Theory of Programming Languages

Budditha Hettige

Department of Computer Engineering

REGULAR EXPRESSION

Regular expression

- Regular expressions are a powerful string manipulation tool
- All modern languages have similar library packages for regular expressions
- Use regular expressions to:
  - Search a string (`search` and `match`)
  - Replace parts of a string (`sub`)
  - Break strings into smaller pieces (`split`)

Python's
Regular Expression Syntax

- Most characters match themselves
  - The regular expression "test" matches the string `'test'`, and only that string
- [x] matches any *one* of a list of characters
  - "[abc]" matches `'a'`,`'b'`,or `'c'`
- [^x] matches any *one* character that is not included in *x*
  - "[^abc]" matches any single character *except* `'a'`,`'b'`,or `'c'`

Python's Regular Expression Syntax

- "." matches any single character
- Parentheses can be used for grouping
  "(abc)+" matches `'abc'`, `'abcabc'`, `'abcabcabc'`, etc.
- *x*|*y* matches *x* or *y*
  "this|that" matches `'this'` and `'that'`, but not `'thisthat'`.

Python's Regular Expression Syntax

- *x** matches zero or more *x*'s

  "a*" matches `''`, `'a'`, `'aa'`, etc.

- *x*+ matches one or more *x*'s

  "a+" matches `'a'`,`'aa'`,`'aaa'`, etc.

- *x*? matches zero or one *x*'s

  "a?" matches `''` or `'a'`

- *x{m, n}* matches *i* *x*'s, where $m \le i \le n$

  "a{2,3}" matches `'aa'` or `'aaa'`

Python's
Regular Expression Syntax

- "\d" matches any digit; "\D" any non-digit
- "\s" matches any whitespace character; "\S" any non-whitespace character
- "\w" matches any alphanumeric character; "\W" any non-alphanumeric character
- "^" matches the beginning of the string; "$" the end of the string
- "\b" matches a word boundary; "\B" matches a character that is not a word boundary

Basic Regular Expression Patterns

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.

Basic Regular Expression Patterns

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 ^

Basic Regular Expression Patterns

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.

Advanced Operators

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 space	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.

More

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.

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	

Figure 2.8 Some characters that need to be backslashed.

Search and Match

- The two basic functions are **re.search** and **re.match**
  - Search looks for a pattern anywhere in a string
  - Match looks for a match staring at the beginning
- Both return *None* (logical false) if the pattern isn't found and a "match object" instance if it is

```
>>> import re
>>> pat = "a*b”
>>> re.search(pat,"fooaaabcde")
       <_sre.SRE_Match object at 0x809c0>
>>> re.match(pat,"fooaaabcde")
>>>
```

Q: What's a match object?

- A: an instance of the match class with the details of the match result

```
>>> r1 = re.search("a*b","fooaaabcde")
>>> r1.group()  # group returns string
  matched
'aaab'
>>> r1.start()  # index of the match
  start
3
>>> r1.end()    # index of the match end
7
>>> r1.span()   # tuple of (start, end)
(3, 7)
```

What got matched?

- Here’s a pattern to match simple email addresses

  \w+@(\w+\.)+(com|org|net|edu)

```
>>> pat1 = "\w+@(\w+\.)+(com|org|net|edu)"
>>> r1 = re.match(pat,"finin@cs.umbc.edu")
>>> r1.group()
'finin@cs.umbc.edu’
```

- We might want to extract the pattern parts, like the email name and host

What got matched?

- We can put parentheses around groups we want to be able to reference

```
>>> pat2 = "(\w+)@((\w+\.)+(com|org|net|edu))"
>>> r2 = re.match(pat2,"finin@cs.umbc.edu")
>>> r2.group(1)
'finin'
>>> r2.group(2)
'cs.umbc.edu'
>>> r2.groups()
r2.groups()
('finin', 'cs.umbc.edu', 'umbc.', 'edu’)
```

- Note that the ‘groups’ are numbered in a preorder traversal of the forest

What got matched?

- We can ‘label’ the groups as well…

```
>>> pat3
  ="(?P<name>\w+)@(?P<host>(\w+\.)+(com|org|
  net|edu))"
>>> r3 = re.match(pat3,"finin@cs.umbc.edu")
>>> r3.group('name')
'finin'
>>> r3.group('host')
'cs.umbc.edu’
```

- And reference the matching parts by the labels

More re functions

- re.split() is like split but can use patterns

```
>>> re.split("\W+", "This... is a test,
short and sweet, of split().")
['This', 'is', 'a', 'test', 'short',
'and', 'sweet', 'of', 'split', '']
```

- re.sub substitutes one string for a pattern

```
>>> re.sub('(blue|white|red)', 'black', 'blue
  socks and red shoes')
 'black socks and black shoes'
```

- re.findall() finds al matches

```
>>> re.findall("\d+","12 dogs,11 cats, 1
  egg")
['12', '11', '1']
```

Compiling regular expressions

- If you plan to use a re pattern more than once, compile it to a re object
- Python produces a special data structure that speeds up matching

```
>>> capt3 = re.compile(pat3)
>>> cpat3
<_sre.SRE_Pattern object at 0x2d9c0>
>>> r3 =
  cpat3.search("finin@cs.umbc.edu")
>>> r3
<_sre.SRE_Match object at 0x895a0>
>>> r3.group()
'finin@cs.umbc.edu'
```

Pattern object methods

Pattern objects have methods that parallel the re functions (e.g., match, search, split, findall, sub), e.g.:

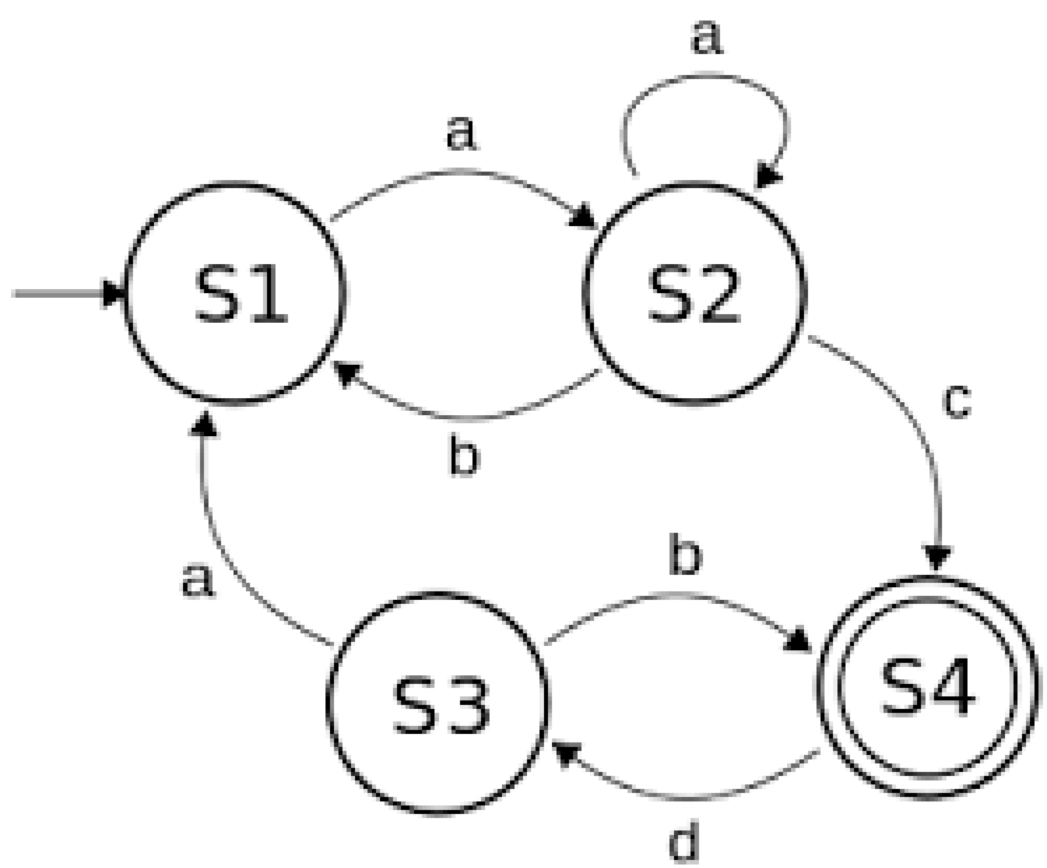
```
>>> p1 = re.compile("\w+@\w+\.+com|org|net|edu")
>>> p1.match("steve@apple.com").group(0)
'steve@apple.com'
>>> p1.search(”Email steve@apple.com today.").group(0)
'steve@apple.com’
>>> p1.findall("Email steve@apple.com and bill@msft.com now.")
['steve@apple.com', 'bill@msft.com’]
>>> p2 = re.compile("[.?!]+\s+")
>>> p2.split("Tired? Go to bed!   Now!! ")
['Tired', 'Go to bed', 'Now', ’ ']
```

email address

sentence boundary

Exercise

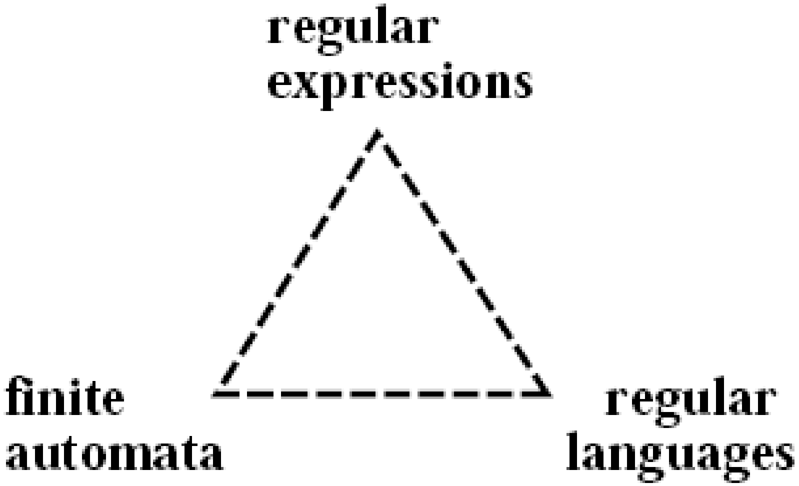
- Write regular expressions for the following languages
  - the set of all alphabetic strings.
  - the set of all lowercase alphabetic strings ending in a *b*.
  - the set of all strings with two consecutive repeated words
- **Example**
  ‘Humbert Humbert’ and ‘the the’ but not ‘the bug’ or ‘the big bug’).



FINITE-STATE AUTOMATA

FSA

- Any regular expression can be implemented as a finite-state automaton
- Both regular expressions and finite-state automata can be used to described regular languages

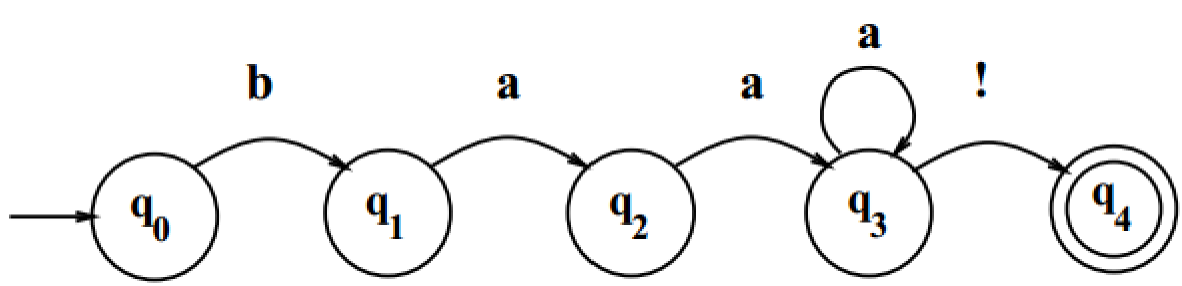


Using an FSA to Recognize

After a while, with the parrot's help, the Doctor got to learn the language of the animals so well that he could talk to them himself and understand everything they said.

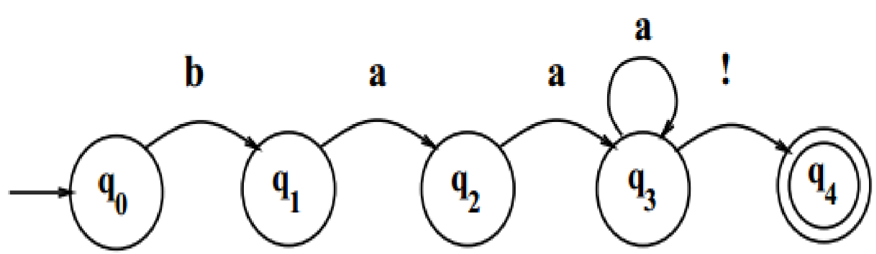
Hugh Lofting, The Story of Doctor Dolittle

baa!
baaa!
baaaa!
baaaaa!
baaaaaa!
. . .

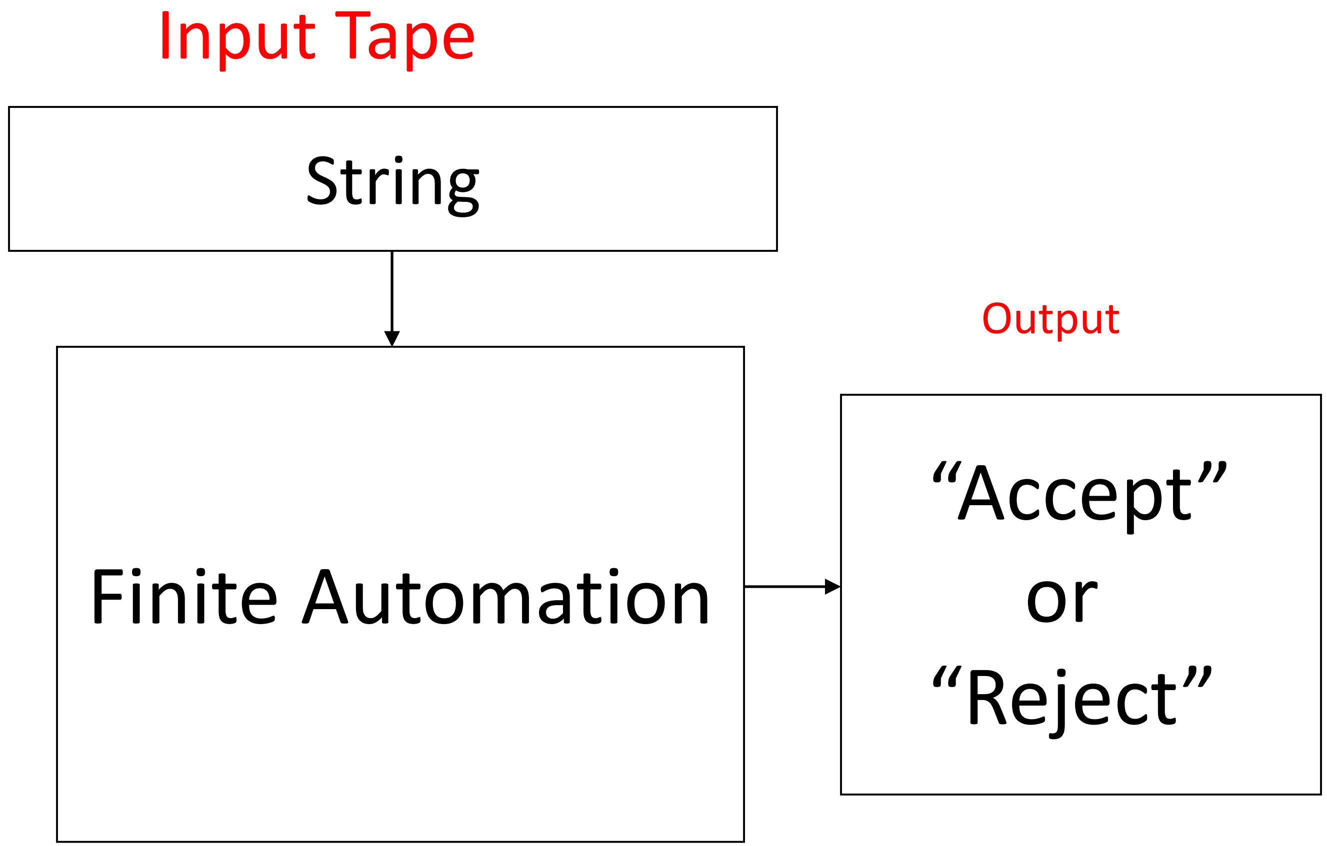


Automaton

- automaton (**finite automaton**, **finite-state automaton**, or **FSA**) recognizes a set of strings
- **Example**
  - Automaton has number of states
  - Start State
  - End State
  - Accepting stat
  - transitions

