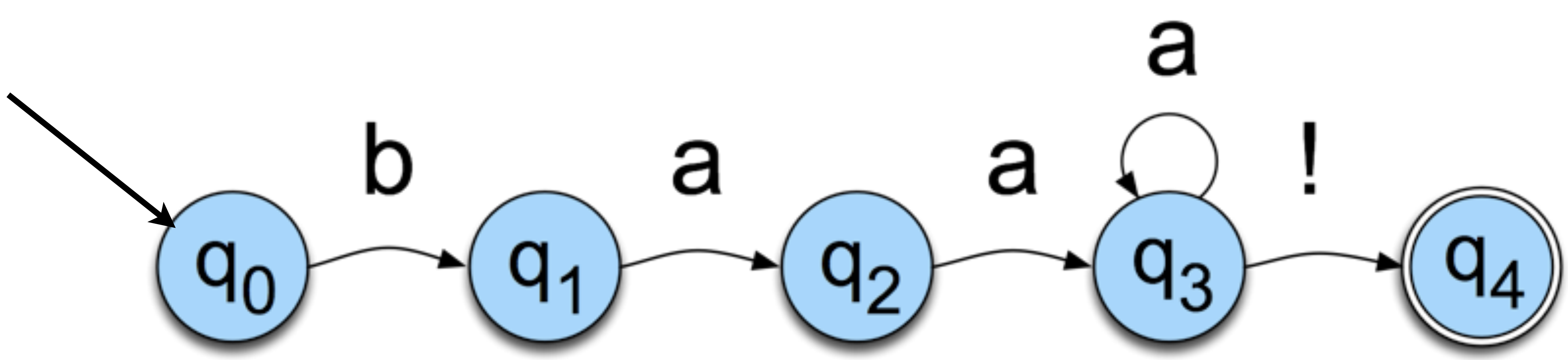
- Let's start with the sheep language
  - /baa+!/



FSAs as Graphs

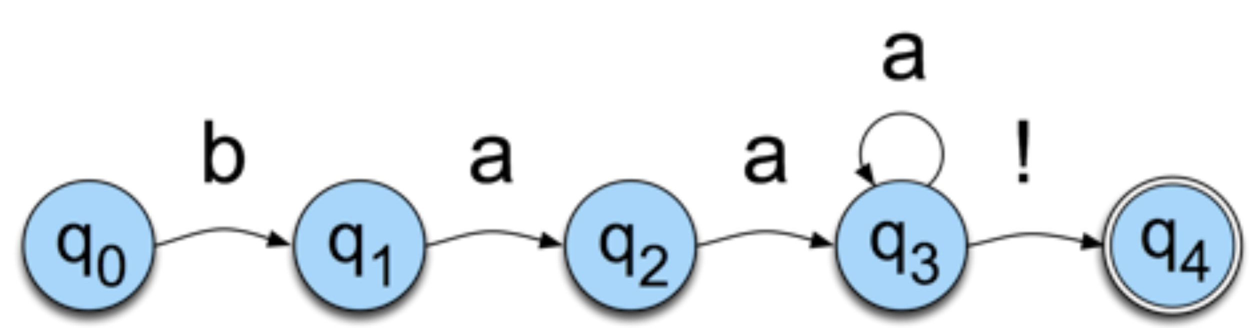
- Let’s start with the sheep language
  - /baa+!/





