Natalie Parde

UIC CS 421

Text Preprocessing and Edit Distance

What is text preprocessing?

- Automated organization, normalization, and manipulation of text such that it can more easily be handled by downstream language processing tasks.

"Have some wine," the March Hare said in an encouraging tone.

Alice looked all round the table, but there was nothing on it but tea. "I don't see any wine," she remarked.

"There isn't any," said the March Hare.

- Lewis Carroll, *Alice's Adventures in Wonderland*

have some wine [PERSON 1] said in an encouraging tone

[PERSON 2] looked all round the table but there was nothing on it but tea

i don't see any wine she remarked

there isn't any said [PERSON 1]

- Lewis Carroll, *Alice's Adventures in Wonderland*

Text preprocessing steps can (and should!) vary depending on your needs.

capitalization

written numbers vs. digits

punctuation

proper nouns

British vs. American spellings (for English text)

Important

Not Important

One way to preprocess text is by using regular expressions.

- Regular expressions: A formal language for specifying text strings.
- How can we search for any of these?
  - Donut
  - donut
  - Doughnut
  - doughnut
  - Donuts
  - doughnuts

Regular Expression Terminology

Regex: Common abbreviation for **reg**ular **ex**pression

Disjunction: Logical OR

Range: All characters in a sequence from c_1-c_2

Negation: Logical NOT

Scope: Indicates to which characters the regex applies

Anchor: Matches the beginning or end of a string

Regular Expressions: Disjunctions (and Ranges)

- Disjunction: Letters inside square brackets [az]
- Range: Hyphen between the first and last characters in the range [a-z]

Pattern	Matches	Example
[dD]onut	donut, Donut	This morning would be better with a **donut**.
[0123456789]	Any digit	This morning would be better with **5** donuts.
[A-Z]	An uppercase letter	**D**onuts are an excellent way to start the day.
[0-9]	Any digit	I just ate **5** donuts.

Pattern	Matches	Example
[^dD]onut	Any letter except “d” or “D” before the sequence “onut”	This morning would be better with a co**c**onut.
[^A-Z]	Not an uppercase letter	D**o**nuts are an excellent way to start the day.
[^^]	Not a caret	**W**hat is your favorite kind of donut?
D^o	The pattern “D^o”	Is **D^o**nut a good name for my donut shop?

Regular Expressions: Negation in Disjunction

- Negation: A caret (^) at the beginning of a disjunction [^az]
  - The caret must be at the beginning of the disjunction to negate it

Regular Expressions: More Disjunction

- The pipe | indicates the union (logical OR) of two smaller regular expressions
- a|b|c is equivalent to [abc]

Pattern	Matches	Example
d|D	“d” or “D”	This morning would be better with a **d**onut.

Regular Expressions: Special Characters

- *****: Means that there must be 0 or more occurrences of the preceding expression
- **.**: A wildcard that can mean any character
- **+**: Means that there must be 1 or more occurrences of the preceding expression
- **?**: Means that there must be 0 or 1 occurrences of the preceding expression
- **{m}**: Means that there must be *m* instances of the preceding expression
- **{m,n}**: Means that there must be between *m* and *n* instances of the preceding expression
- **(abc)**: Means that the operation should be applied to the specified sequence

Regular Expressions: Special Characters

Pattern	Matches	Example
donuts*	“donut” or “donuts” or “donutss” or “donutsss”….	This morning I had many **donuts**.
.onut	Any character followed by “onut”	Can I have a co**conut donut**?
donuts+	“donuts” or “donutss” or “donutsss”….	Do you want one donut or two **donuts**?
donuts?	“donut” or “donuts”	Do you want one **donut** or two **donuts**?
donuts{1}	“donuts”	Do you want one donut or two **donuts**?
donuts{0,1}	“donut” or “donuts”	Do you want one **donut** or two **donuts**?
.o(nut)?	Any character followed by “o” or “onut”	Can I have a dis**co donut**?

Regular Expressions: Anchors

- Indicate that a pattern should be matched only at the beginning or end of a word

Pattern	Matches	Example
^Donuts	“Donuts” only when it is at the beginning of a string	**Donuts** are an excellent way to start the day.
donuts\.$	“donuts.” only when it is at the end of the string	I just ate 5 **donuts.**

Case Example: Regex for “the”

the

Fails on: The

[tT]he

Fails on: other

^[tT]he$

Fails on: Grab the disinfectant!

Errors

- In iterating through possible solutions to avoid failures, we were trying to fix two types 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

- This is a recurring theme in NLP!
- 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)

Extra Regular Expression Tips

- If you want to match a special character, you will need to escape it with a backslash: \
- You can also use **shorthand character classes**
  - Can save time and space when searching for all of a specific category of characters
- Beware that different programming languages:
  - May have different sets of shorthand character classes
  - May implement the same shorthands differently
- **Make sure you know the definitions for these in the language you're using!**

Shorthand Character Classes

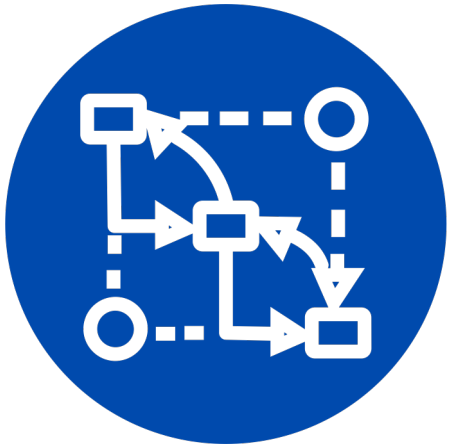
- Can be used inside or outside disjunctions
- Can be negated
- Common shorthand character classes:
  - **\d**: All digits
  - **\s**: All whitespace characters