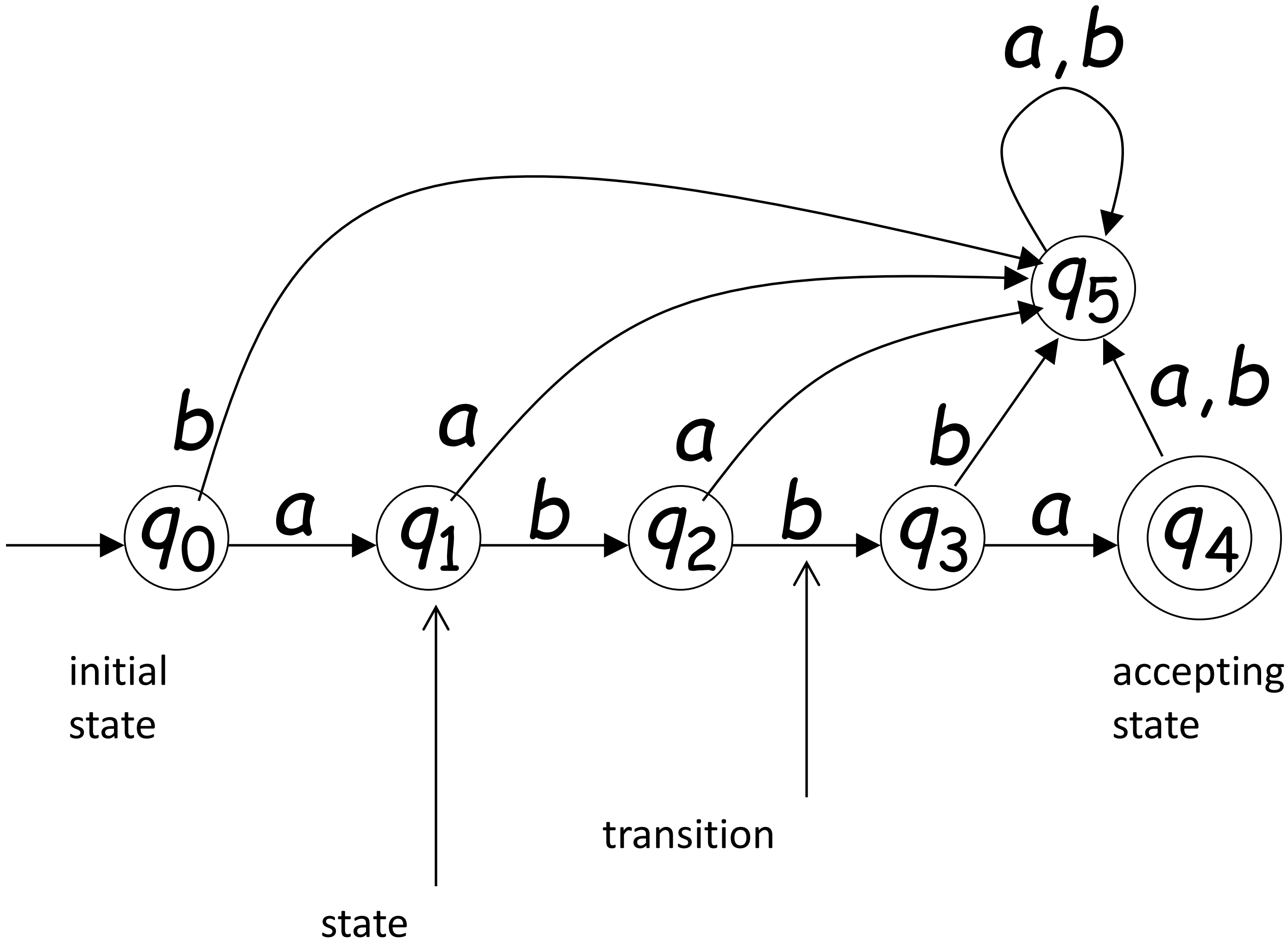


Deterministic Finite Automaton (DFA)