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



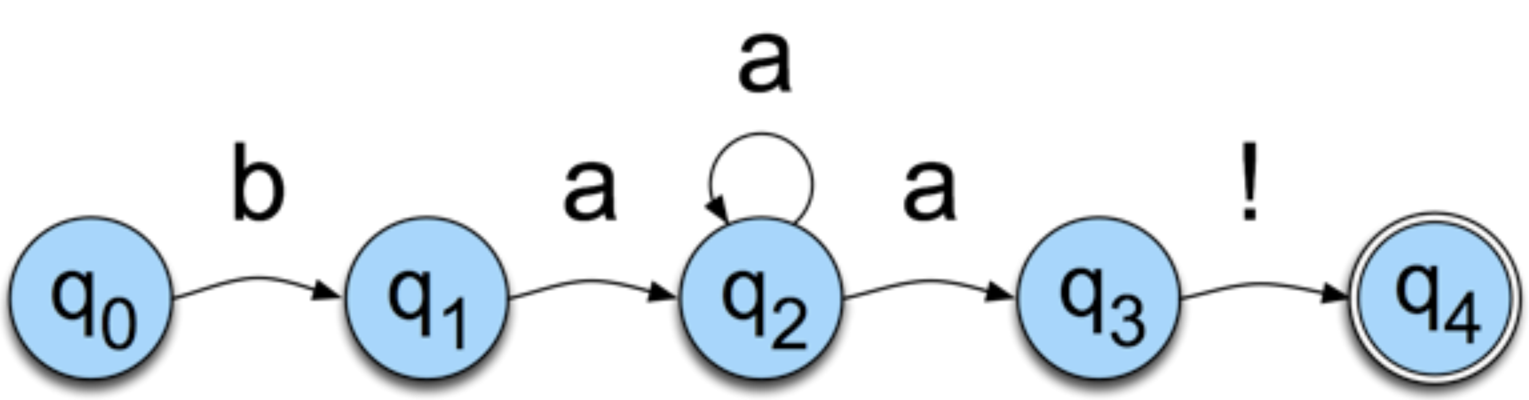
More Formally

- 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 QxΣ to Q

Yet Another View

- The guts of FSAs can be represented as tables

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

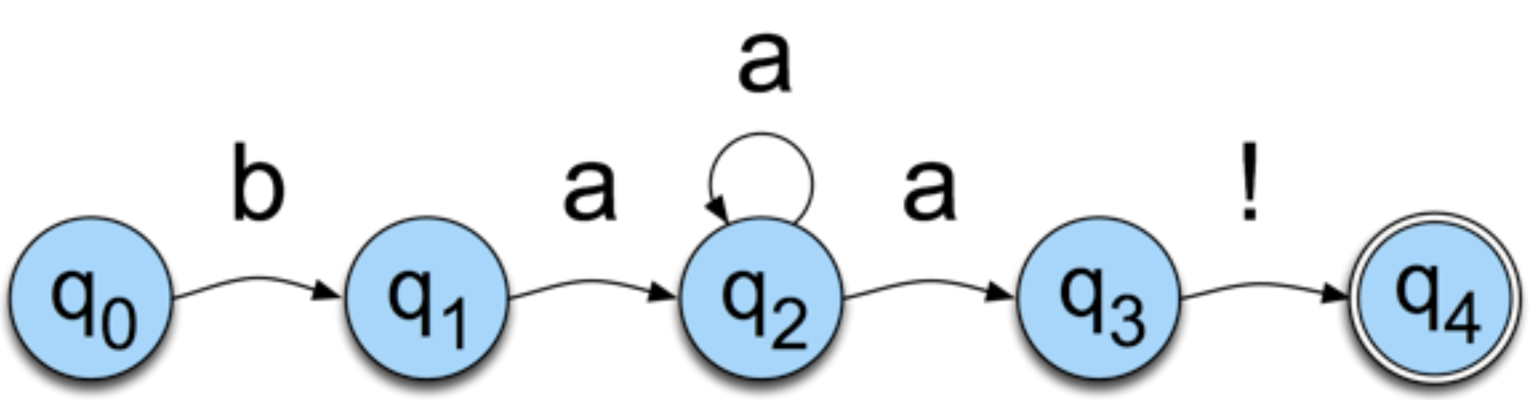


Yet Another View

- The guts of FSAs can be represented as tables

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

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

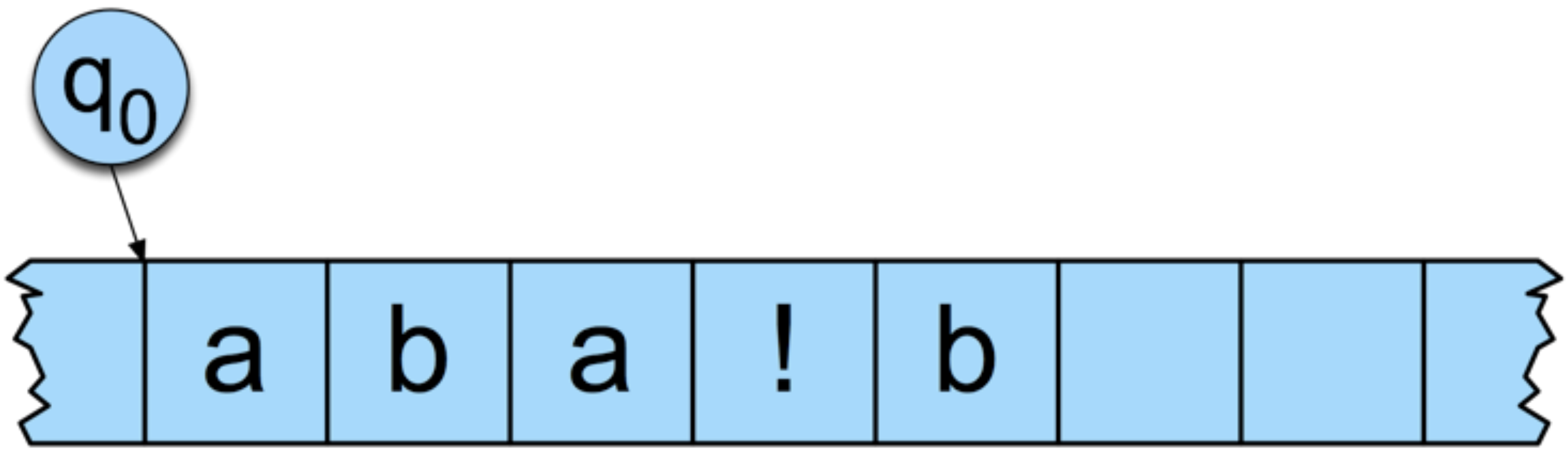


Recognition

- 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

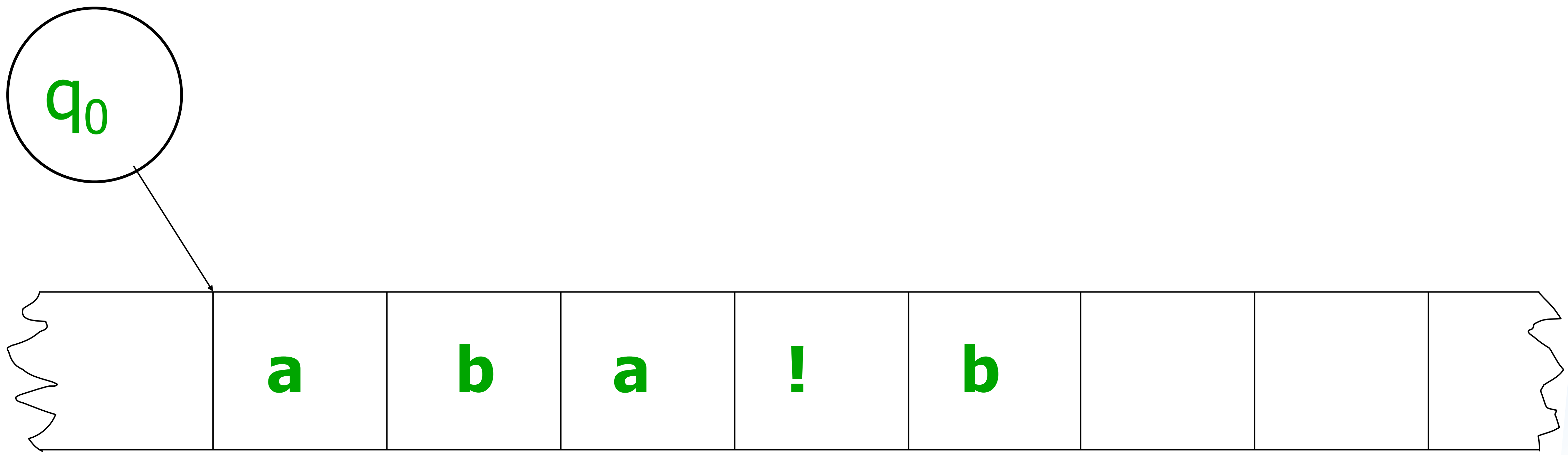
- Traditionally, (Turing's notion) this process is depicted with a tape.