Regular Expressions: Takeaway Points

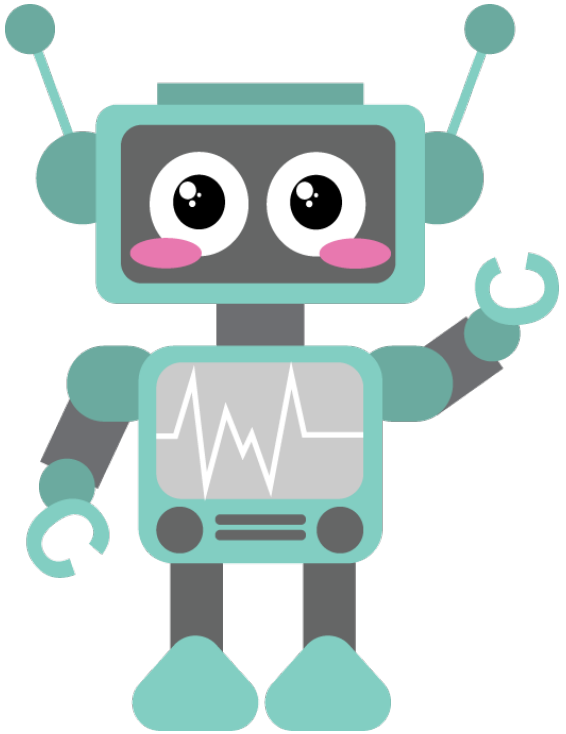
Regular expressions are a surprisingly powerful tool!

They are critical to text tokenization and normalization.

They may also be used to extract **features** for machine learning classifiers.



Regular expressions can be matched using finite state automata.



What are finite state automata?

- **Computational models that can generate regular languages** (such as those specified by a regular expression)
- Also used in other NLP applications that function by **transitioning between finite states**
  - Dialogue systems
  - Morphological parsing
- Singular: Finite State Automaton (FSA)
- Plural: Finite State Automata (FSAs)

Key Components

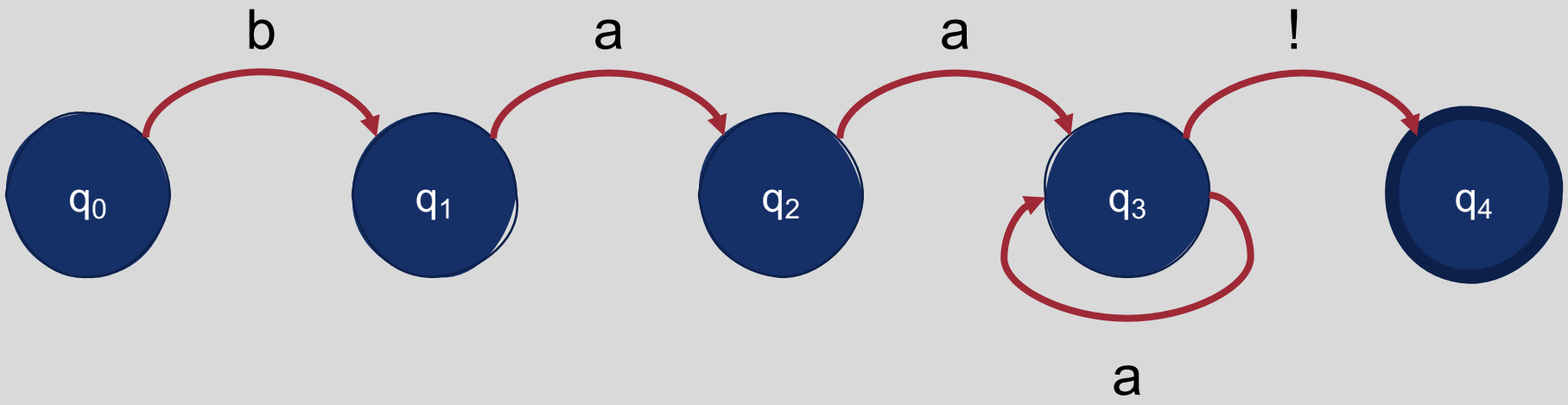
- Finite set of states
  - Start state
  - Final state
- Set of transitions from one state to another

How do FSAs work?

- For a given sequence of items (characters, words, etc.) to match, **begin in the start state**
- **If the next item** in the sequence **matches a state that can be transitioned to** from the current state, **go to that state**
- **Repeat**
  - If no transitions are possible, **stop**
  - If the state you stopped in is a final state, **accept the sequence**

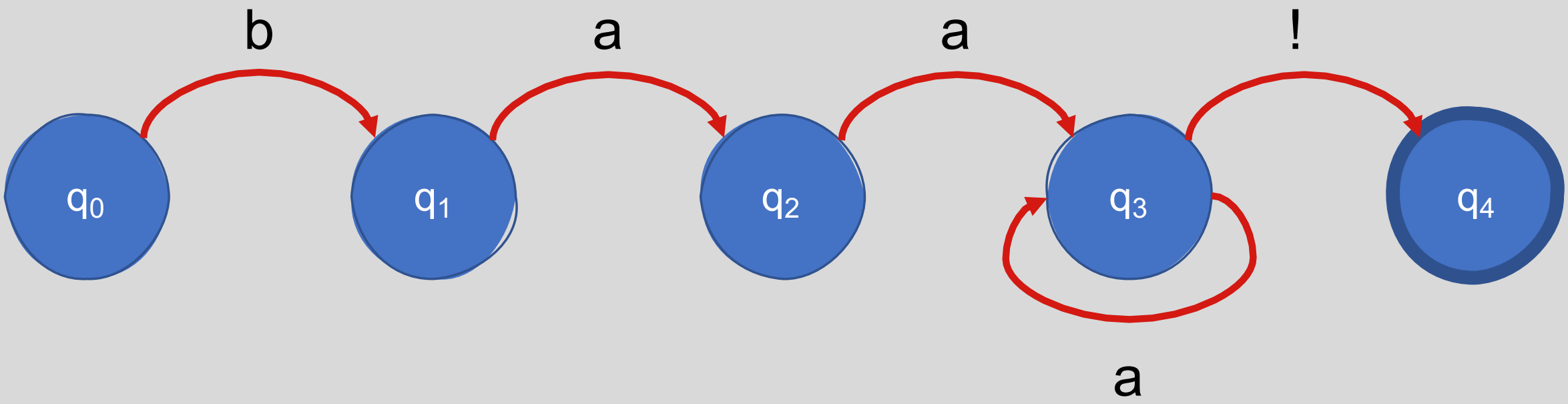
FSAs are often represented graphically.