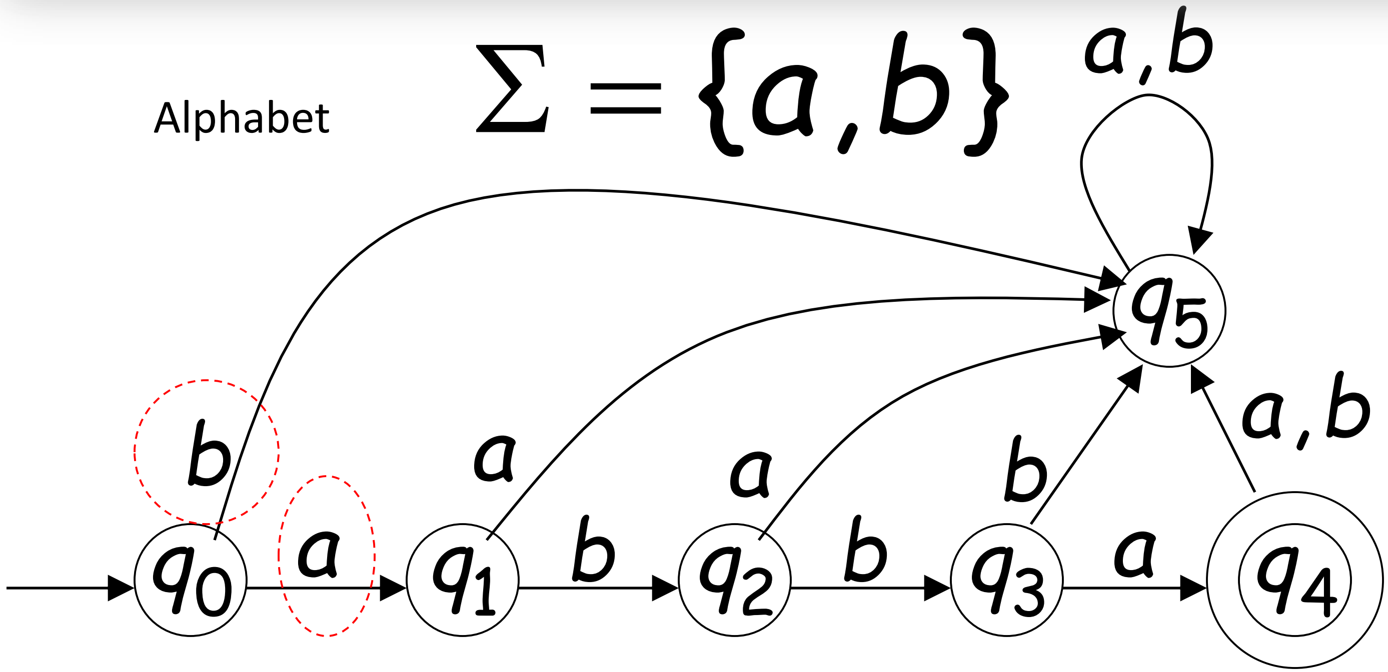


Transition Graph

-

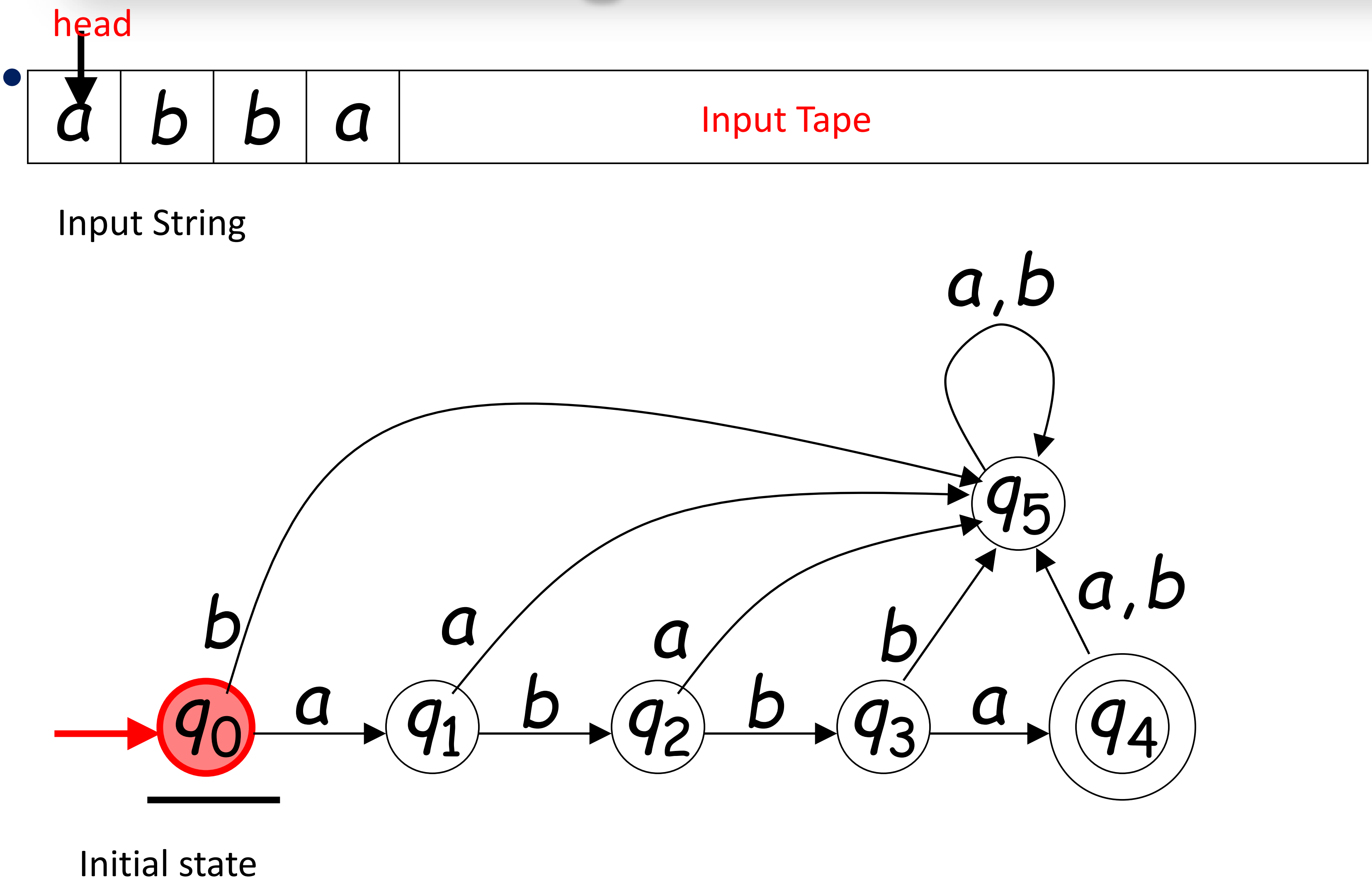


DFA

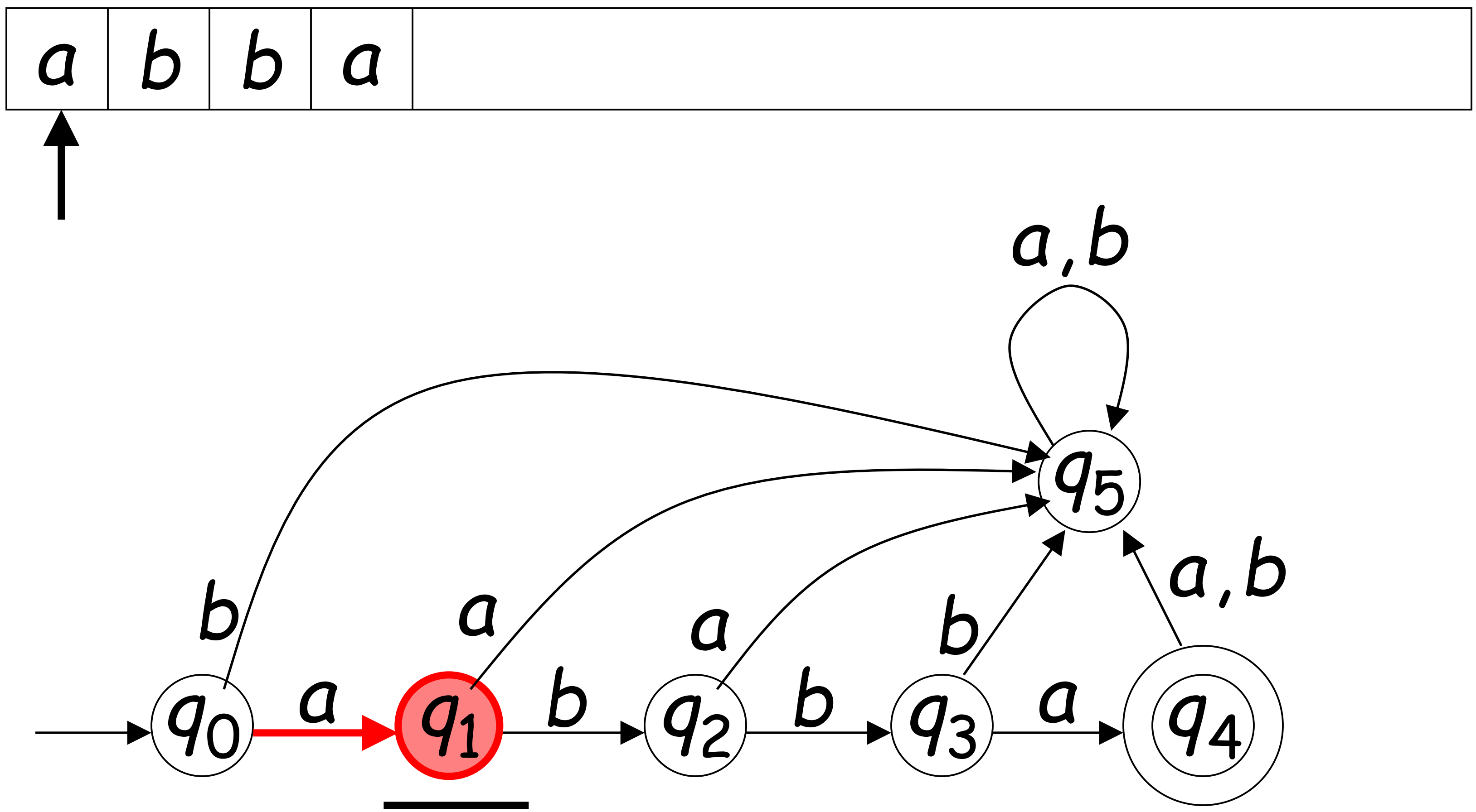


For every state, there is a transition for every symbol in the alphabet

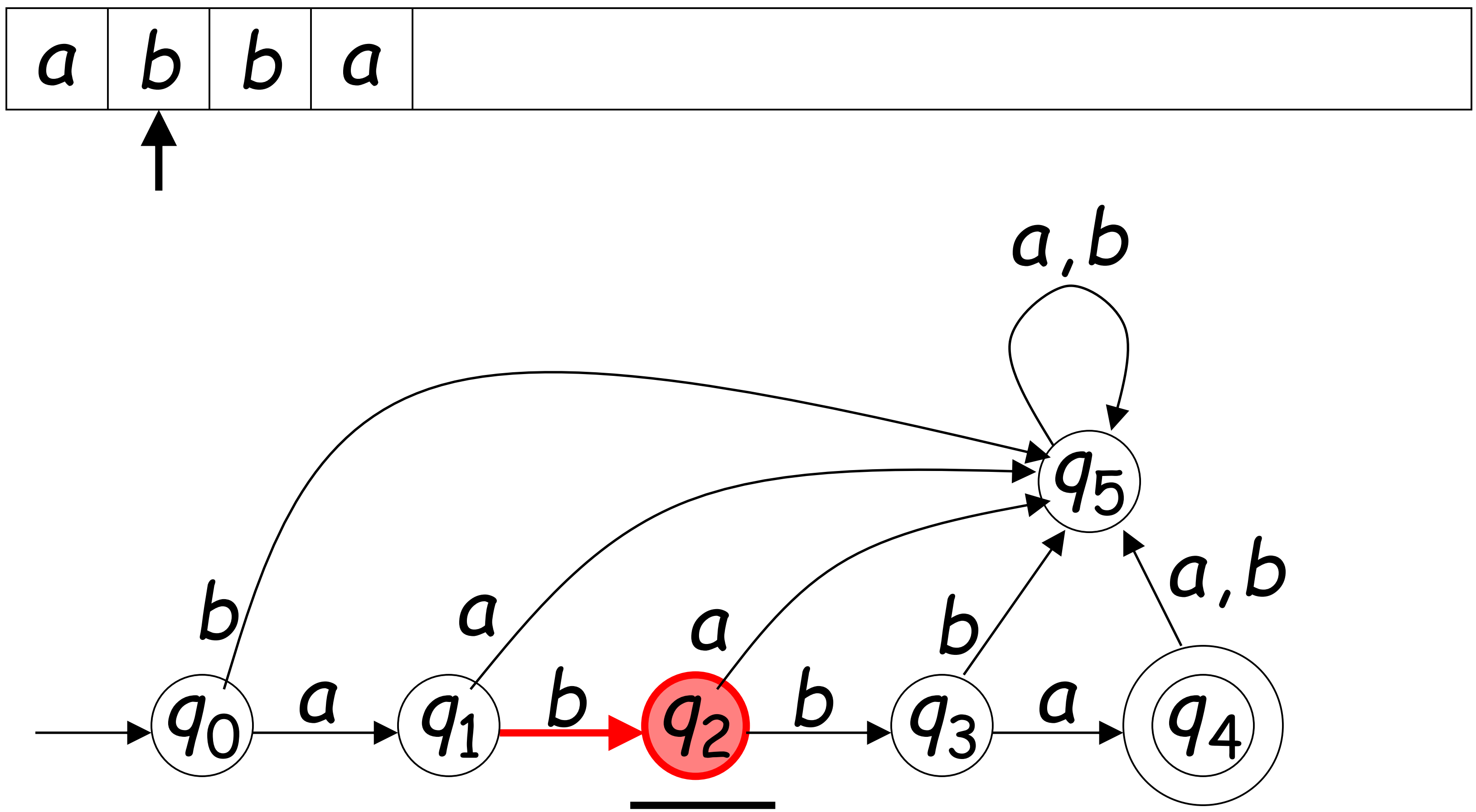
Initial Configuration



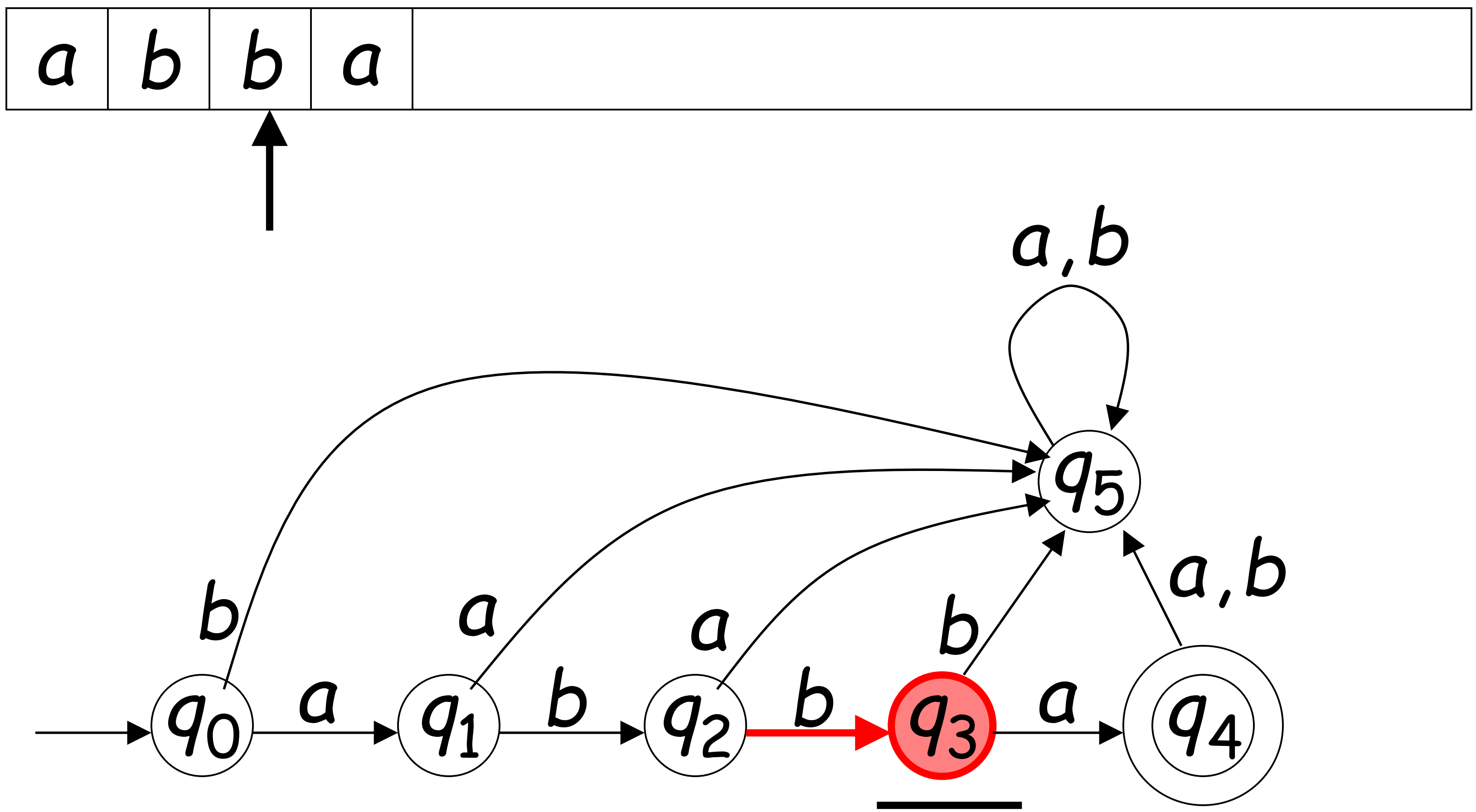
Scanning the Input



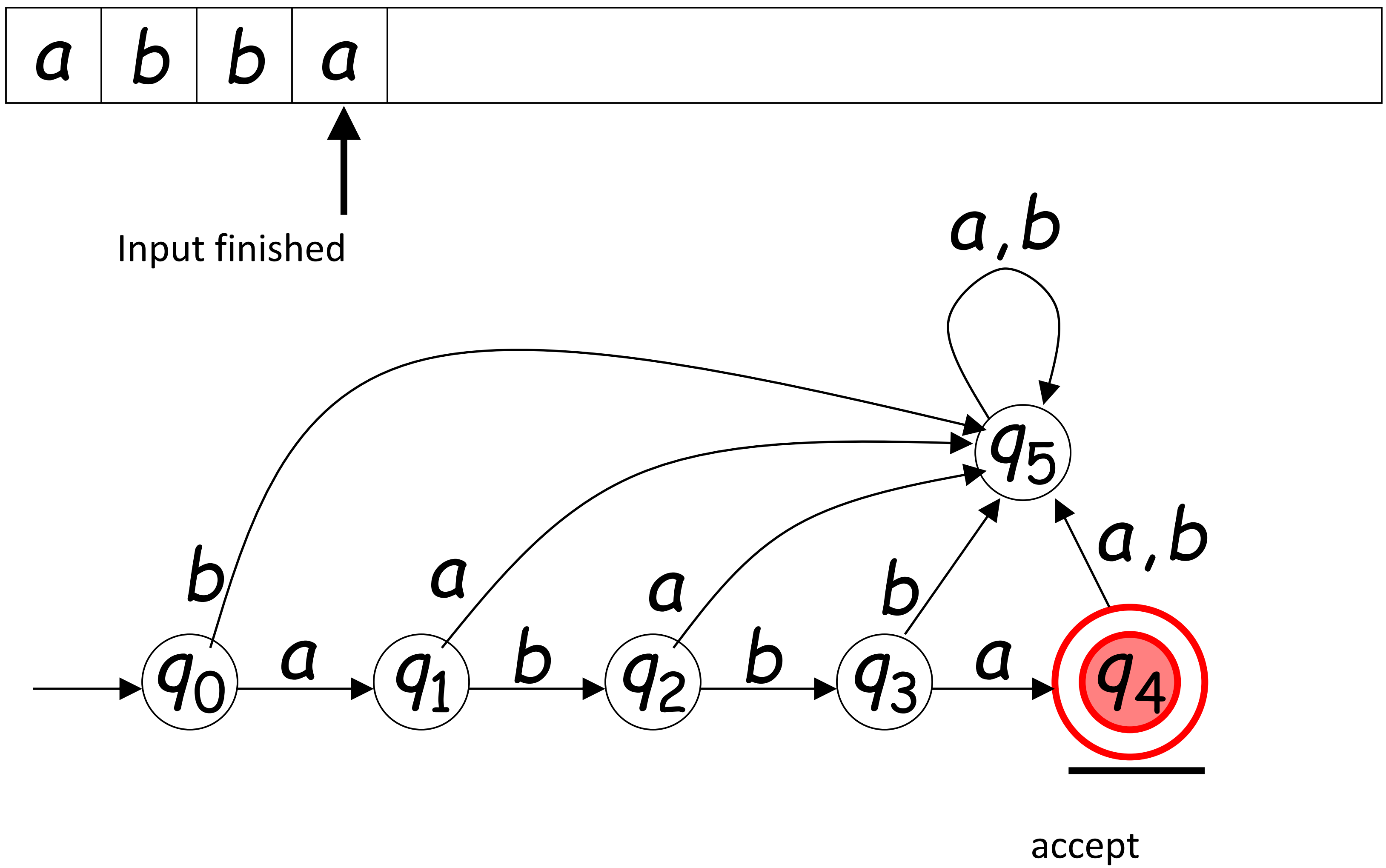
Scanning the Input



Scanning the Input



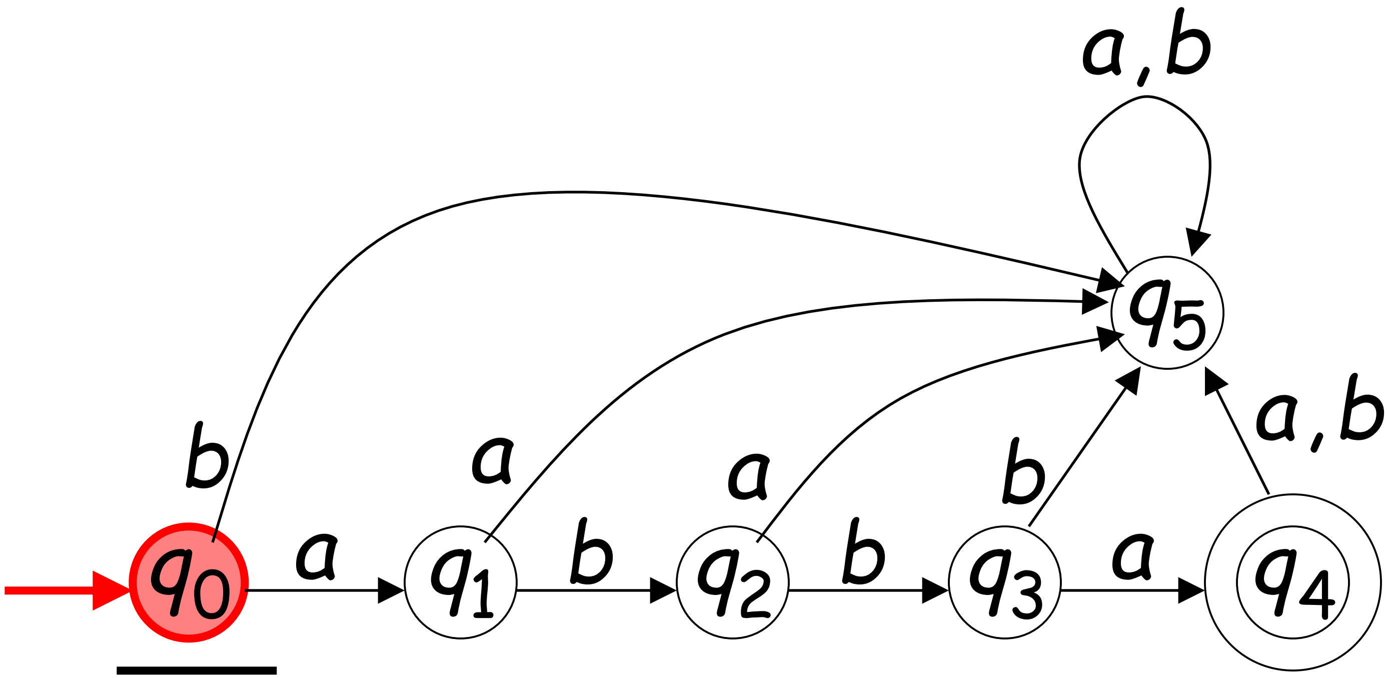
Scanning the Input



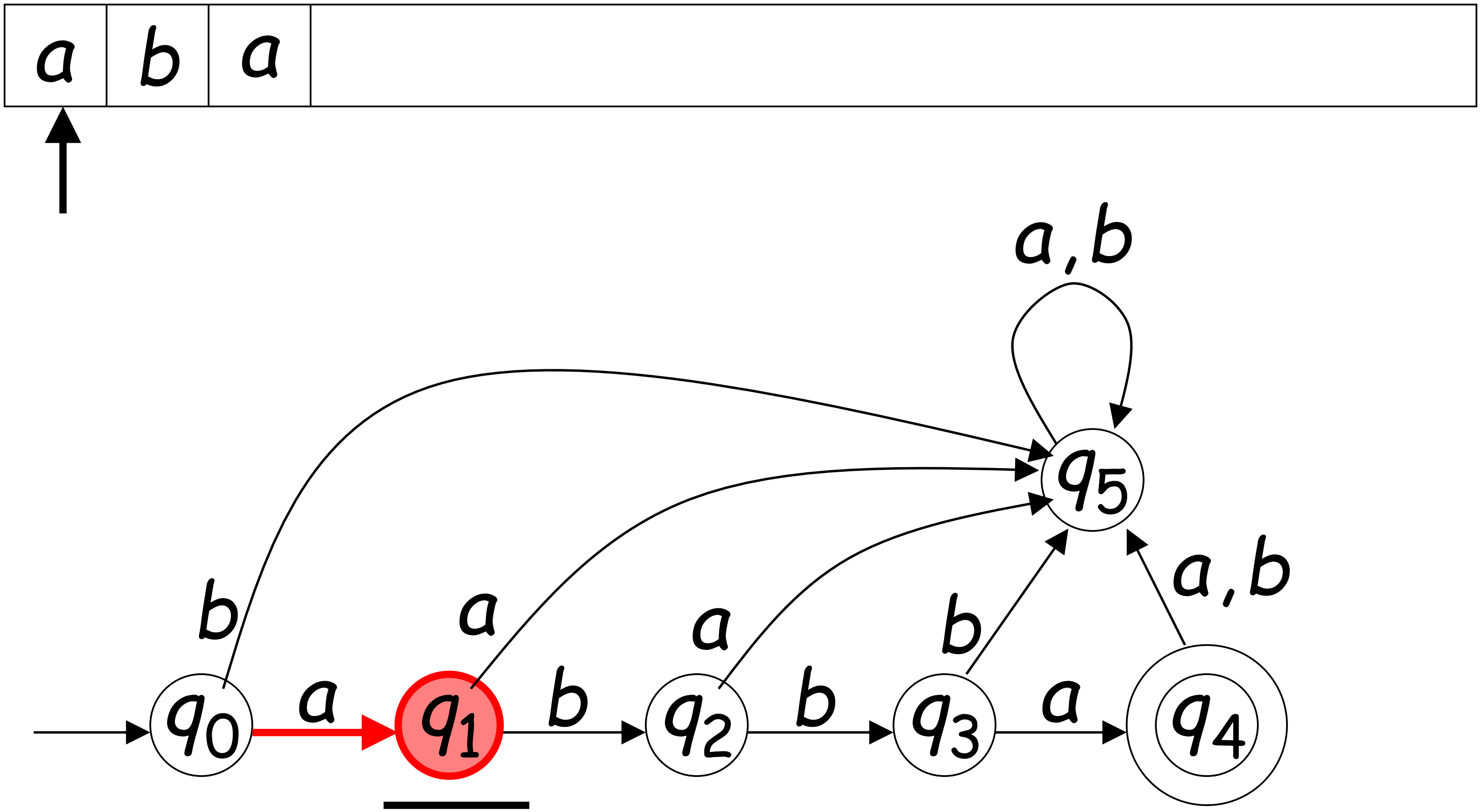
A Rejection Case