Recognition

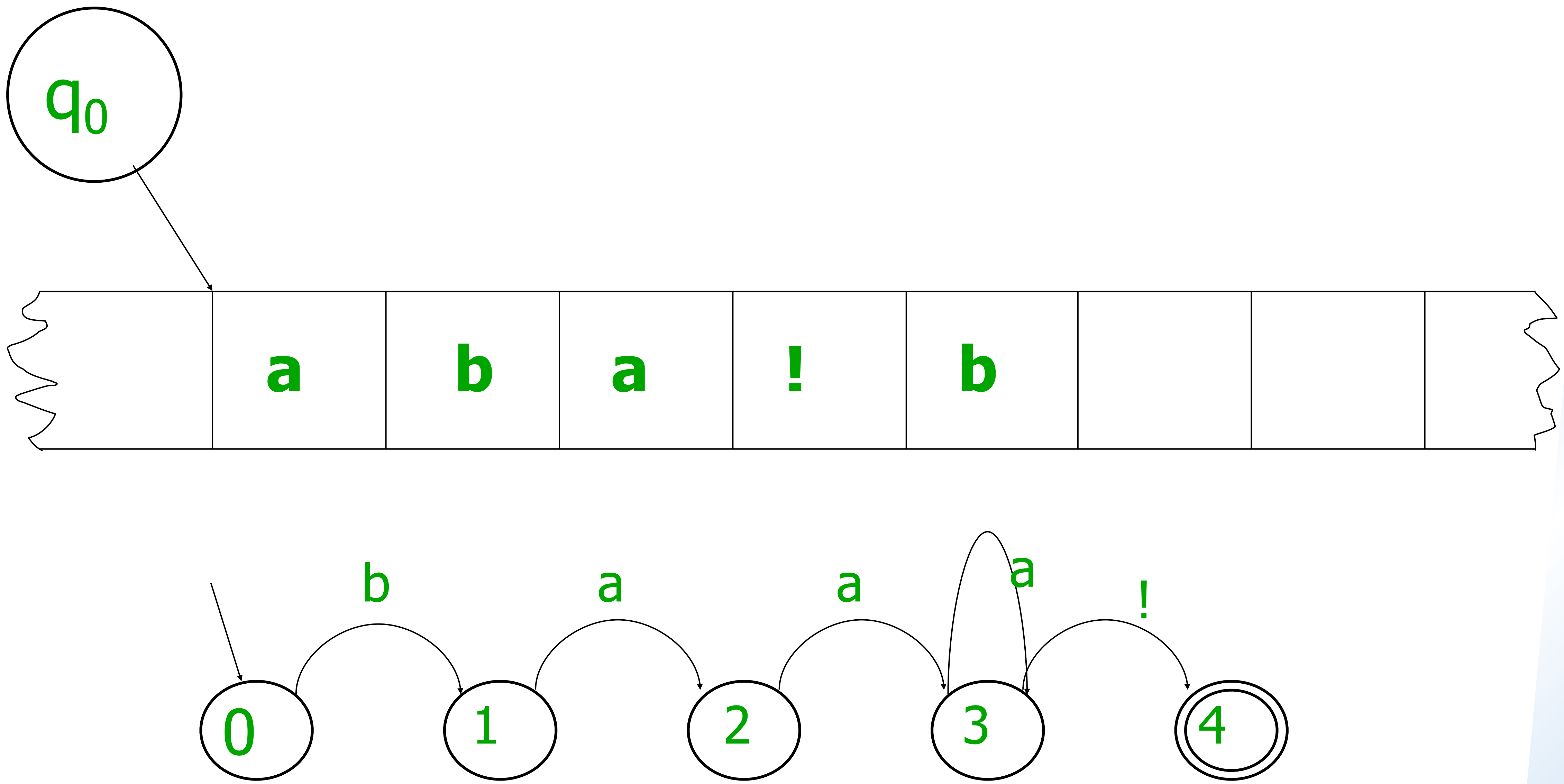
- Start in the start state
- Examine the current input
- Consult the table
- Go to a new state and update the tape pointer.
- Until you run out of tape.