- Nodes = states
- Arcs = transitions

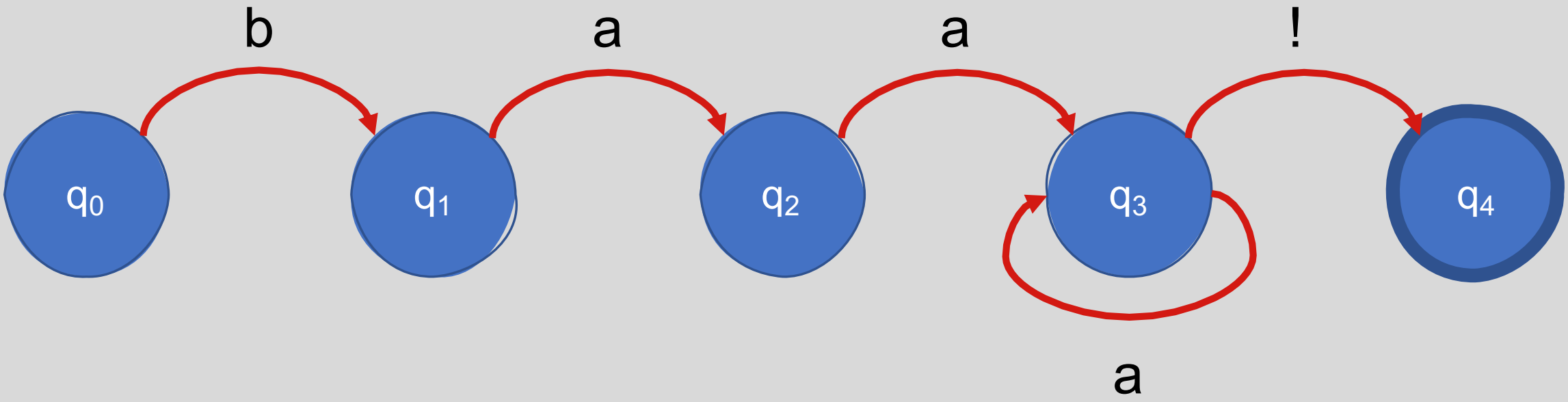


What do we know about this FSA?

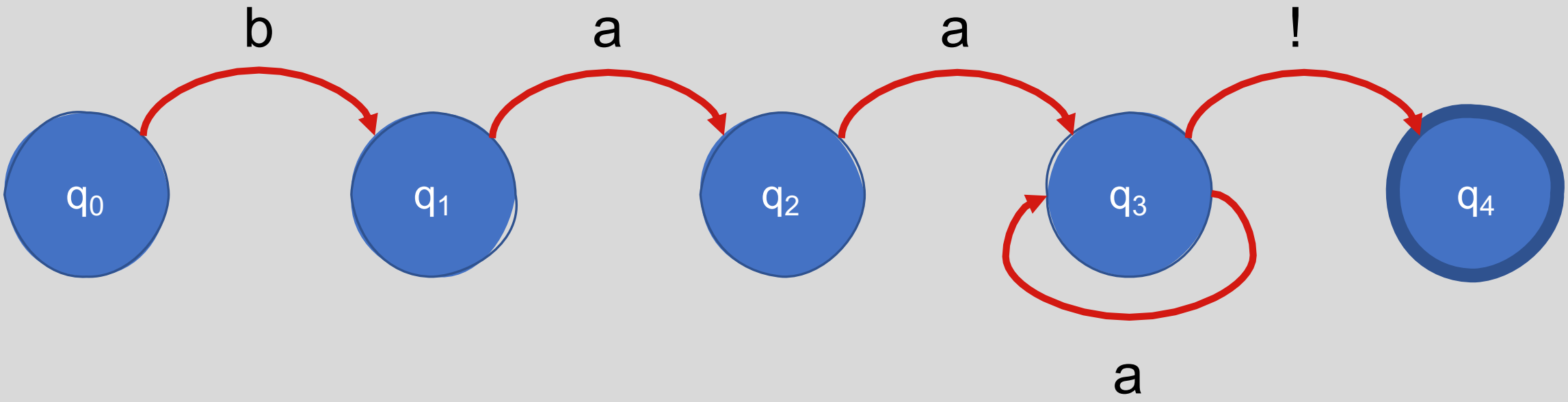
- Five states
  - q_0 is the start state
  - q_4 is the final (accept) state
- Five transitions
- Alphabet = {a, b, !}



Regex that this FSA matches: baa+!