a	b	a	

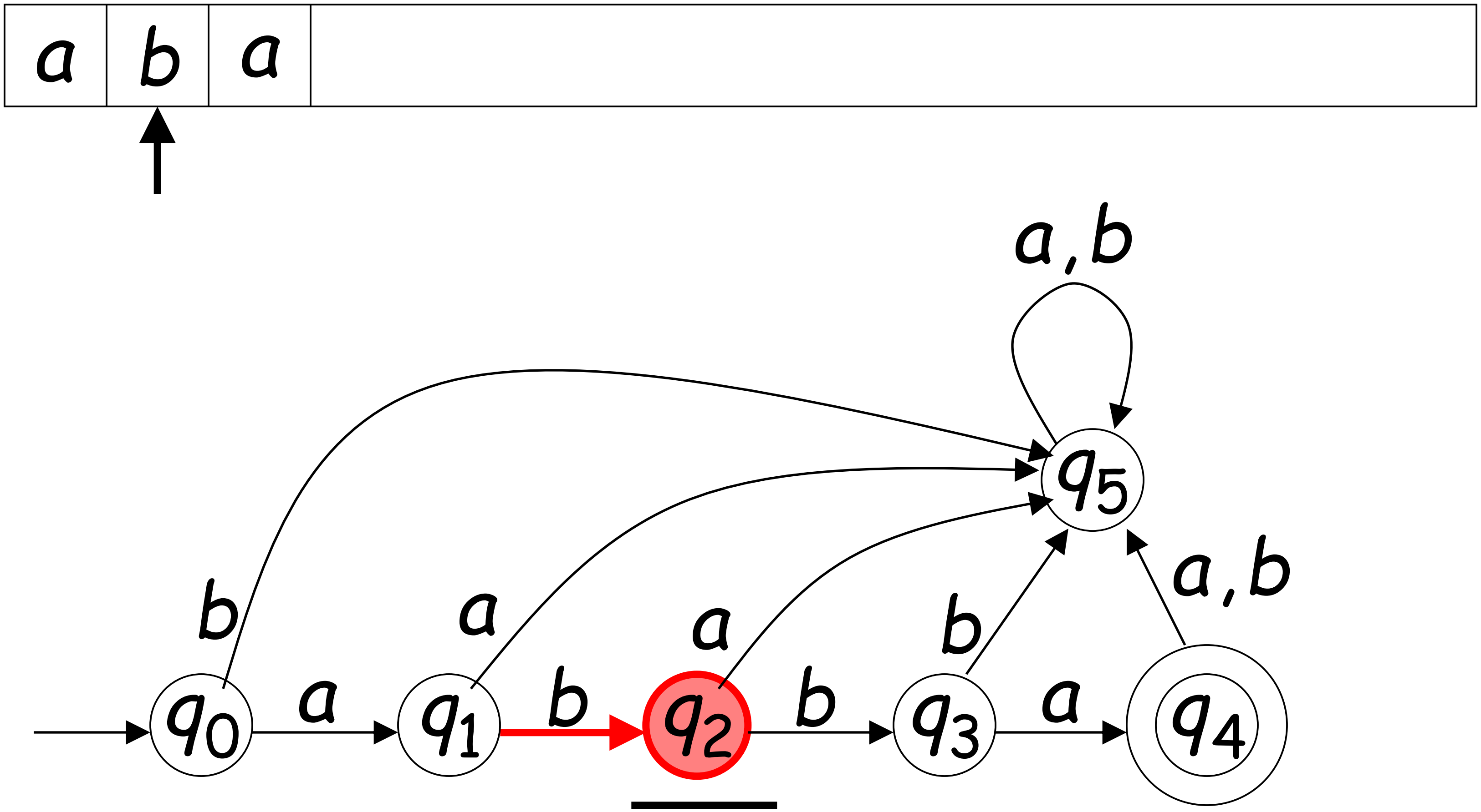
Input String



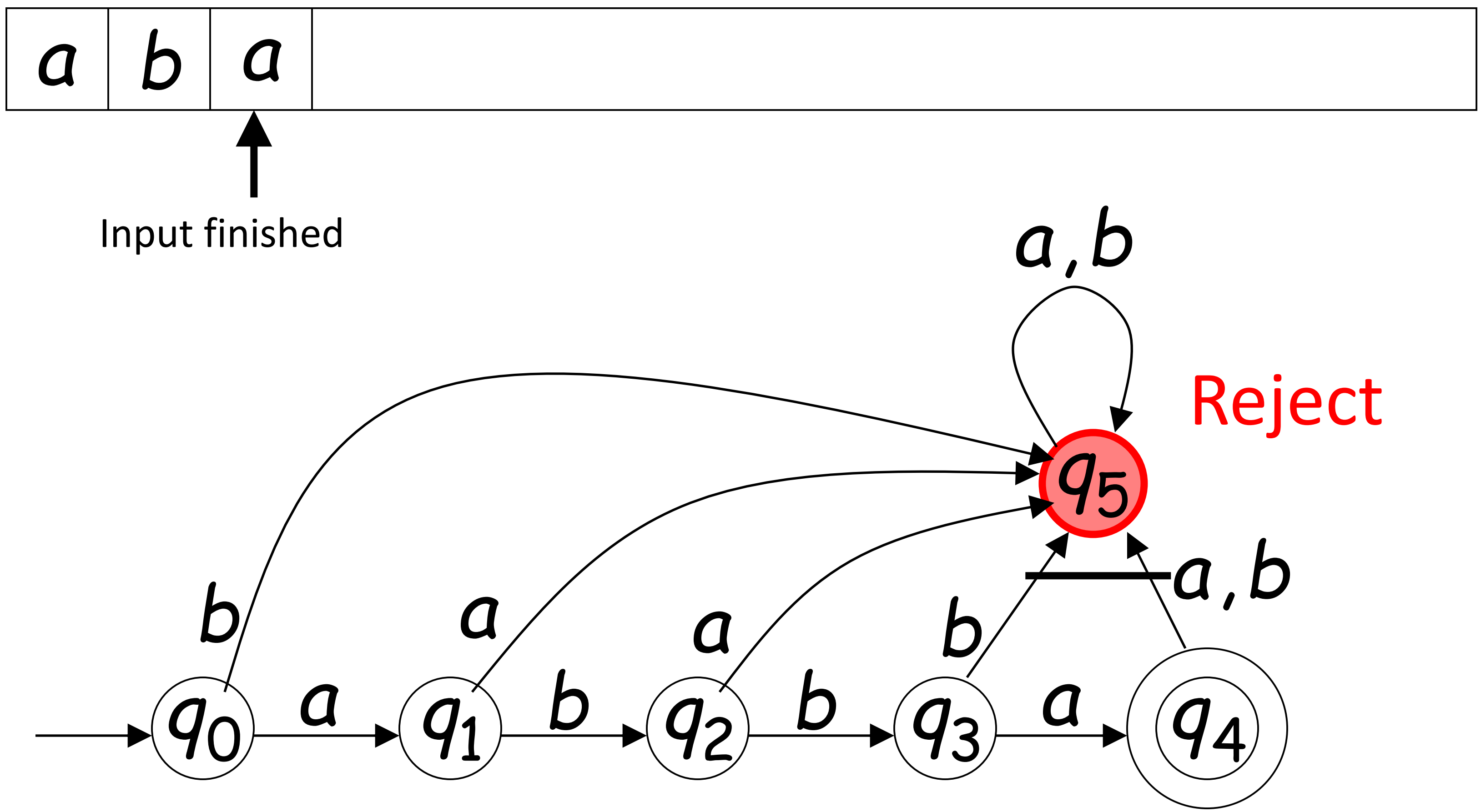
A Rejection Case



A Rejection Case



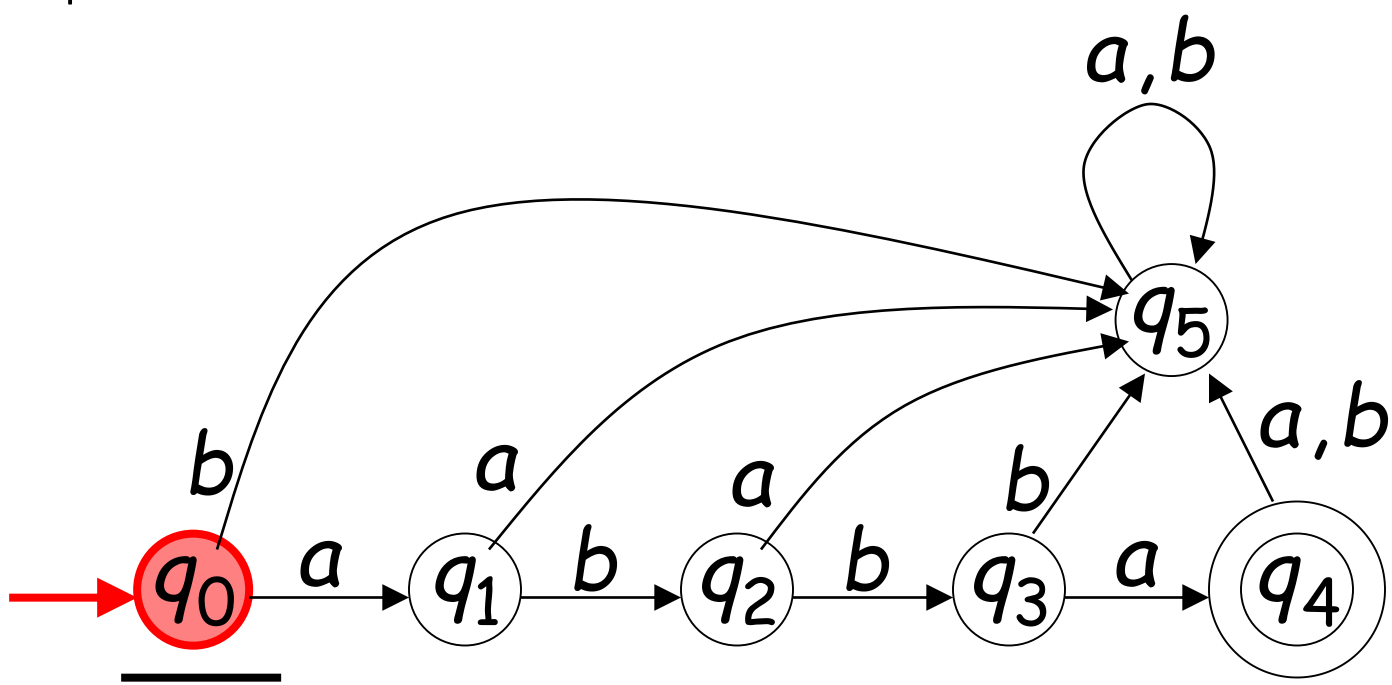
A Rejection Case



Another Rejection Case

(λ) Tape is empty

Input Finished

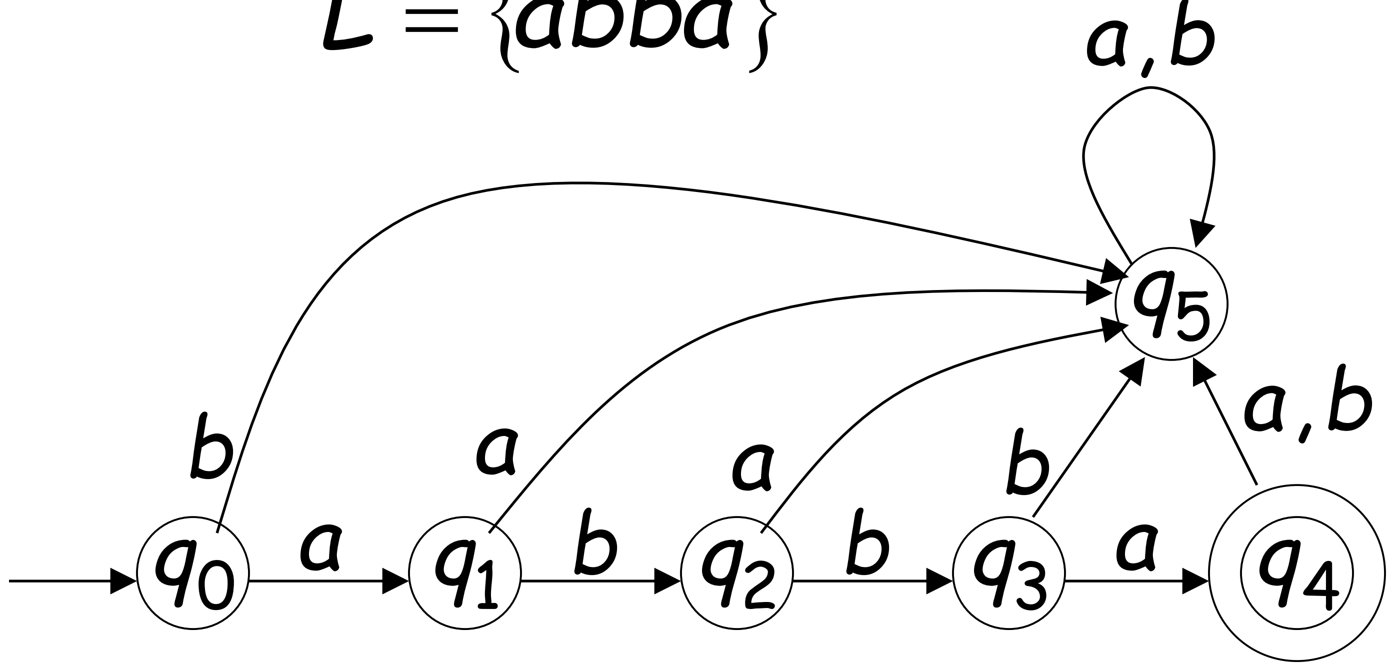


reject

Accepted

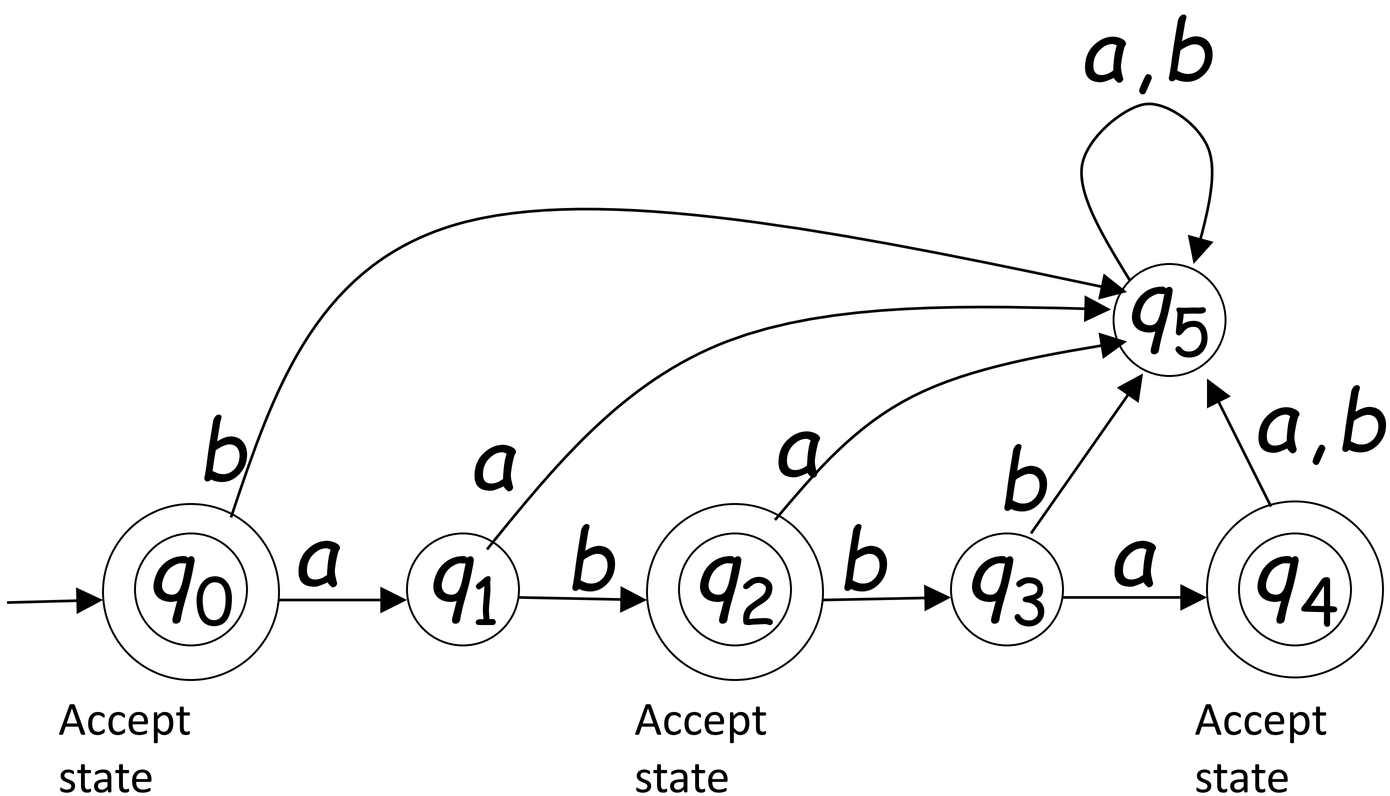
Language Accepted:

$$L = \{abba\}$$



Another Example

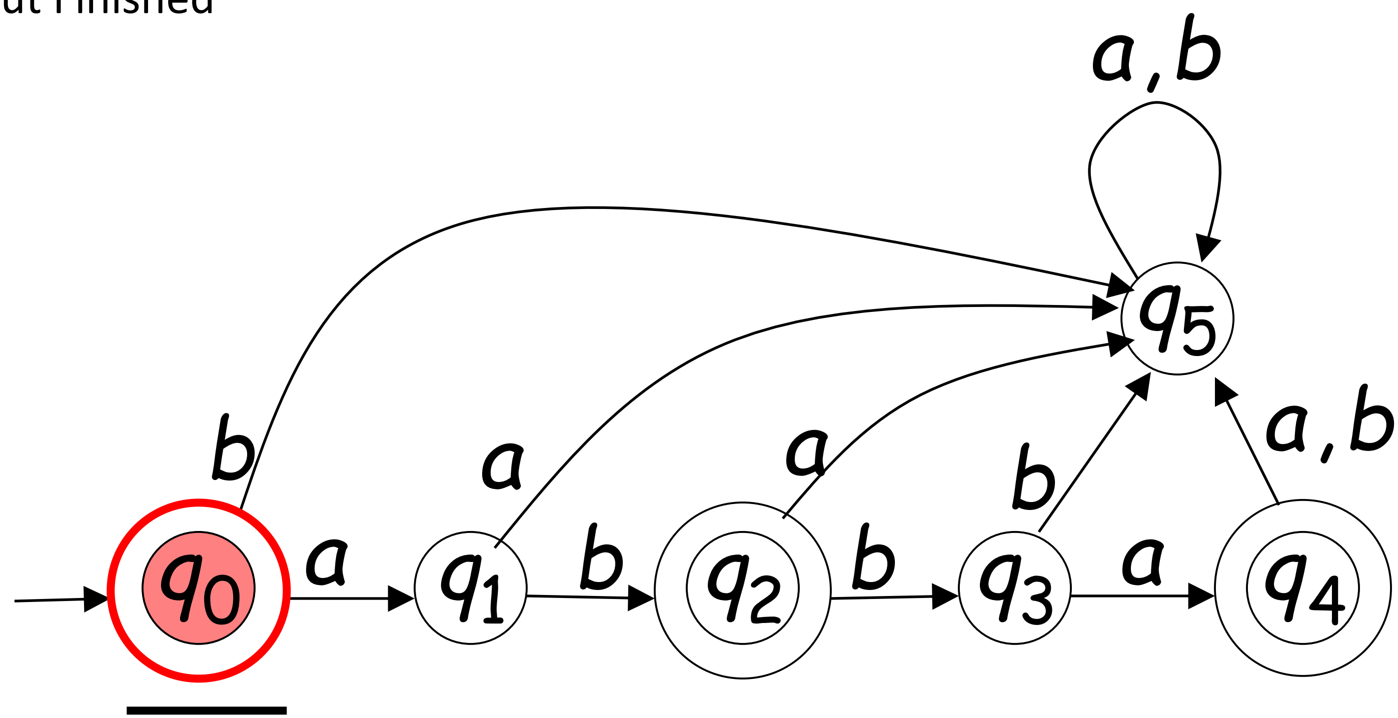
$$L = \{\lambda, ab, abba\}$$



Empty Tape

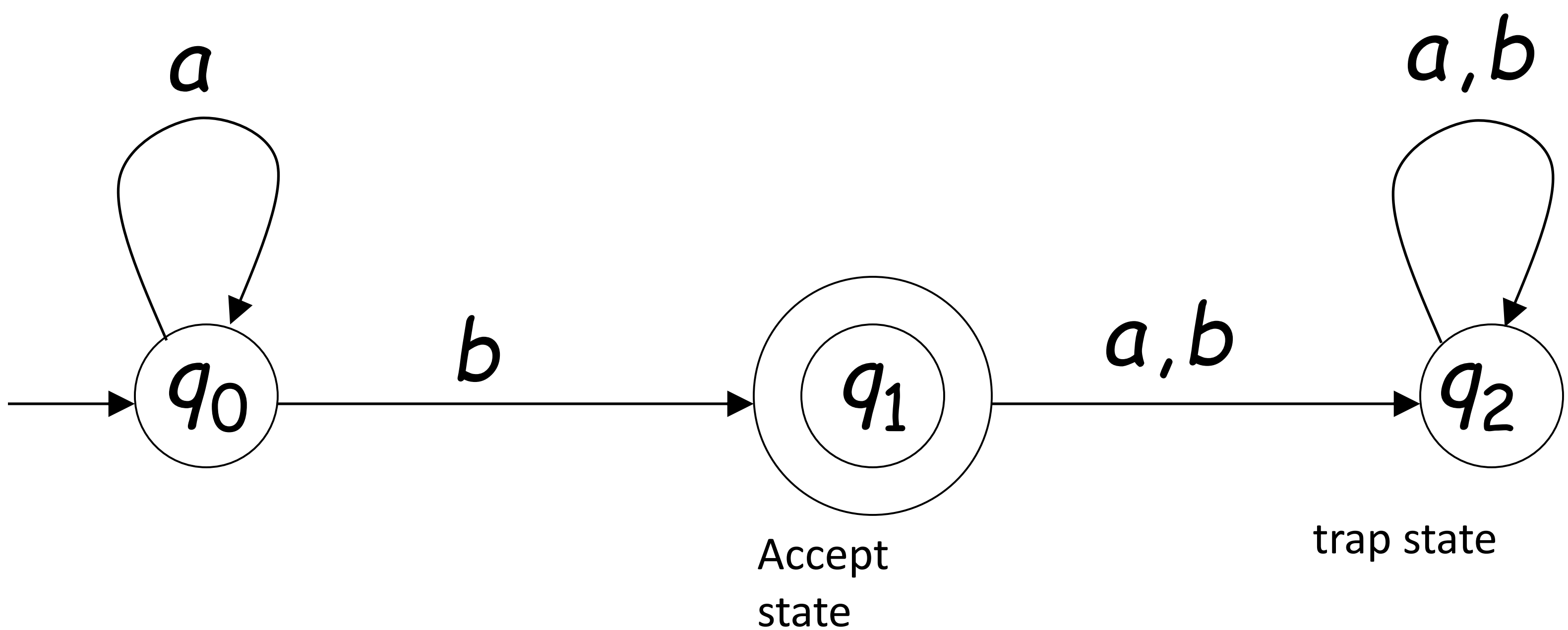
(λ)