Tracing a Rejection



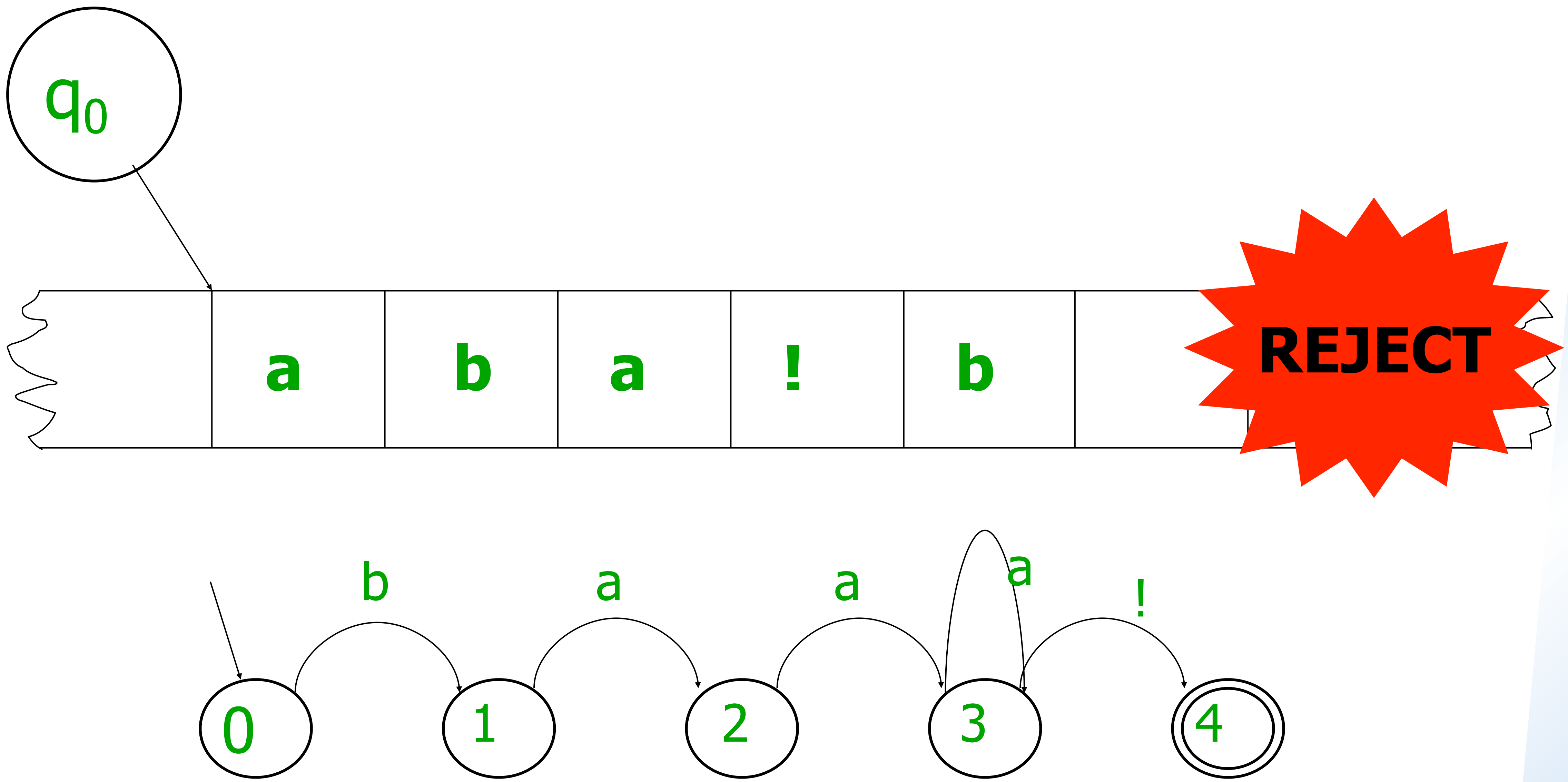
Slide from Dorr/Monz

Tracing a Rejection



Slide from Dorr/Monz

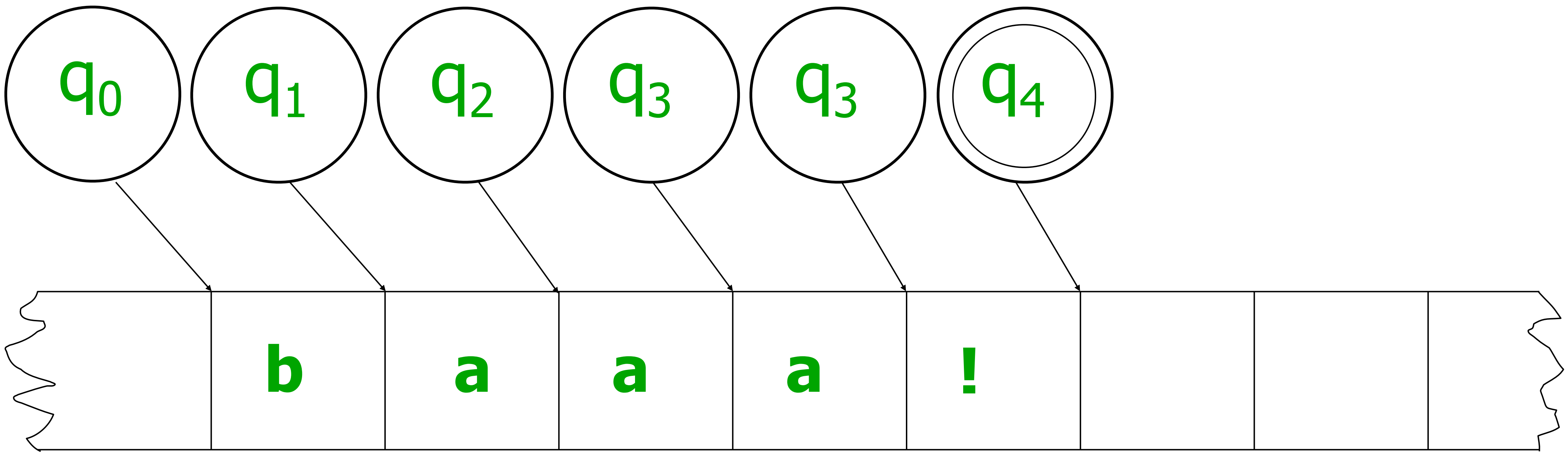
Tracing a Rejection



10/17/11

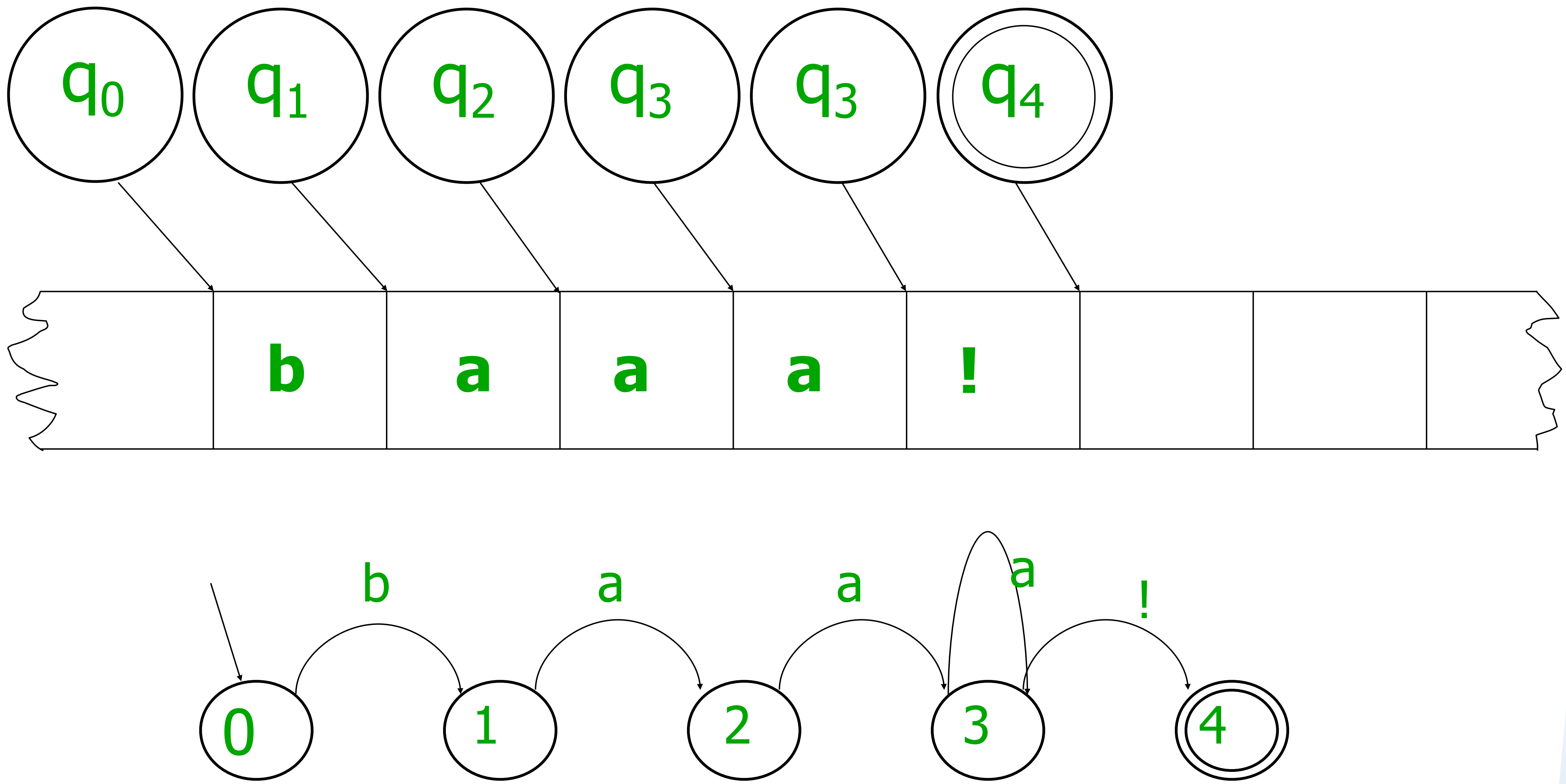
Slide from Dorr/Monz

Tracing an Accept