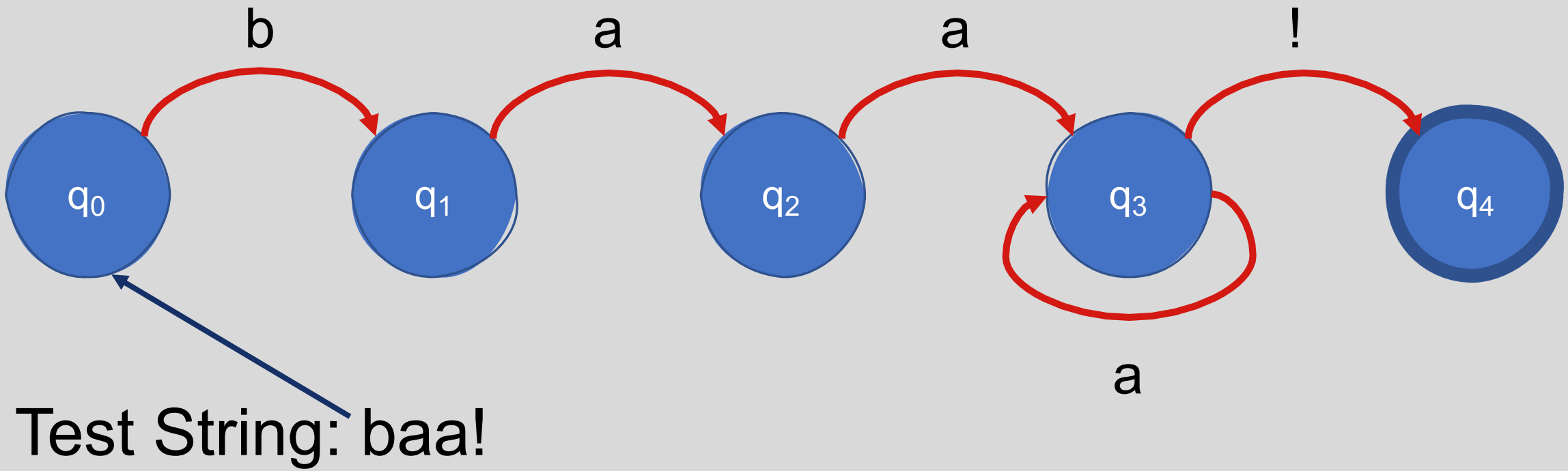


Regex that this FSA matches: baa+!

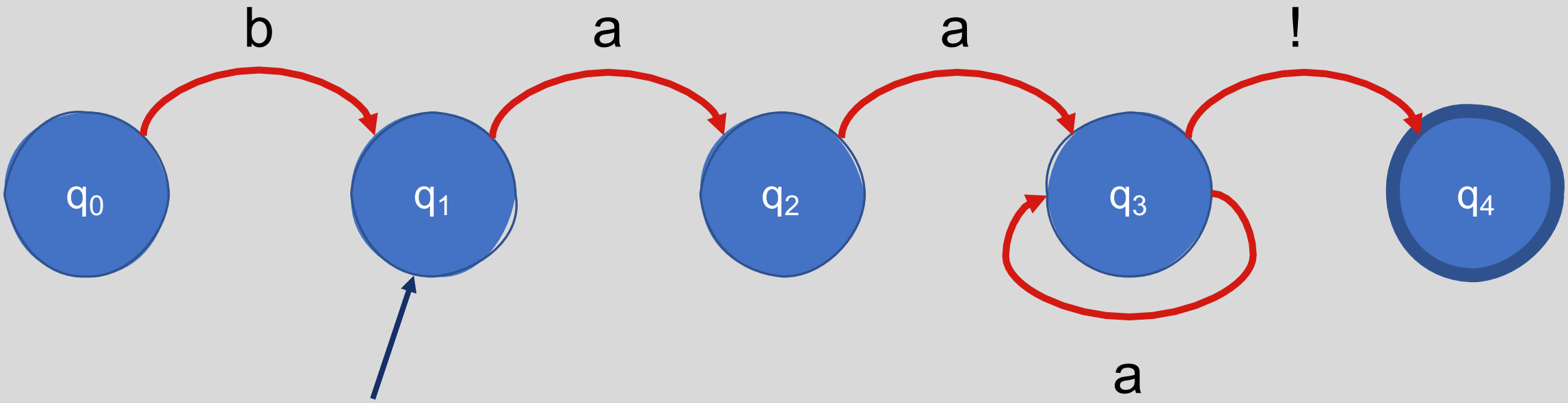


Test String: baa!

Regex that this FSA matches: baa+!

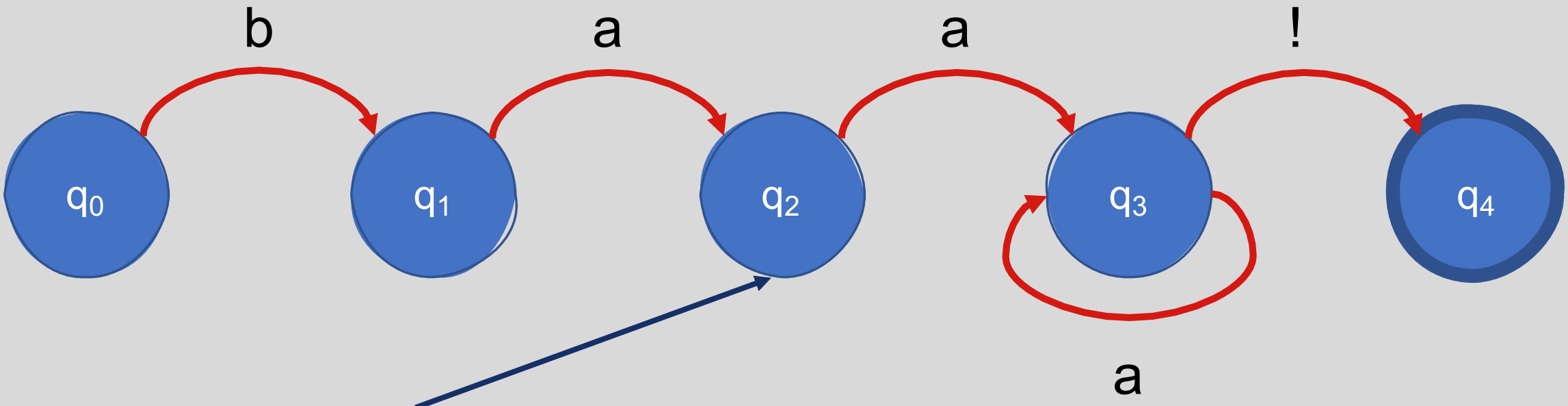


Regex that this FSA matches: baa+!