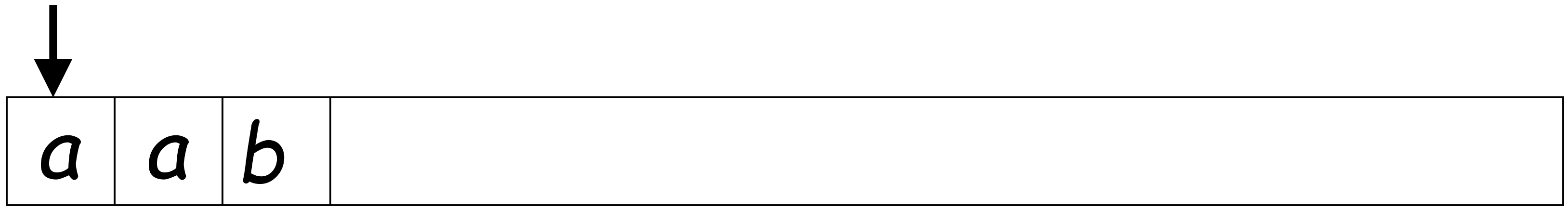
Input Finished



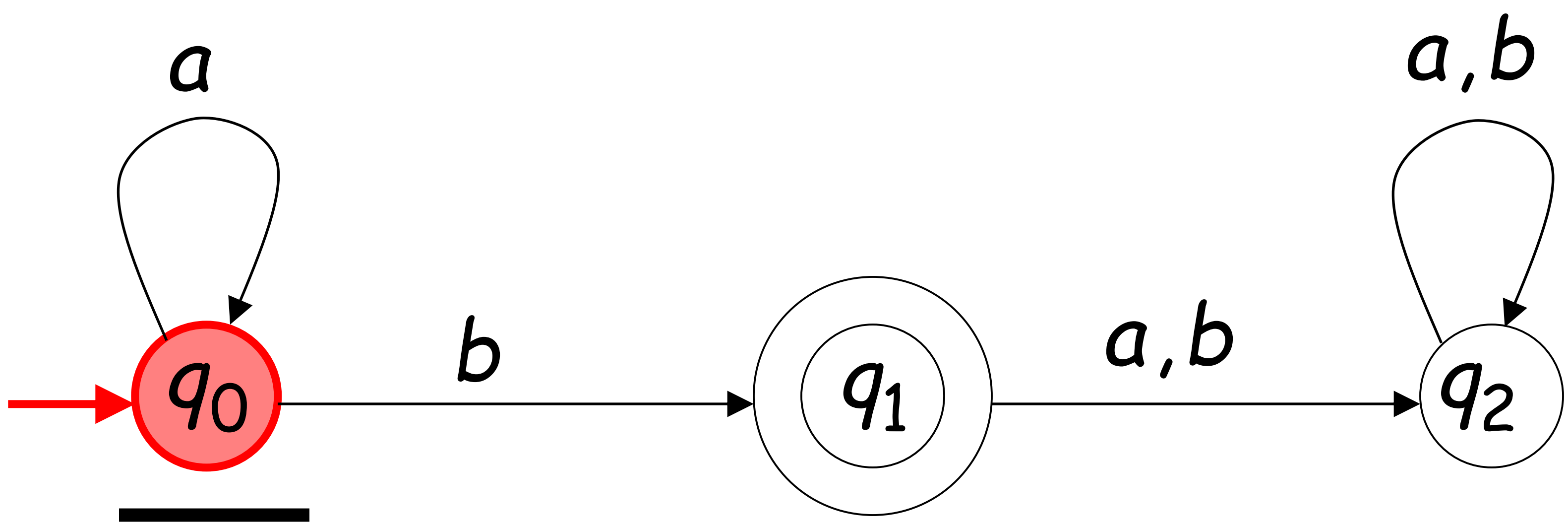
accept

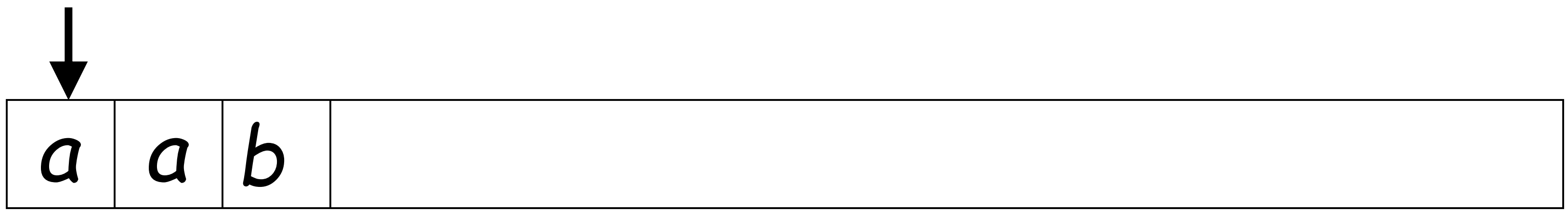
Another Example

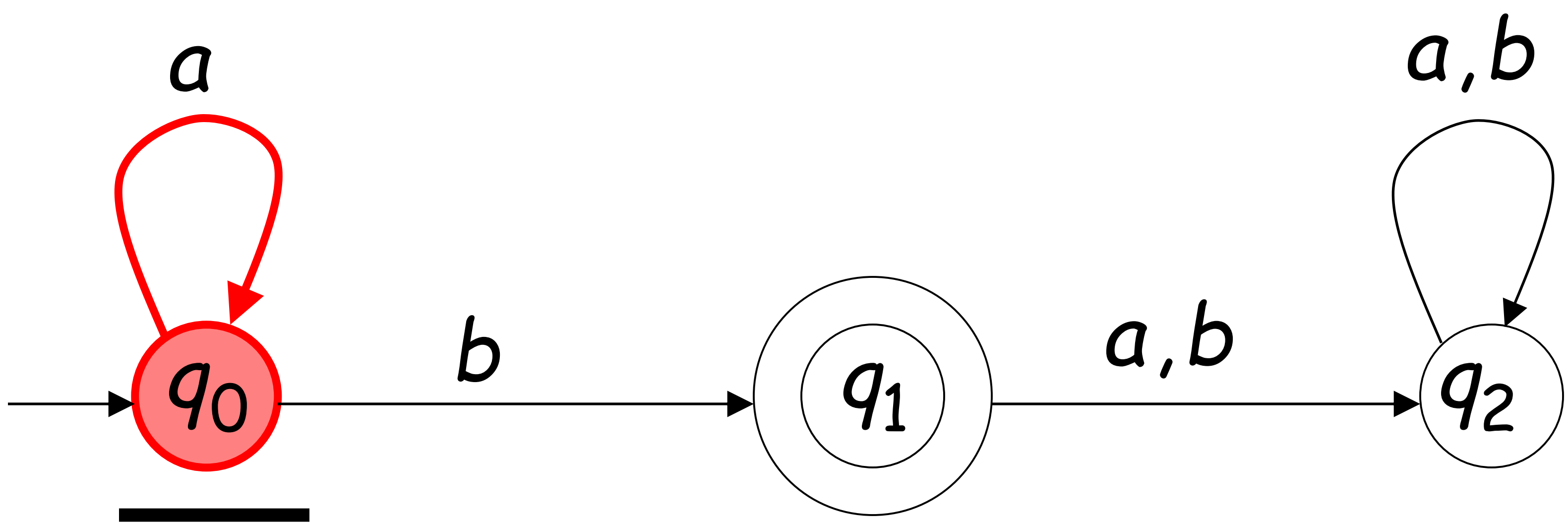


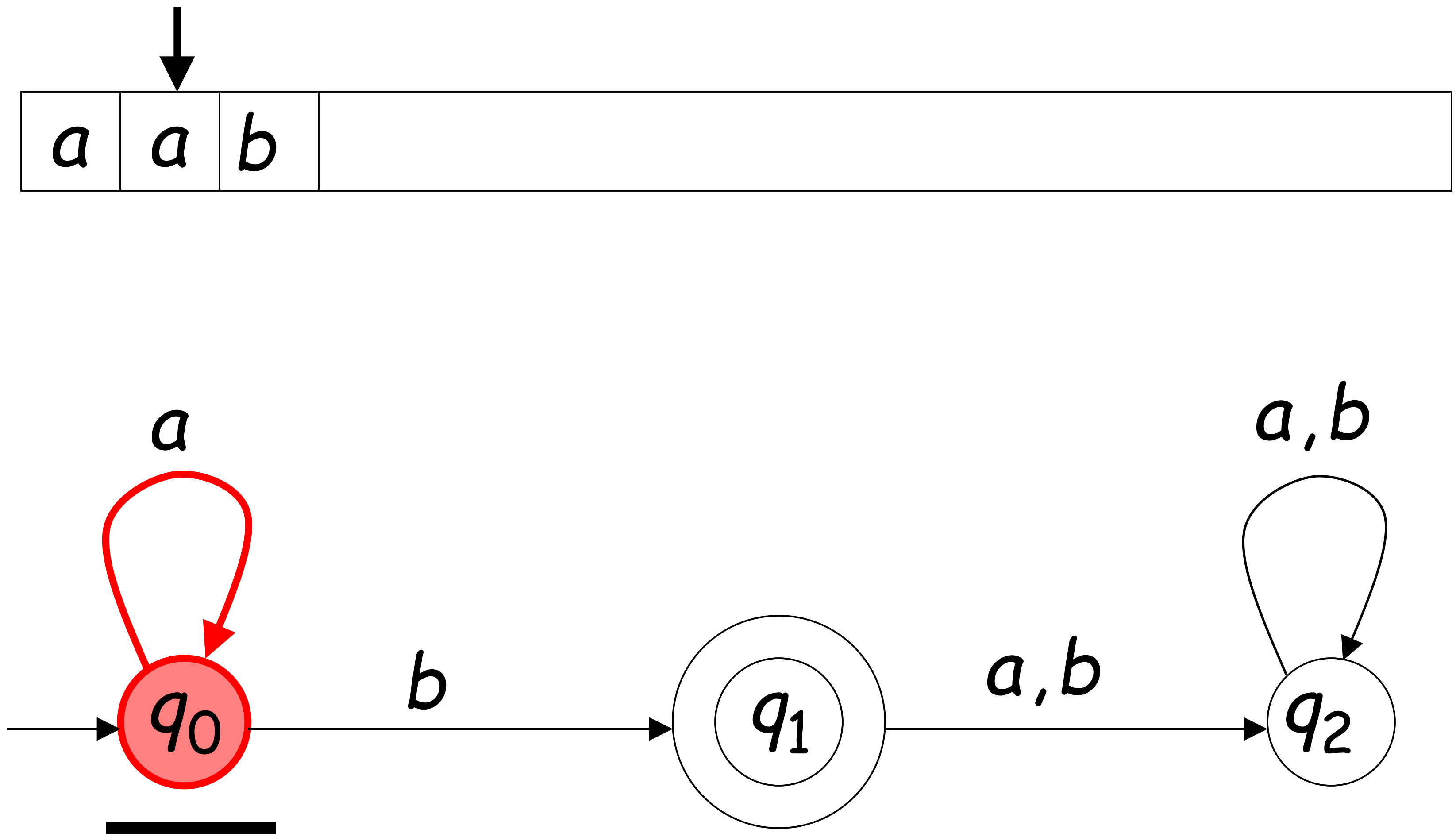


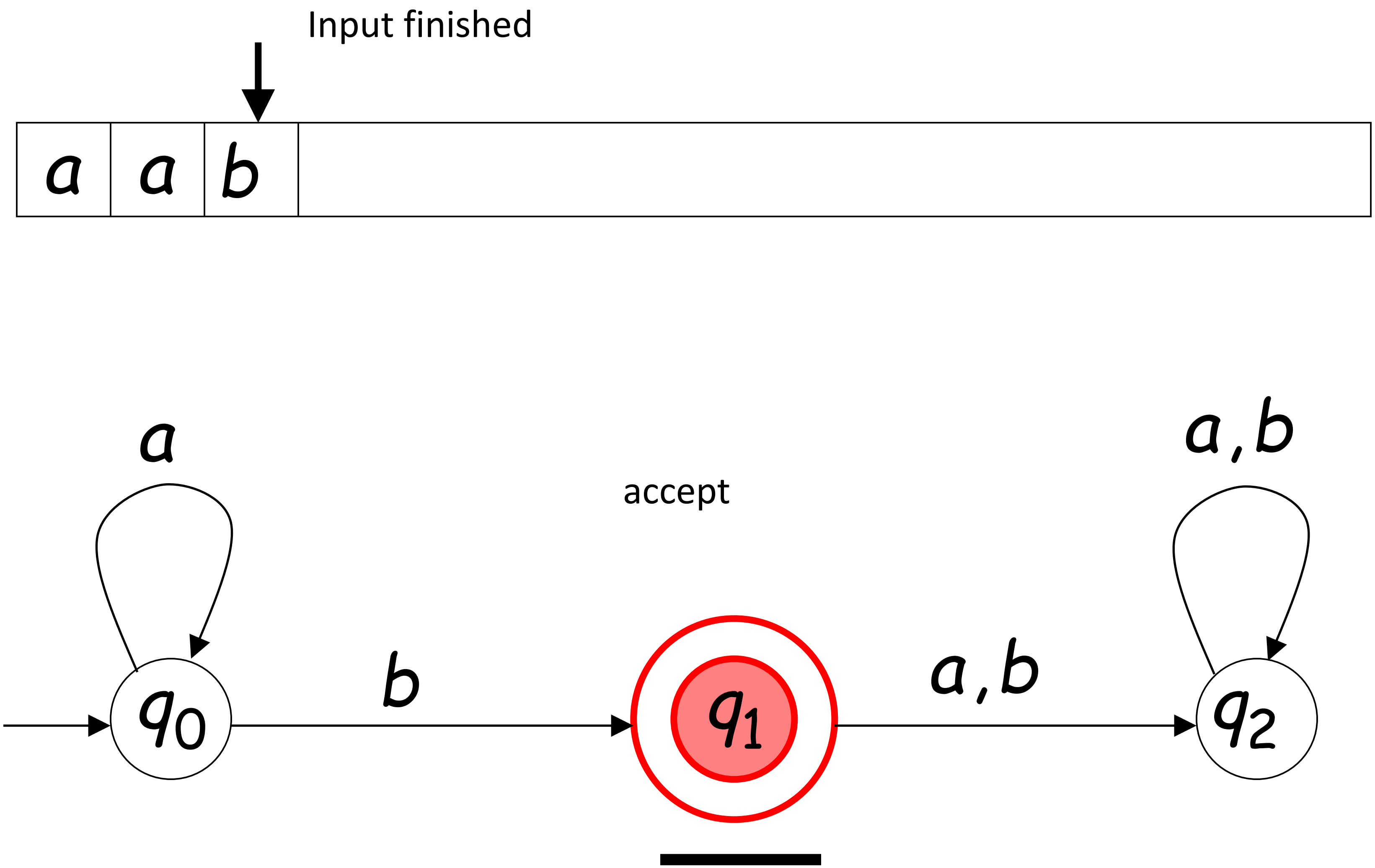
Input String

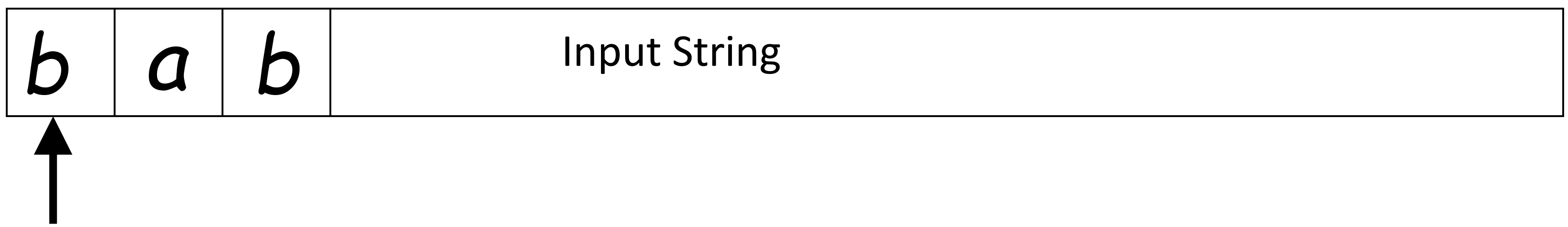



a
a
b

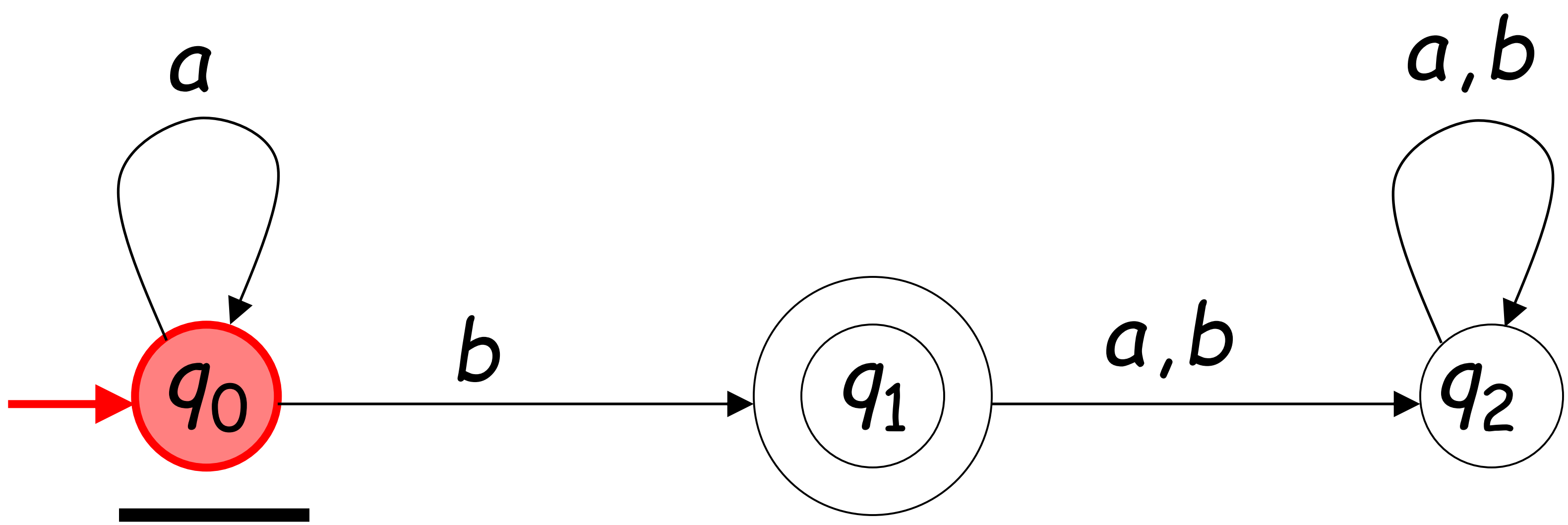

a
a, b
b
q_0
q_1
a, b
q_2