Slide from Dorr/Monz

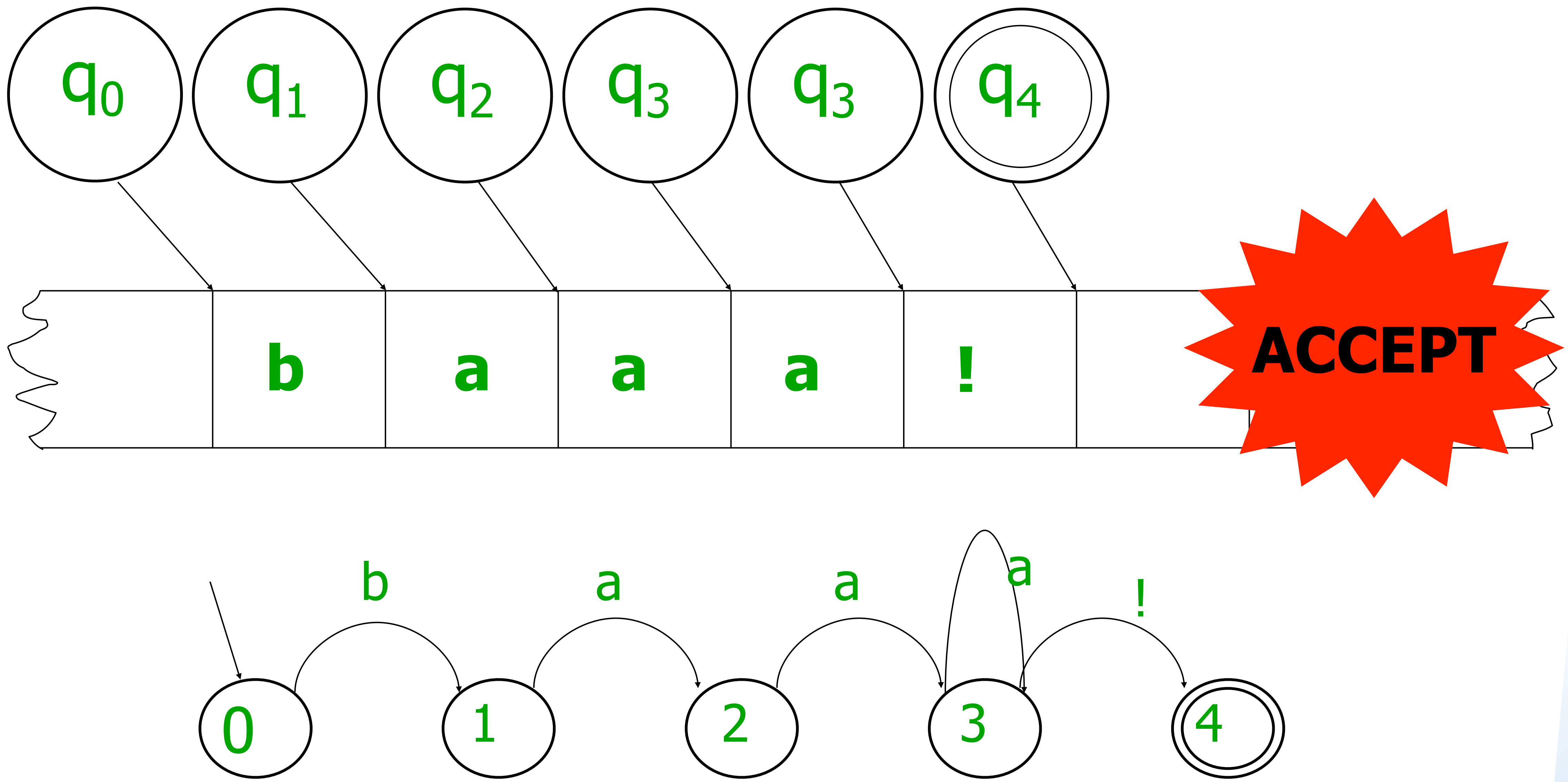
Tracing an Accept



10/17/11

Slide from Dorr/Monz

Tracing an Accept



Slide from Dorr/Monz

Regular expression search

http://www.inf.ed.ac.uk/teaching/courses/il1/2010/labs/2010-10-07/regex.xml

http://www.learn-javascript-tutorial.com/RegularExpressions.cfm#h1.2