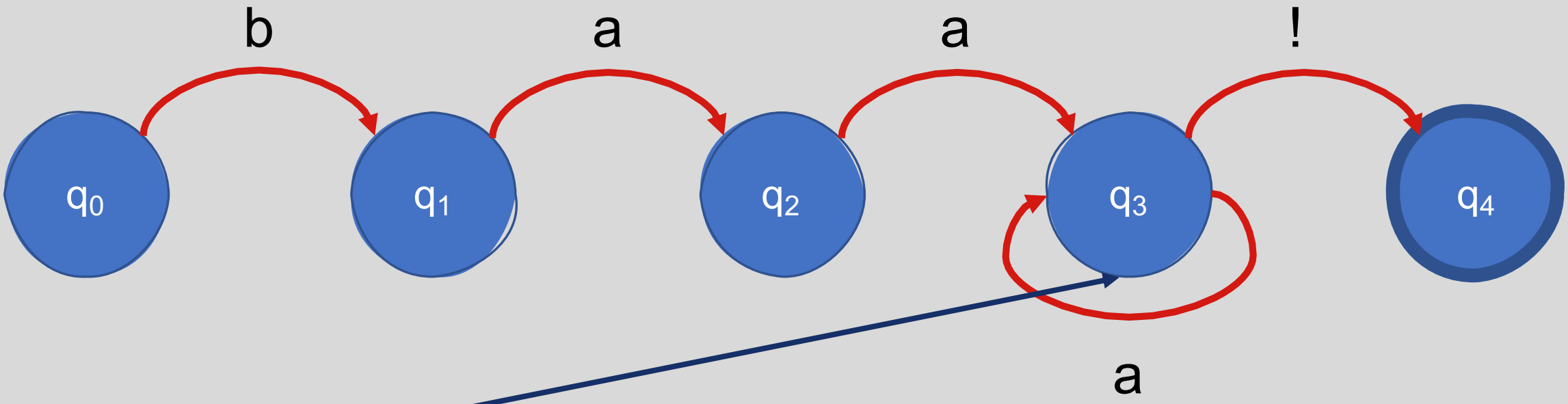
Test String: **b**aa!

Regex that this FSA matches: baa+!



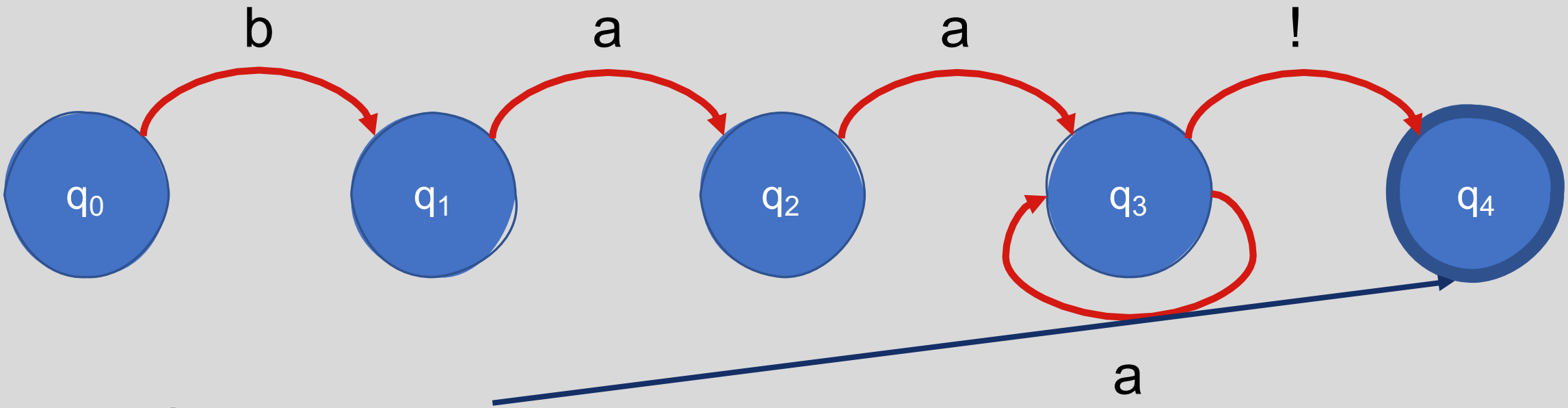
Test String: baa!

Regex that this FSA matches: baa+!