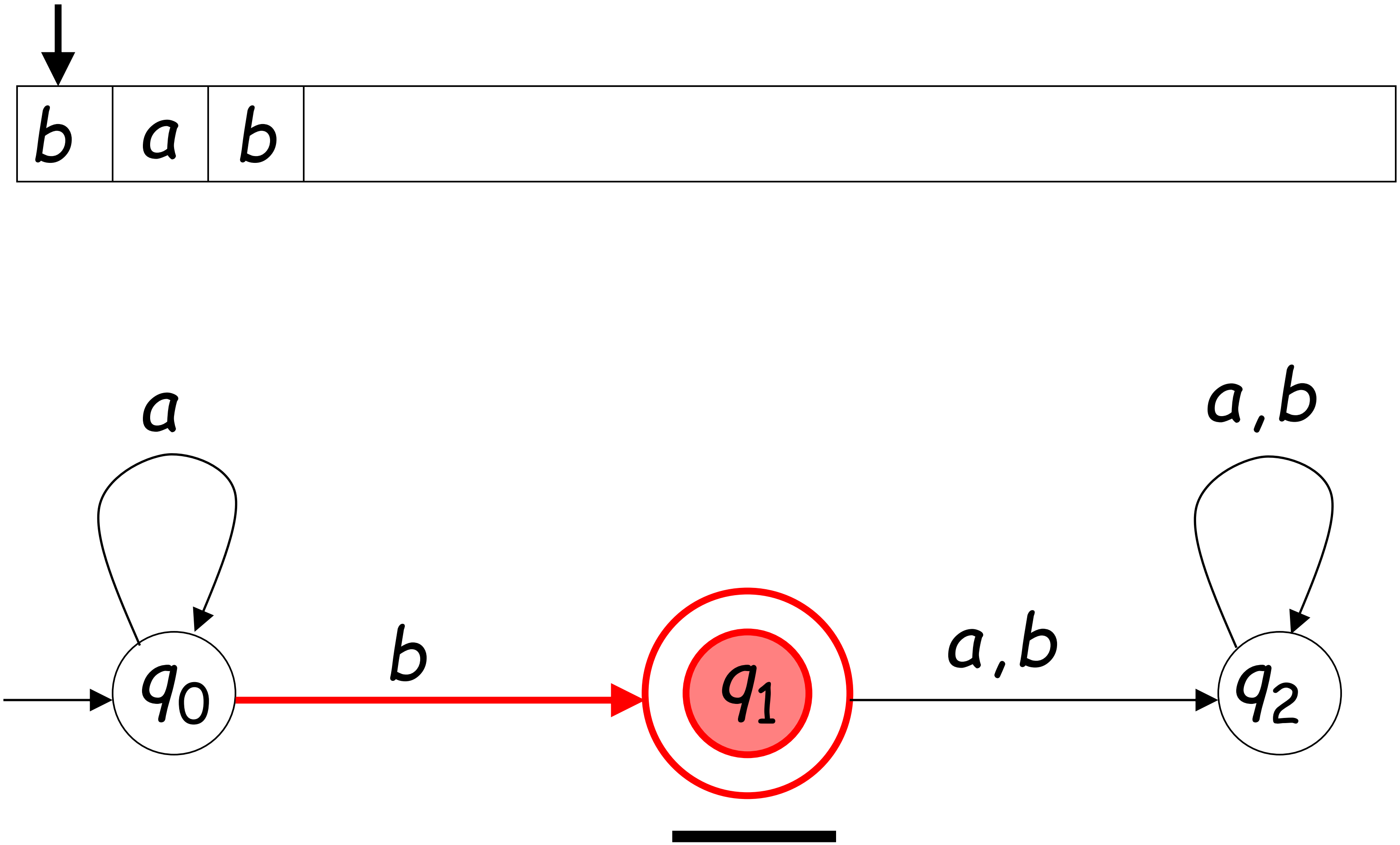

a
a
b
a
a,b
b
q_0
q_1
a,b
q_2

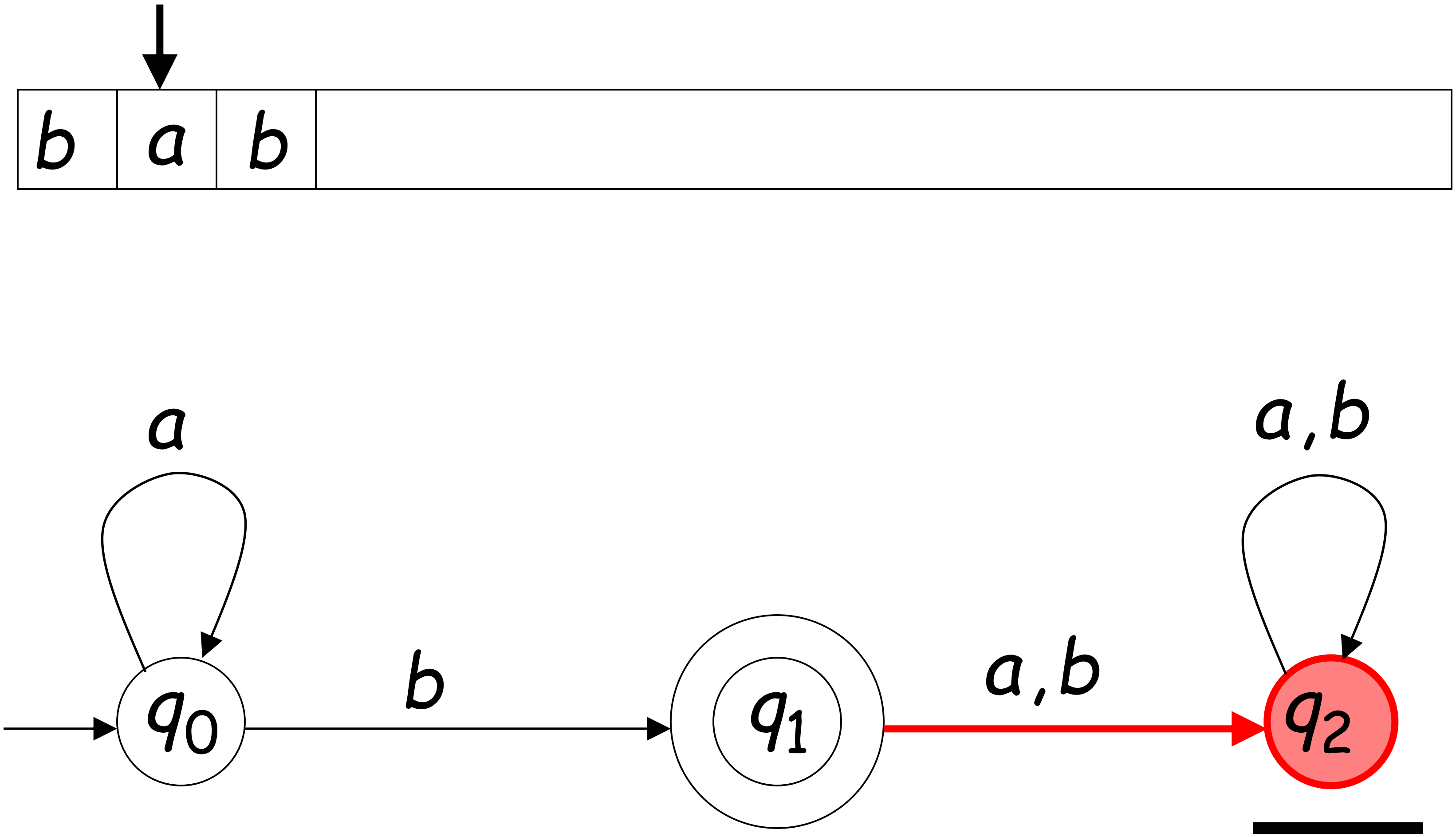

Input finished
a a b
a
accept
a,b
b
a,b
q_0
q_1
q_2



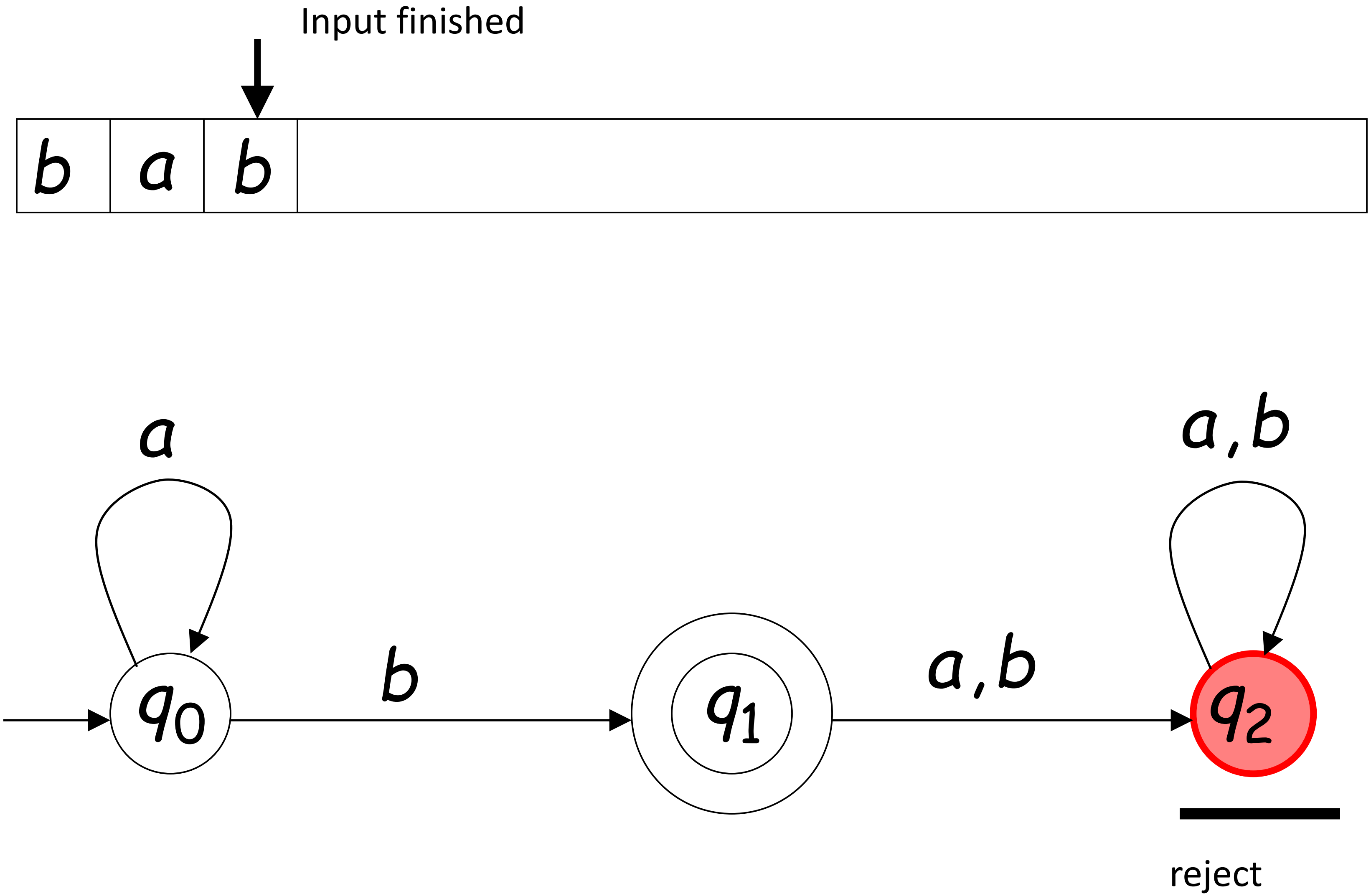
A rejection case




b
a
b
a
a,b
b
a,b
q_0
q_1
q_2

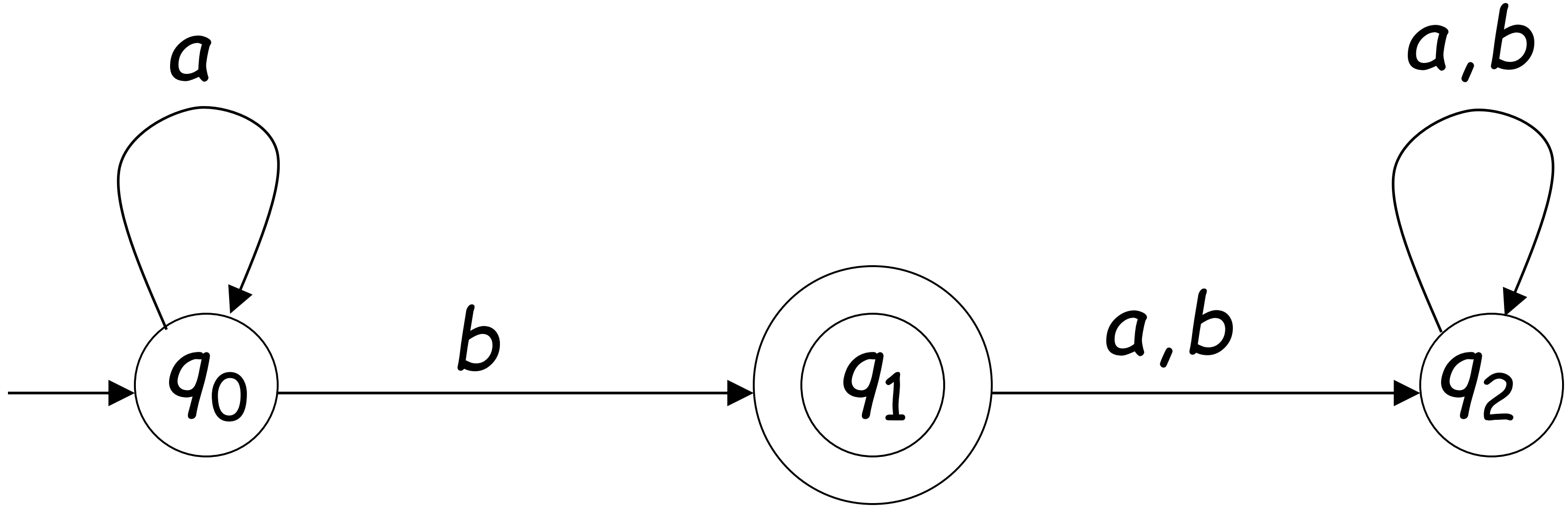


b
a
b
a
a,b
b
a,b
q_0
q_1
q_2


Input finished
b
a
b
a
b
a,b
a,b
q_0
q_1
q_2
reject

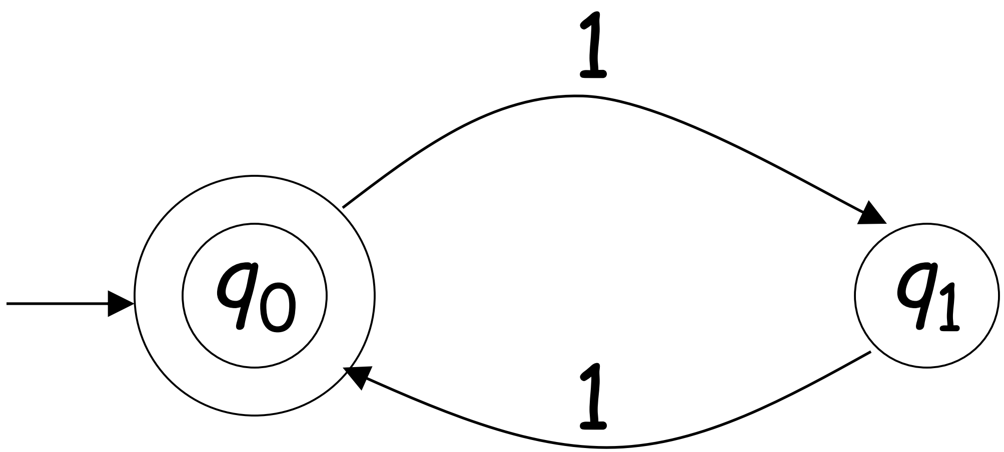
Language Accepted:

$$L = \{a^n b : n \geq 0\}$$



Another Example

Alphabet: $\Sigma = \{1\}$

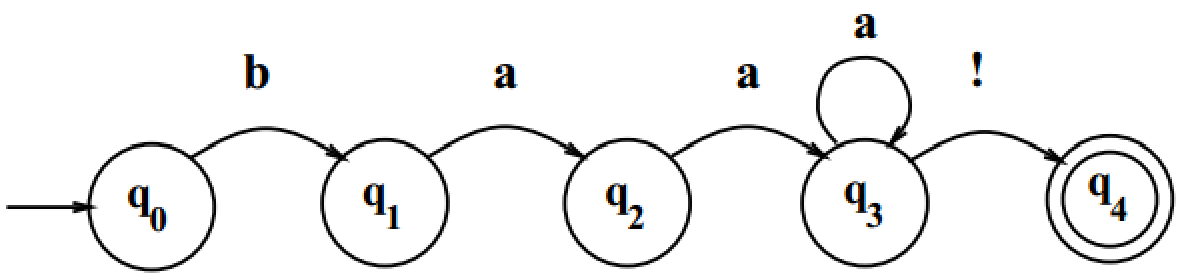


Language Accepted:

$$EVEN = \{x : x \in \Sigma^* \text{ and } x \text{ is even}\}$$
$$= \{\lambda, 11, 1111, 111111, \ldots\}$$

State-transition table

- Represent an automaton
- **Example**



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

Finite automaton

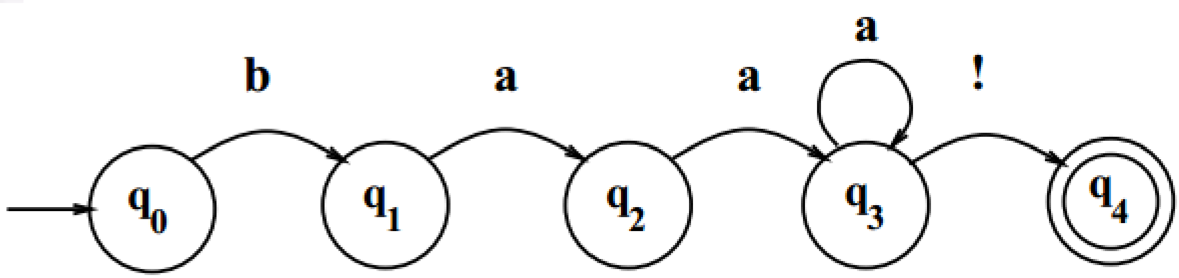
- Q: a finite set of N states $q_0, q_1, \ldots q_N$
- Σ: a finite input alphabet of symbols
- q_0: the start state
- F: the set of final states, $F \subseteq Q$
- $\delta(q,i)$: the transition function or transition matrix between states. Given a state $q \in Q$ and an input symbol $i \in \Sigma$, $\delta(q,i)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q;

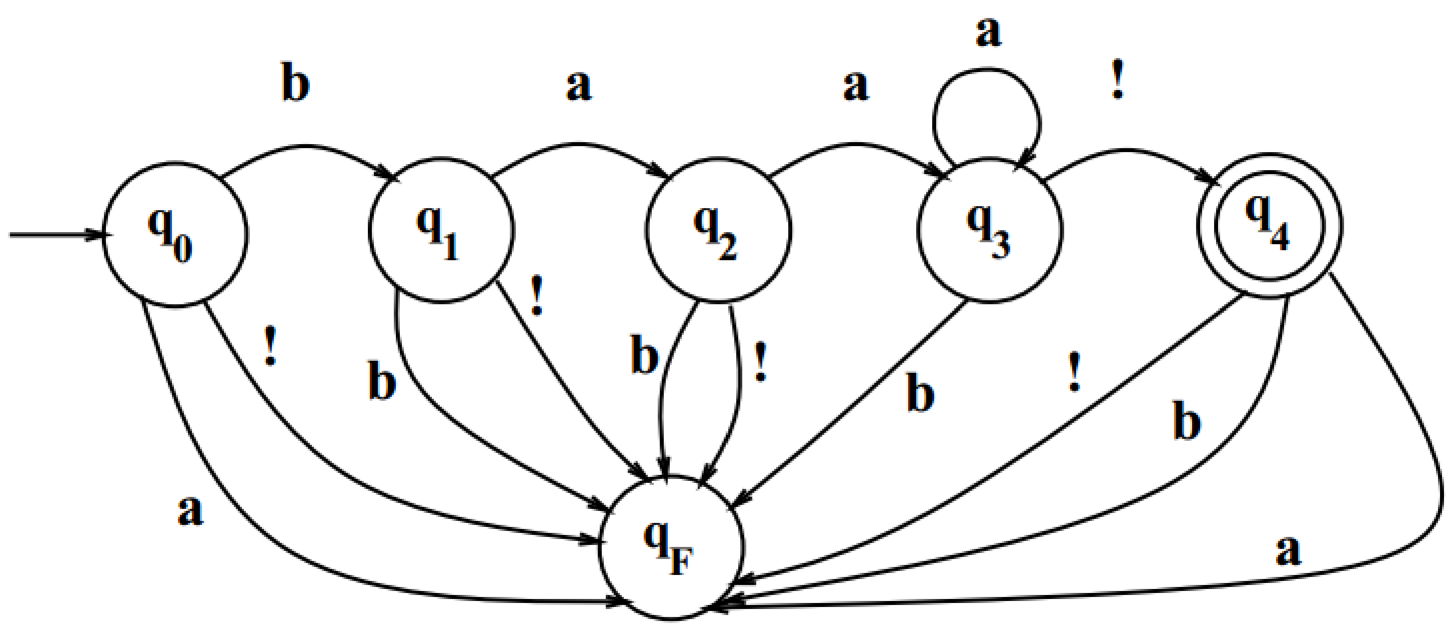
A deterministic algorithm

```
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
```

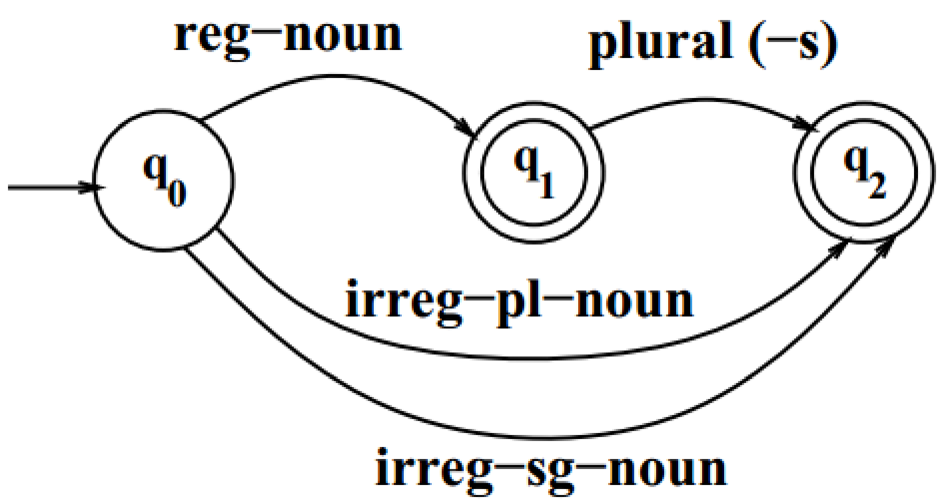
Fail State





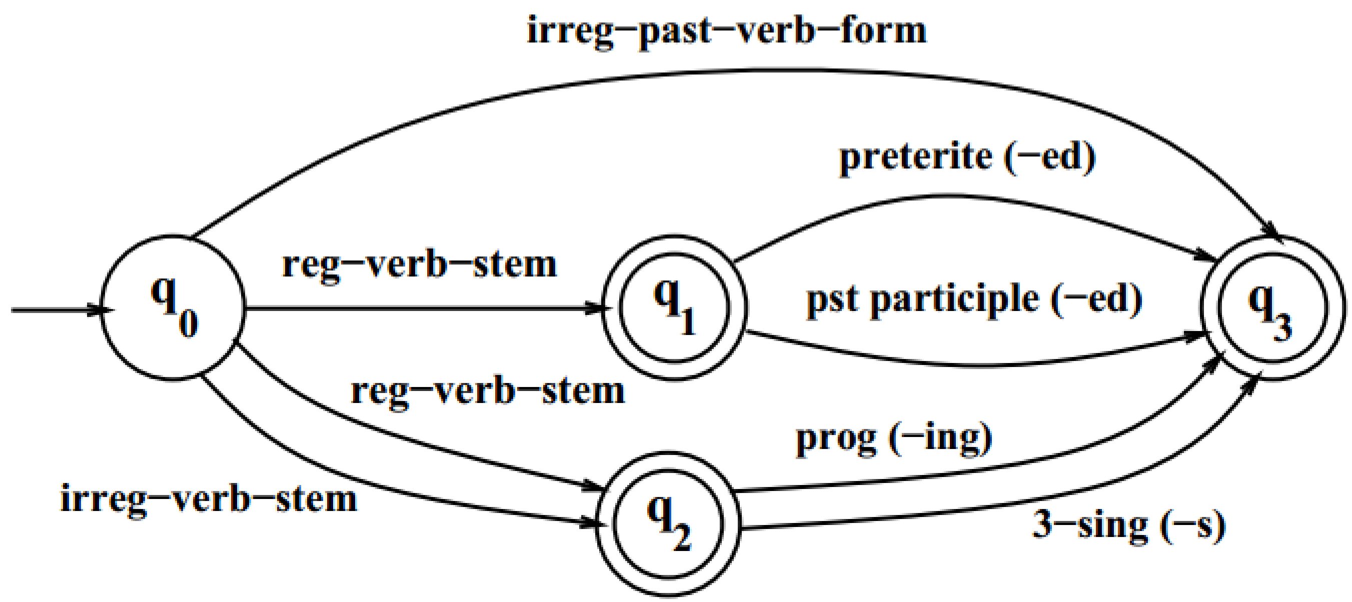
Language Example

- A finite-state automaton for English nominal inflection



Language Example

- A finite-state automaton for English verbal inflection



Exercise

- Write regular expressions for the following languages:

a. the set of all alphabetic strings.

b. the set of all lowercase alphabetic strings ending in a *b*.

c. the set of all strings with two consecutive repeated words (for example ‘Humbert Humbert’ and ‘the the’ but not ‘the bug’ or ‘the big bug’).

d. the set of all strings from the alphabet a, b such that each a is immediately preceded and immediately followed by a b.

e. all strings which start at the beginning of the line with an integer (i.e. 1,2,3...10...10000...) and which end at the end of the line with a word.

f. all strings which have both the word *grotto* and the word *raven* in them. (but not, for example, words like *grottos* that merely *contain* the word *grotto*).

g. write a pattern which places the first word of an English sentence in a register. Deal with punctuation.

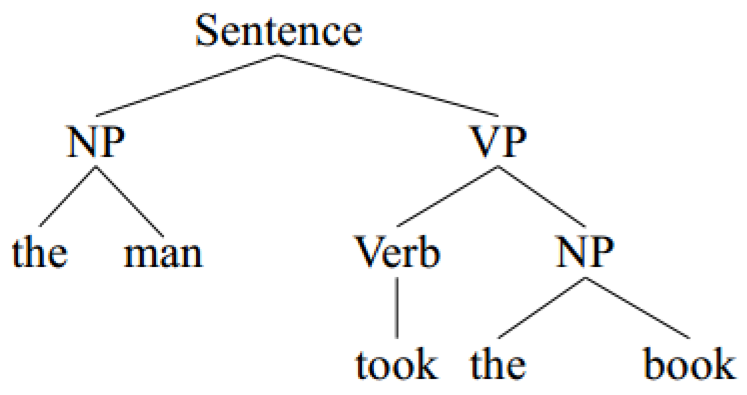
Syntax and CFG

Overview

- What is syntax of a language
- Part of Speech
- Syntax representation
- Context free Grammar
- English Language syntax
- Sinhala Language syntax
- Syntax analysis
- Syntax generation
- Applications (Syntax processing)

Syntax

- Syntax is the study of formal relationships between words
- The word **syntax** comes from the Greek '*syntaxis*' meaning 'setting out together or arrangement'



Part of Speech Tagging

- Words are traditionally grouped into equivalence classes called
  - parts of speech
  - word classes
  - morphological classes
  - lexical tags.
- The part of speech for a word gives a significant amount of information about the **word** and its **neighbors**

English Part of Speech

- ADJECTIVE - modifies a noun.
  Examples: yellow, pretty, useful
  Adjectives have three degrees: Positive, Comparative, and Superlative.

  Example: old, older, oldest
- ARTICLE - specifies whether the noun is specific or a member of a class.

  Examples: a, an, the
- ADVERB - modifies a verb or an adjective. Many adverbs have the suffix **-ly**.

  Examples: very, extremely, carefully

English Part of Speech

- CONJUNCTION - joins components of a sentence or phrase.
  Examples: and, but, or
- INTERJECTION - is used for exclamations.
  Examples: Oh!, Aha!
- NOUN - names an object or action. *Common nouns* refer to ordinary things. *Proper nouns* are usually capitalized and refer to persons, specific things or specific places.
  Examples: mouse, fire, Michael

English Part of Speech

- PREPOSITION - indicates relationship or relative position of objects.
  Examples: in, about, toward
- PRONOUN - is used in place of a noun. *Personal pronouns* are used to refer to persons. *Interrogative pronouns* introduce questions. *Demonstrative pronouns* refer to a previously mentioned object or objects. *Relative pronouns* introduce clauses.
  Examples: he, this
- VERB - specifies an action or links the subject to a complement. The tense of a verb indicates the time when the action happened, e.g., past, present, of future.
  Examples: take, is, go, fire

Part of Speech Tagging

- Part-of-speech tagging (or just **tagging** for short) is the process of assigning a part-of-speech or other lexical class marker to each word in a corpus

VB	DT	NN	.
Book	that	flight	.

VBZ	DT	NN	VB	NN	?
Does	that	flight	serve	dinner	?

- *book* is **ambiguous**. That is, it has more than one possible usage and part of speech

Degree of ambiguity

Unambiguous (1 tag)	**35,340**	
Ambiguous (2-7 tags)	**4,100**	
2 tags	3,760	
3 tags	264	
4 tags	61	
5 tags	12	
6 tags	2	
7 tags	1	("still")

Figure 8.7 The number of word types in Brown corpus by degree of ambiguity (after DeRose (1988)).