Search for the following expressions

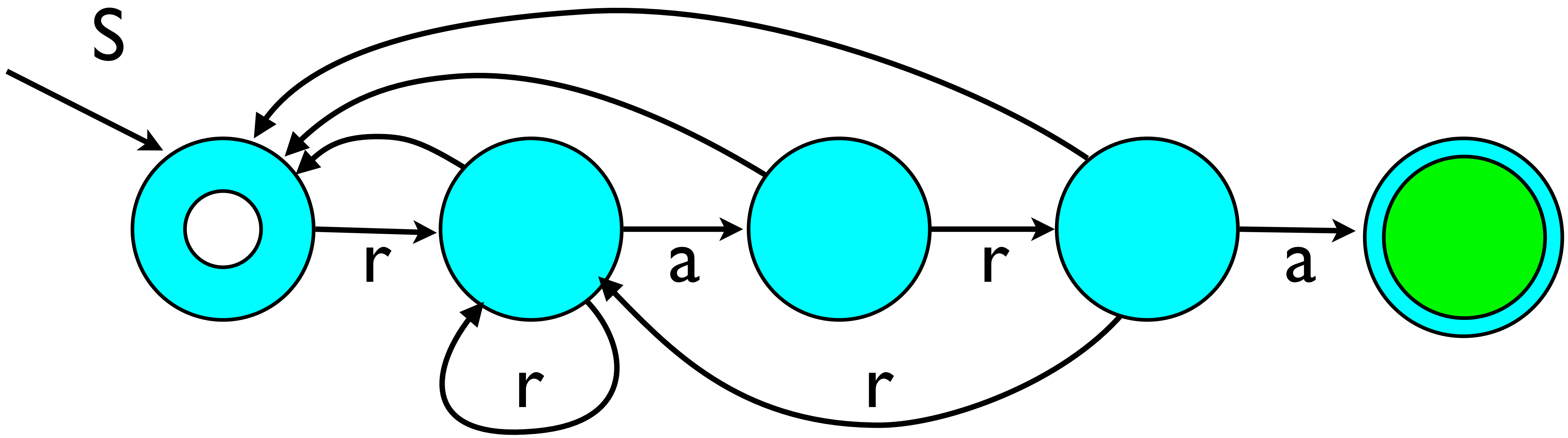
- Alice
- brillig
- m.m
- c..c
- [A-Z][A-Z]+
- J|j
- (J|j)
- \(.*\)
- I.*I
- I.*?I
- I.+I

What does . stand for? (any character)
* is for repetition - zero or more times
[aeiou] is for any vowel