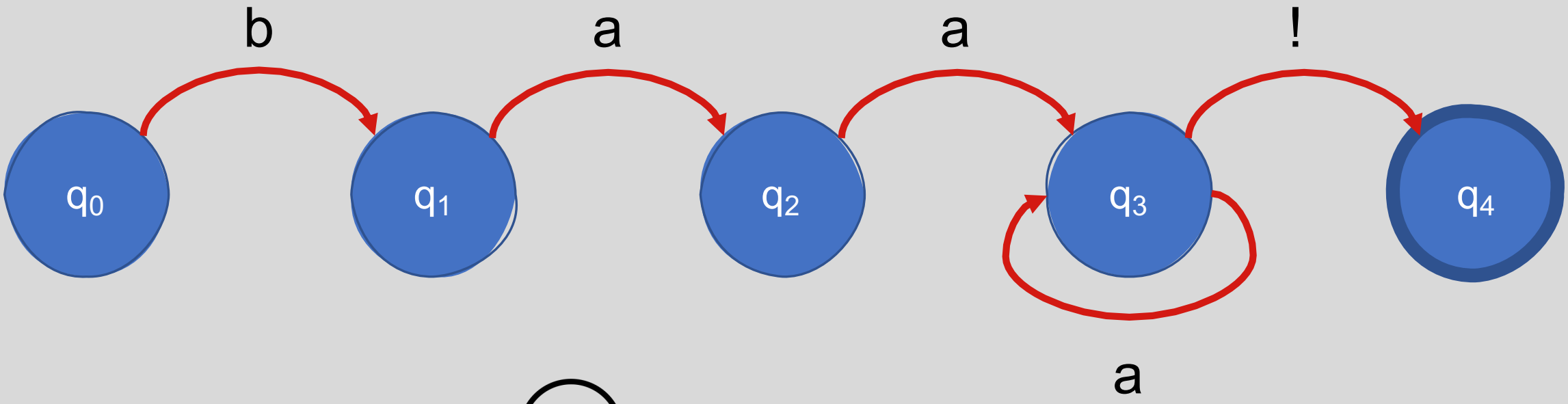
Test String: baa!

Regex that this FSA matches: baa+!



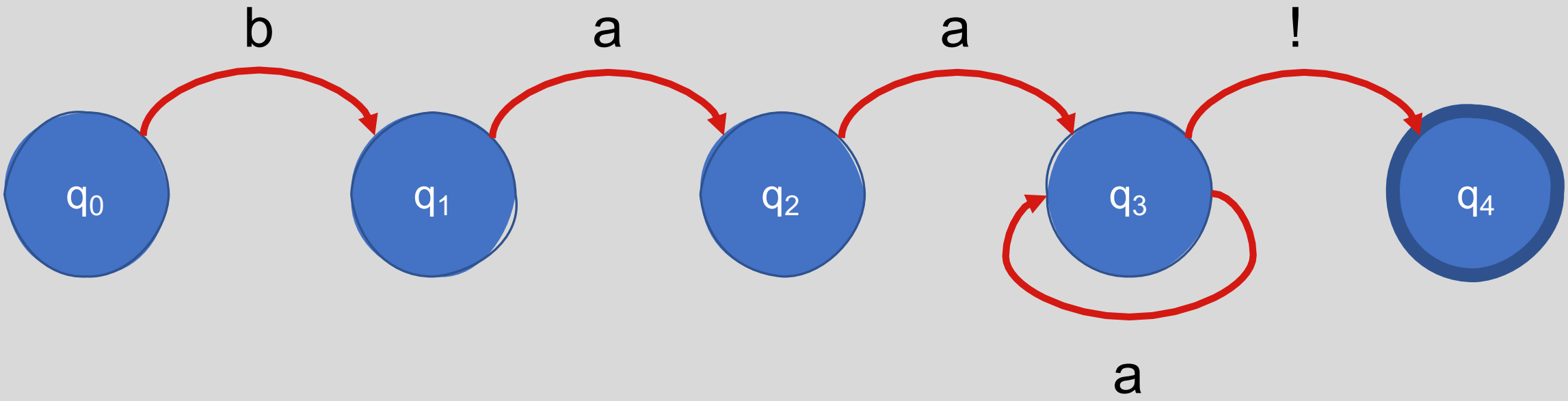
Test String: baa!

Regex that this FSA matches: baa+!



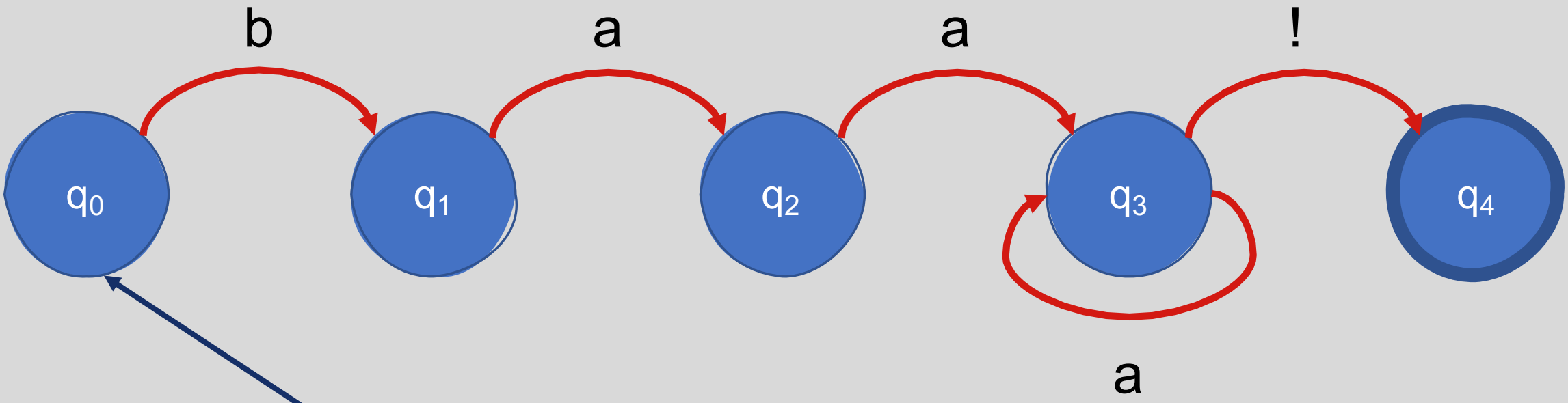
Test String: baa!