Tag sets for English

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*

Tag sets for English

Tag	Description	Example	Tag	Description	Example
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	“	Left quote	(‘ or “)
POS	Possessive ending	*’s*	”	Right quote	(’ or ”)
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one’s*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Sinhala Part of Speech

1. *Noun - නාම පද.*
2. *Verb - ක්‍රියා පද.*
3. *Upasarga – උපසර්ග පද (no direct matching with English grammar)*
4. *Nipatha – නිපාත පද (no direct matching with English grammar)*

TAG	Description	Example
NNR	Common Noun Root	මිනිස්, පුටු, බලු
NNM	Common Noun Masculine	මිනිසා, බල්ලා, ශිෂ්‍යයෝ, එළුවන්
NNF	Common Noun Feminine	තිලියෝ , ඇතින්න
NNN	Common Noun Neuter	පුටු, ගම
NNPA	Proper Noun Animate	රත්නායකගේ
NNPI	Proper Noun Inanimate	රත්නපුරේ, ලක්ස්ප්‍රේ
PRPM	Pronoun Masculine	ඔහු, ඒකා, කොයිකා
PRPF	Pronoun Feminine	ඇය, ඕකී
PRPN	Pronoun Neuter	එය, ඕක
PRPC	Pronoun Common	මම, ඔවුහු
QFNUM	Number Quantifier	එක, දෙවනි
DET	Determiner	මේ, ඒ, අර, ඔය, බොහෝ, සියලු,
JJ	Adjective	රළු, සුමුදු
RB	Adverb	වහා, ඉතින්
RP	Particle	ම, ලු, ය, ද, නම්, වැනි, වුකලී

VFM	Verb Finite Main	බලයි, බැලූහ
VNF	Verb Non Finite	බලා, බලමින්, බලද්දී, බලතොත්
VP1	Verb Participle 1	බලන,බැලූ, කළ
VP2	Verb Participle 2	බලනු, බැලූවා
VP3	Verb Participle 3	බැලීය
VP4	Verb Participle 4	බලන්නේ, බැලූවේ, කළේ
VNN	Verbal Non Finite Noun	බැලීම්, බැලිලී, බැලුම්
POST	Postpositions	ගැන, ලෙස, සඳහා
CC	Conjunctions	සහ , ද, සමඟ, හෝ
NVB	Noun in Kriya Mula	පාඩම් කරනවා, භාජනය කරනවා
JVB	Adjective in Kriya Mula	තීකරු වෙනවා, එකඟ වෙනවා, අඩු කරනවා
UH	Interjection	අහෝ , චිඃ, ෂික්, ආඃ
FRW	Foreign Word	Computer
SYM	Not Classified	A4

Tagging algorithms

- **Rule-based** taggers and **Stochastic** taggers.
- **Rule-based taggers** generally involve a large database of hand-written disambiguation rule which specify
  - **ENGTWOL**
- **Stochastic taggers** generally resolve tagging ambiguities by using a training corpus to compute the probability of a
  - **HMM tagger**

Rule based Tagging

- earliest algorithms for automatically assigning part-of-speech were based on a two-stage architecture
- The first stage used a dictionary to assign each word a list of potential parts of speech
- The second stage used large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word.
- The **ENGTWOL** tagger is based on the same two stage architecture

ENGTWOL Results

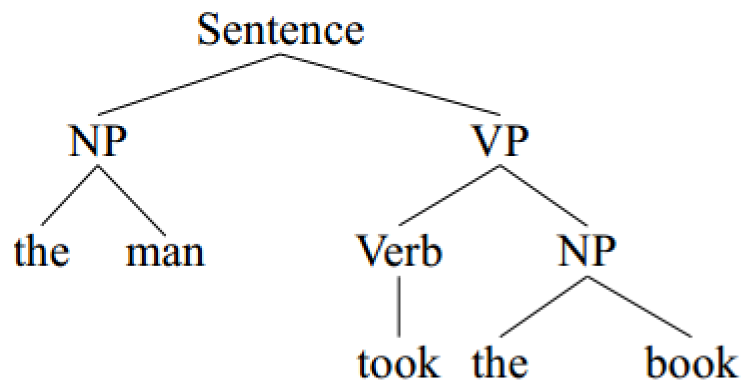
Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLUTE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	IMPERATIVE VFIN
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

Transformation-Based Tagging

- TBL is based on rules that specify what tags should be assigned to what words
- TBL is a machine learning technique, in which rules are automatically induced from the data.
- TBL is a supervised learning technique; it assumes a pre-tagged training corpus

Other Issues

- Multiple tags and multiple words
- **Tag indeterminacy** arises when a word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate.
  - Some taggers allow the use of multiple tags
- The second issue concerns **multi-part words**
  - allow prepositions like '*in terms of*' to be treated as a single word by adding numbers to each tag
- **Unknown words**



Context-free grammar

Constituency

- The fundamental idea of constituency is that groups of words may be CON- STITUENT have as a single unit or phrase, called a **constituent**
- **Example**
  - **noun phrase** often acts as a unit
- **Context-free grammars** allow us to model these constituency facts

preposed or postposed constructions

On September seventeenth, I'd like to fly from Atlanta to Denver
I'd like to fly on September seventeenth from Atlanta to Denver
I'd like to fly from Atlanta to Denver on September seventeenth

But again, while the entire phrase can be placed differently, the individual words making up the phrase cannot be:

*On September, I'd like to fly seventeenth from Atlanta to Denver
*On I'd like to fly September seventeenth from Atlanta to Denver
*I'd like to fly on September from Atlanta to Denver seventeenth

English Noun Phrase

```
<noun phrase> =
       "the" <specific proper noun> |
       <proper noun> |
       <non-personal pronoun> |
       <article> [<adverb>* <adjective>] <noun> |
       [<adverb>* <adjective>] <noun-plural> |
       <proper noun-possessive> [<adverb>* <adjective>] <noun> |
       <personal possessive adjective> [<adverb>* <adjective>] <noun>
|
       <article> <common noun-possessive>
              [<adverb>* <adjective>] <noun>
```

<"the"> <specific proper noun>
the Atlantic Ocean
the Sahara

<proper noun>
John
America
Dr. Allen
State Street

<non-personal pronoun>
someone
anyone
this

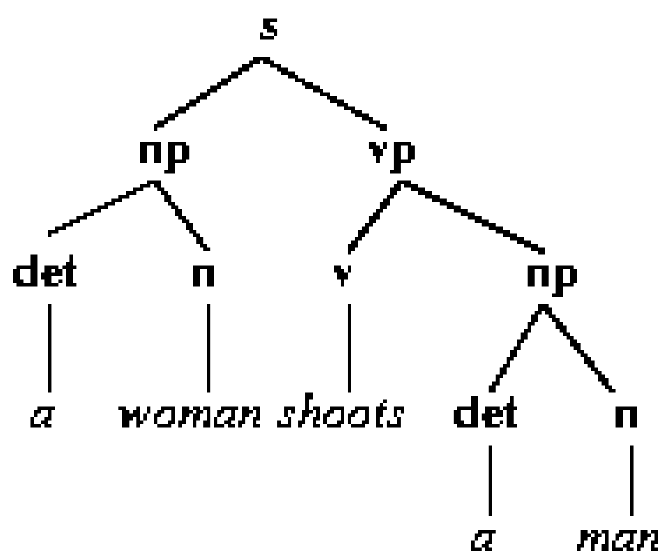
<article> [<adverb>* <adjective>] <noun>
a very long bridge
the book
the extremely pretty dress

[<adverb>* <adjective>] <noun-plural>
very yellow flowers
books

Context-Free Grammar

- Most commonly used mathematical system for modeling constituent structure
- Phrase-Structure Grammar

S	→	NP VP
NP	→	ART NOUN
NP	→	NP PP
PP	→	P NP
VP	→	VERB NP
VP	→	VERB NP PP
ART	→	the
ART	→	a
NOUN	→	telescope
NOUN	→	man
NOUN	→	spider
VERB	→	saw
VERB	→	complimented
P	→	with
P	→	in



Context-free grammar

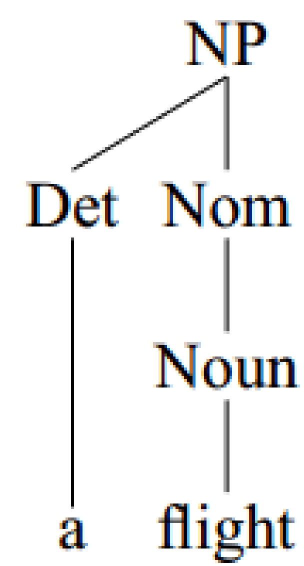
- Consists of a set of **rules** or **productions**
- Context free rules can be hierarchically embedded
- Symbols that correspond to words in the language ('the', 'nightclub') are called **terminal** symbols
- The symbols that express clusters or generalizations of these are called **nonterminals**
- In each context-free rule, the item to the right of the arrow (→) is an ordered list of one or more terminals and **nonterminals**

Example

$$NP \rightarrow Det\ Nominal$$
$$NP \rightarrow ProperNoun$$
$$Nominal \rightarrow Noun \mid Noun\ Nominal$$

$$Det \rightarrow a$$
$$Det \rightarrow the$$
$$Noun \rightarrow flight$$

- String *a flight* can be **derived** from the nonterminal *NP*
- Sequence of rule expansions is called a **derivation** of the string of words
- Represent a derivation by a **parse tree**
- **bracketed notation** is another way to represent a parse tree



$[_S [_{NP} [_{Pro}\ \text{I}]] [_{VP} [_V\ \text{prefer}] [_{NP} [_{Det}\ \text{a}] [_{Nom} [_N\ \text{morning}] [_N\ \text{flight}]]]]]$

A more formal definition

- A CFG is a 4-tuple <N,Σ,P, S> consisting of
  - a set of non-terminal symbols N
  - a set of terminal symbols Σ
  - a set of productions P
    - $A \rightarrow \alpha$
    - A is a non-terminal
    - α is a string of symbols from the infinite set of strings $(\Sigma \cup N)\star$
  - a designated start symbol S

What context free means

All the use of the term context-free really means is that the non-terminal on the left-hand side of the rule is sitting over there all by itself.

$A \rightarrow B\ C$

In other words, I can rewrite A as BC, regardless of the context in which I find the A.

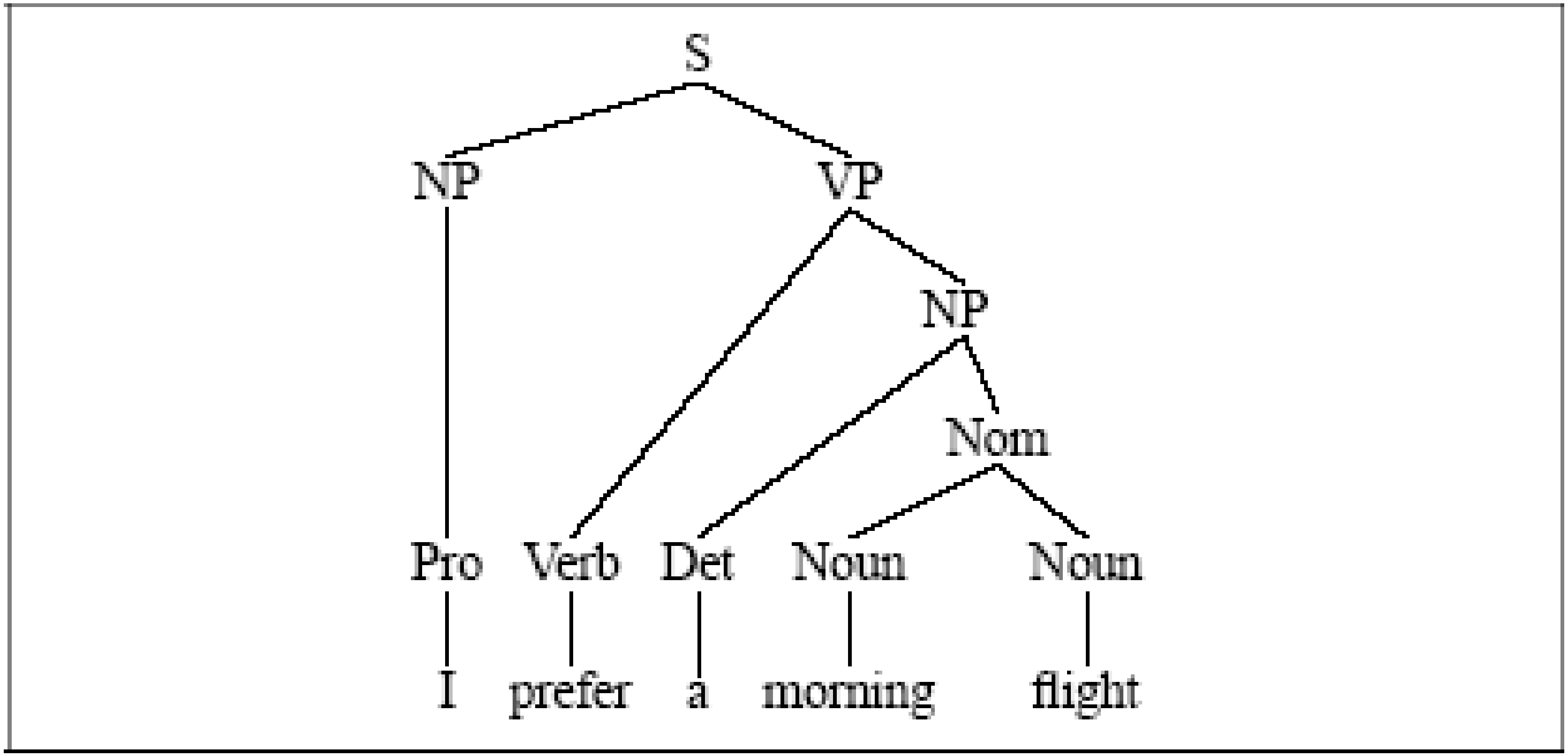
An example lexicon

$$
\begin{aligned}
Noun &\rightarrow flights \mid breeze \mid trip \mid morning \mid \ldots \\
Verb &\rightarrow is \mid prefer \mid like \mid need \mid want \mid fly \\
Adjective &\rightarrow cheapest \mid non-stop \mid first \mid latest \\
&\quad \mid other \mid direct \mid \ldots \\
Pronoun &\rightarrow me \mid I \mid you \mid it \mid \ldots \\
Proper\text{-}Noun &\rightarrow Alaska \mid Baltimore \mid Los\ Angeles \\
&\quad \mid Chicago \mid United \mid American \mid \ldots \\
Determiner &\rightarrow the \mid a \mid an \mid this \mid these \mid that \mid \ldots \\
Preposition &\rightarrow from \mid to \mid on \mid near \mid \ldots \\
Conjunction &\rightarrow and \mid or \mid but \mid \ldots
\end{aligned}
$$

An example grammar

S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	$\rightarrow$	*Noun Nominal*	morning + flight
	|	*Noun*	flights
VP	$\rightarrow$	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	$\rightarrow$	*Preposition NP*	from + Los Angeles

A simple parse tree



Sentence-level Constructions

- Consistency we will continue to focus on sentences

```
<English Sentence> =
       <Simple Sentence> |
       <Compound Sentence>

<Simple Sentence> =
       <Declarative Sentence> |
       <Interrogative Sentence> |
       <Imperative Sentence> |
       <Conditional Sentence>

<Compound Sentence> =
       <Simple Sentence> <conjunction> <Simple Sentence> |
       "Either" <Declarative Sentence> "or" <Declarative Sentence>
       "Either" <Imperative Sentence> "or" <Imperative Sentence>
```

Basic types of sentences

Declaratives

- John left.
- $S \rightarrow NP\ VP$

Imperatives

- Leave!
- $S \rightarrow VP$

Yes-No Questions

- Did John leave?
- $S \rightarrow Aux\ NP\ VP$

WH Questions (who, where, what, which, why, how)

- When did John leave?
- $S \rightarrow Wh\text{-}NP\ Aux\ NP\ VP$
- $S \rightarrow Wh\text{-}NP\ VP$

Recursion

- Nominal → Nominal PP (PP) (PP)
  - Is an example of RECURSIVE rule
- Other examples:
  - NP → NP PP
  - VP → VP PP
- Recursion a powerful device, but could have bad consequences (see lectures on parsing)

Recursion and VP attachment

- Flights to Miami
- Flights to Miami from Boston
- Flights to Miami from Boston in April
- Flights to Miami from Boston in April on Friday
- Flights to Miami from Boston in April on Friday with lunch.

Coordination

- NP → NP and NP
  - John and Mary left
- VP → VP and VP
  - John talks softly and carries a big stick
- S → S and / but / S
  - Kim is a lawyer but Sandy is reading medicine.
- In fact, probably English has a
  - XP → XP and XP

  rule

Write suitable CFG for English NP

```
<noun phrase> =
       "the" <specific proper noun> |
       <proper noun> |
       <non-personal pronoun> |
       <article> [<adverb>* <adjective>] <noun> |
       [<adverb>* <adjective>] <noun-plural> |
       <proper noun-possessive> [<adverb>* <adjective>] <noun> |
       <personal possessive adjective> [<adverb>* <adjective>] <noun>
|
       <article> <common noun-possessive>
              [<adverb>* <adjective>] <noun>
```

Write suitable CFG for English VP

```
<verb> = <V1s> |<V2s> |<V3s> |
       <V1p> |<V2p> |<V3p> |
       <Vpast> |<linking verb>

<linking verb> = "am" |"are" |"is" | "was"| "were" |
       "look" | "looks" | "looked" |
       "become" | "became" | "become" | ...

<verb phrase> =
       ("had" |"have" |"has") ["not"] <Vpastp> |
       ("had" |"have" |"has") ["not"] "been" [<Vpastp> | <Ving>] |
       <auxV> ["not"] "have" <Vpastp> |
       <auxV> ["not"] "have" "been" [<Vpastp> | <Ving>] |
       <auxV> ["not"] "be" [<Vpastp> | <Ving>] |
       <auxV> ["not"] <Vinf> |
       "ought" ("to" | "not") <Vinf> |
       "ought" ("to" | "not") "be" [<Vpastp> | <Ving>] |
       "ought" ("to" | "not") "have" <Vpastp> |
       "ought" ("to" | "not") "have" "been" [<Vpastp> | <Ving>] |
       ("do" |"does" |"did") ["not"] [<Vinf>] |
       ("am" |"are" |"is" |"was" |"were") ["not"] [<Vpastp> | <Ving>] |
       ("am" |"are" |"is" |"was" |"were") ["not"] "being" [<Vpastp>] |
       ("am" |"are" |"is" |"was" |"were") ["not"] "going" "to" [<Vinf>]
```