More Examples

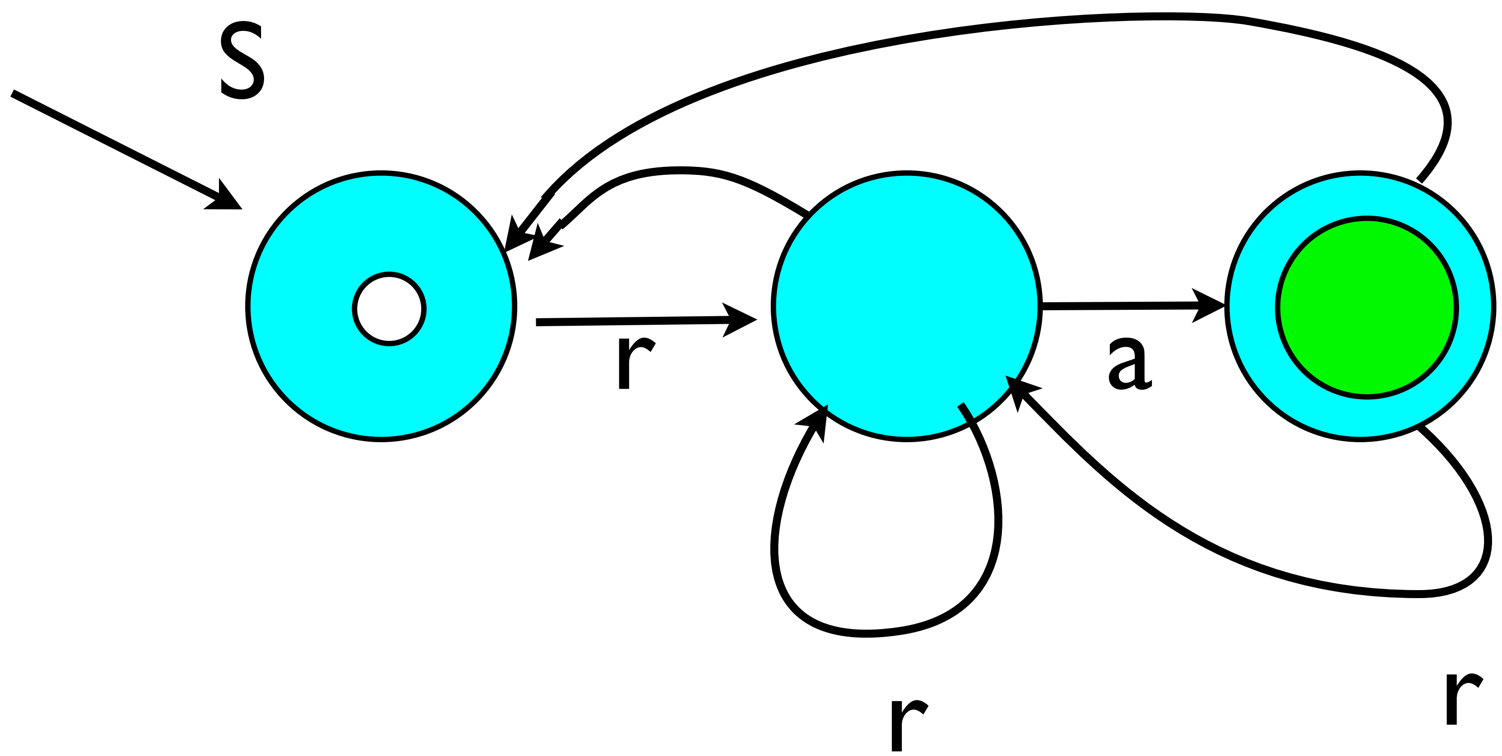
Finite State Automata and Regular Expressions

rara



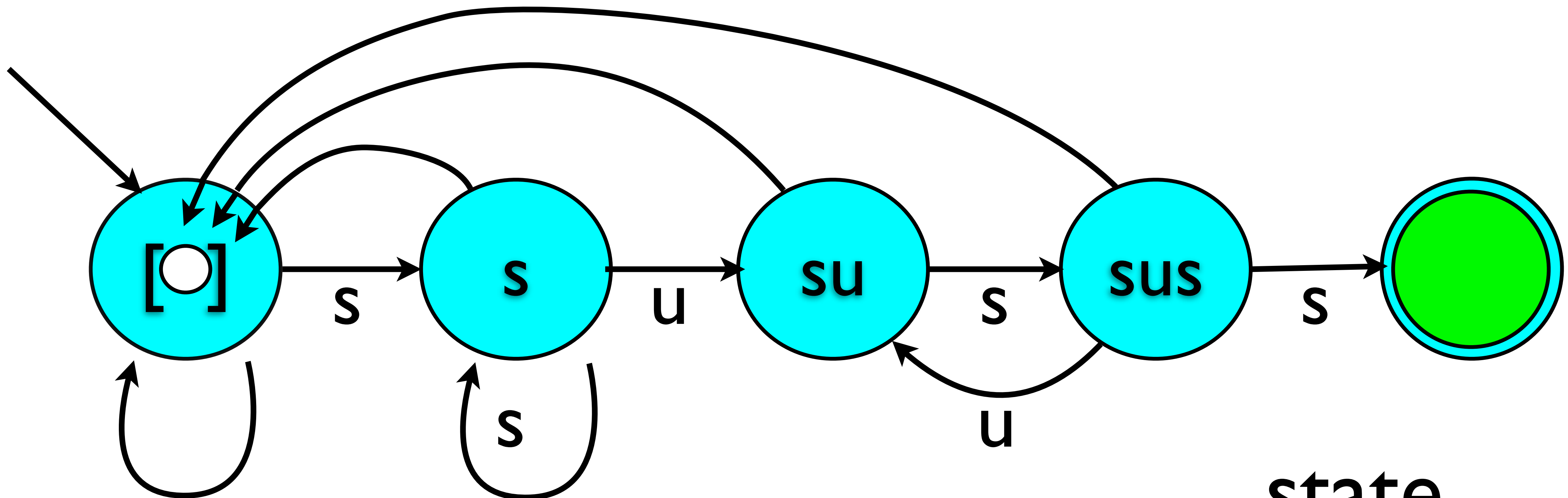
Rara is similar

ra(ra)*



ra(ra)*

suss this?



FSAs can be represented as:

- graphs
- transition tables

state

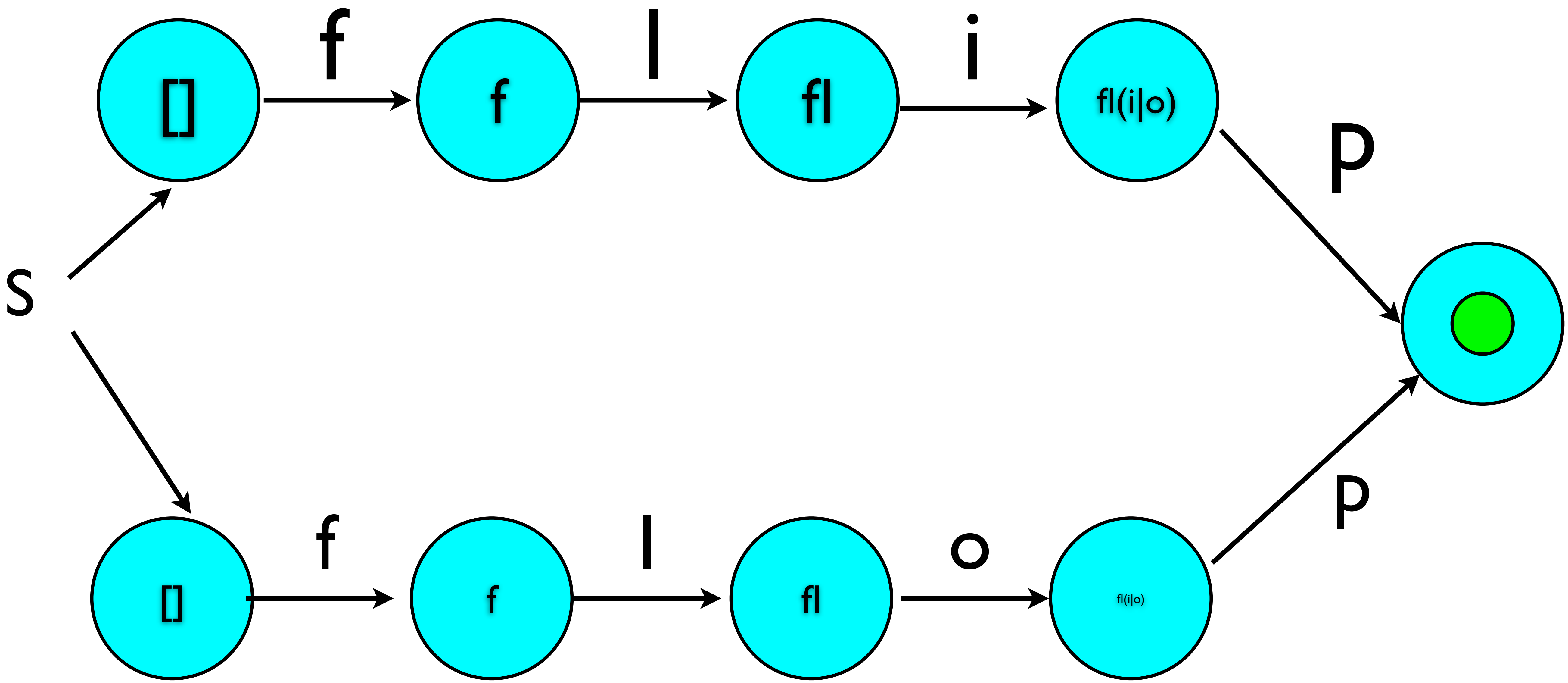
input

	[]	s	su	sus	●
s	s	s	sus	s	
u	[]	su	[]	●	
.	[]	[]	[]	[]	

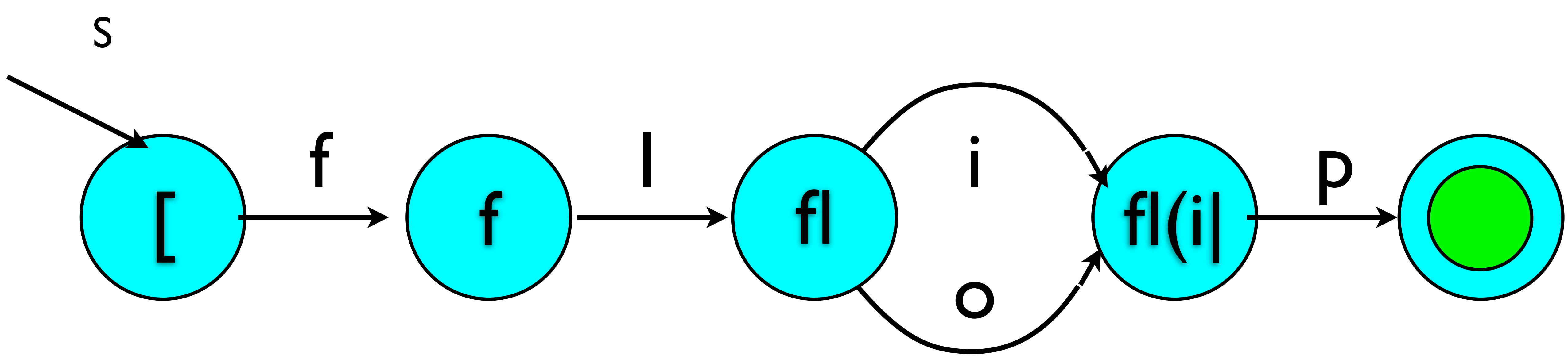
If you're in state s and you're looking at a u, go to state su

Now we try to write finite state machines that will search for regular expressions.

(flip)|(flop)



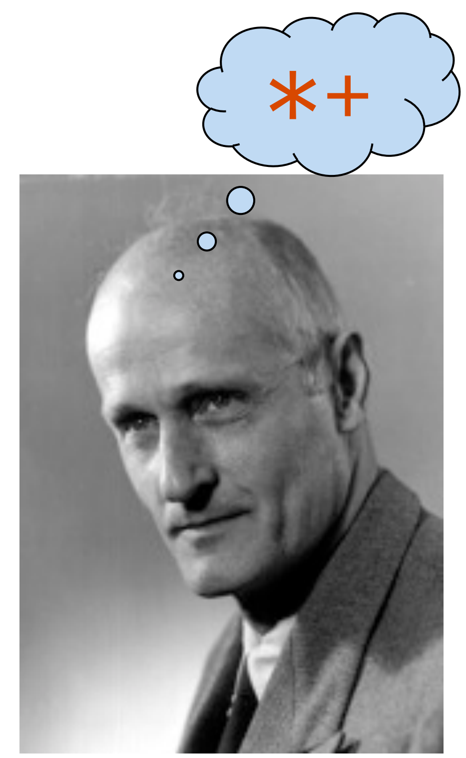
fl(i|o)p



regular expressions

- any character is a regexp
  - matches itself
- if R and S are regexps, so is RS
  - matches
    a match for R followed by a match for S
- if R and S are regexps, so is R|S
  - matches
    any match for R or S (or both)
- if R is a regexp, so is R* (R+)
  - matches
    any sequence of 0 (1) or more matches for R

Kleene *, +



Stephen Cole Kleene

1909-1994