Regex that this FSA matches: baa+!



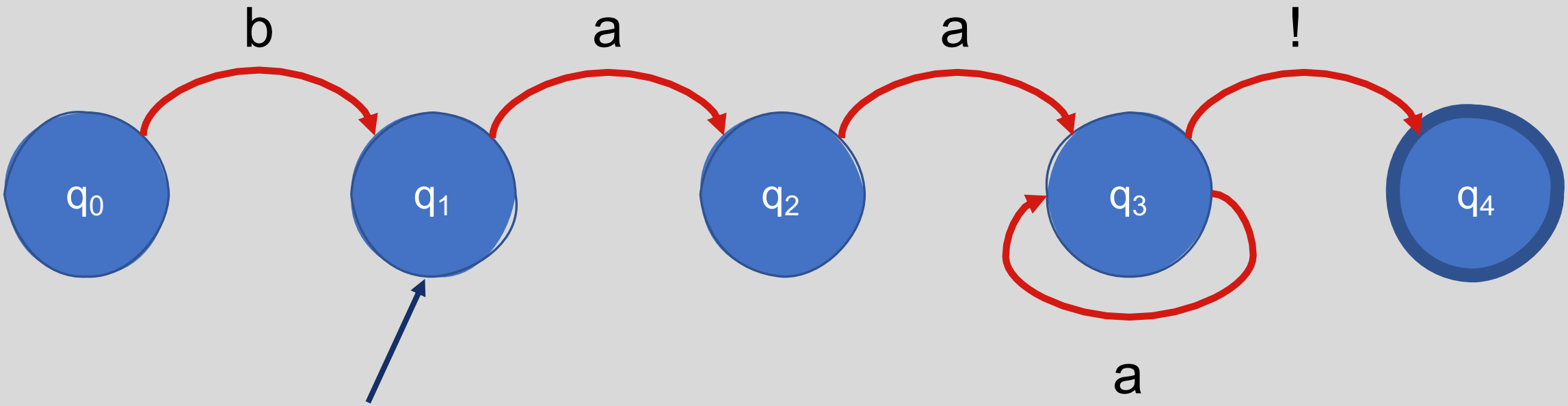
Test String: baabaa!

Regex that this FSA matches: baa+!



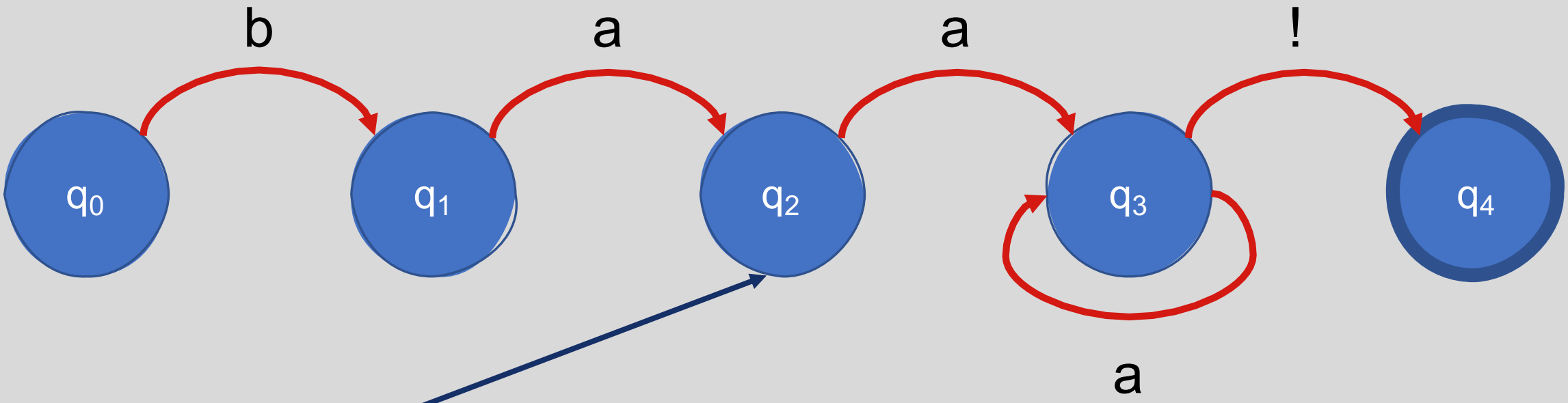
Test String: baabaa!

Regex that this FSA matches: baa+!



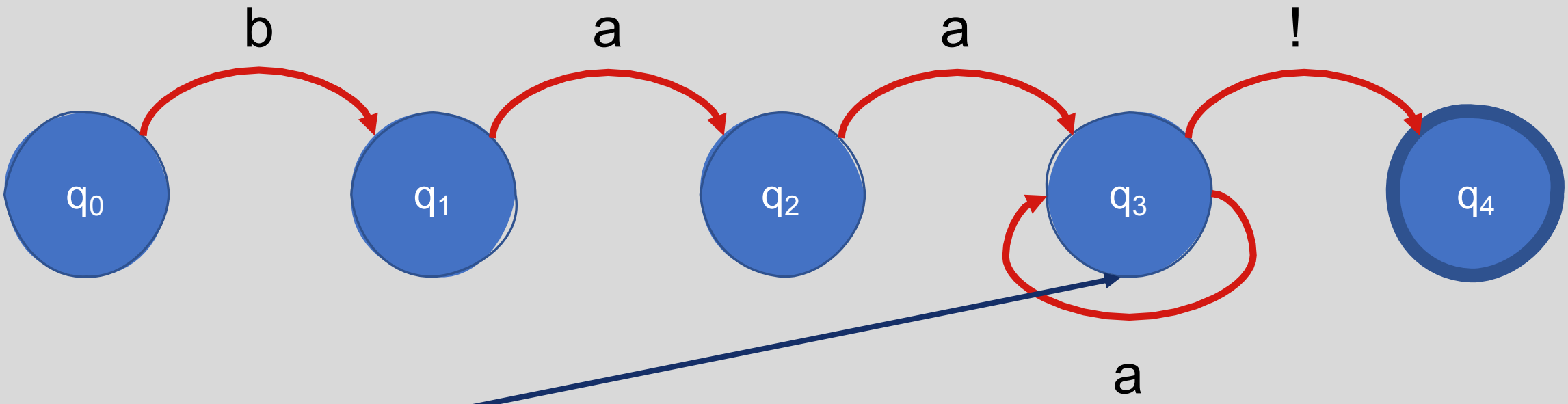
Test String: **b**aabaa!

Regex that this FSA matches: baa+!



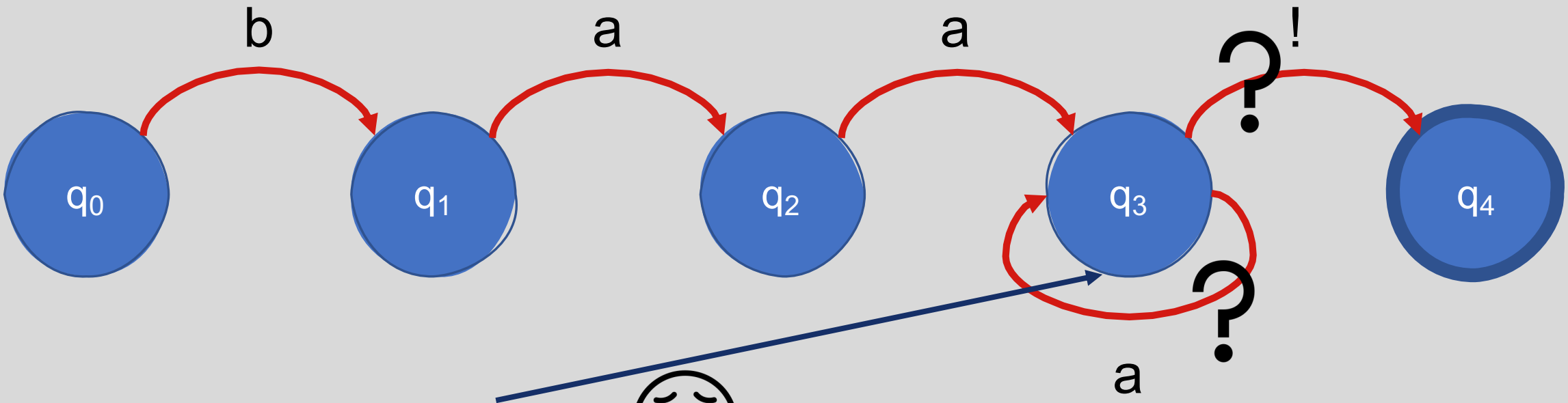
Test String: baabaa!

Regex that this FSA matches: baa+!



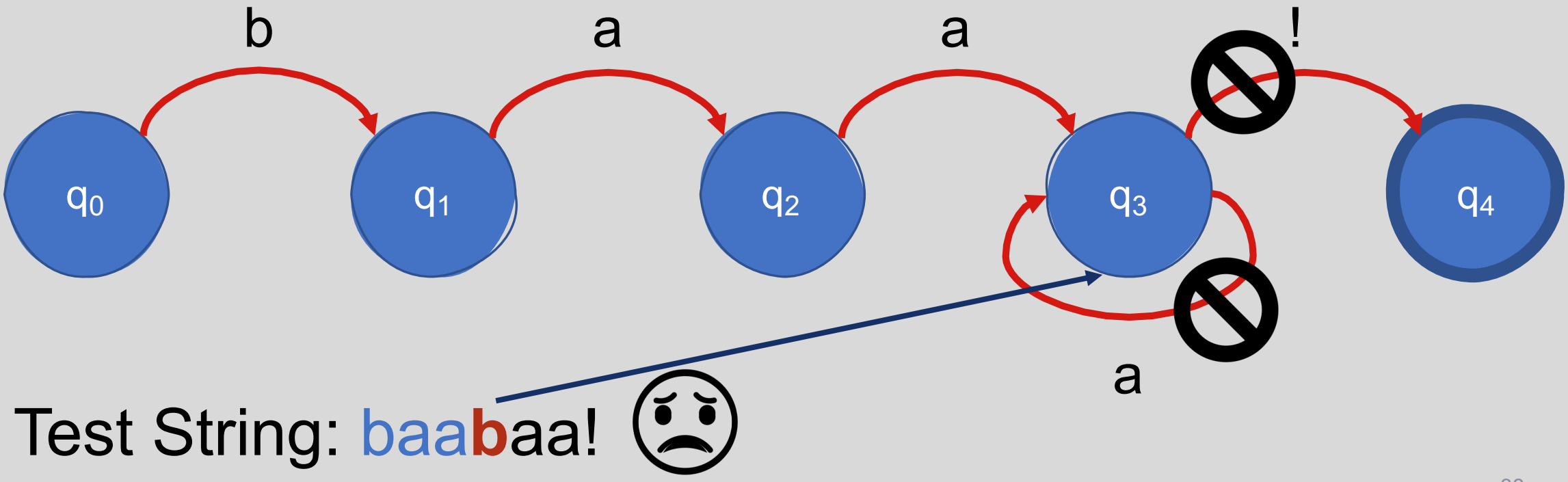
Test String: baabaa!

Regex that this FSA matches: baa+!



Test String: baabaa!

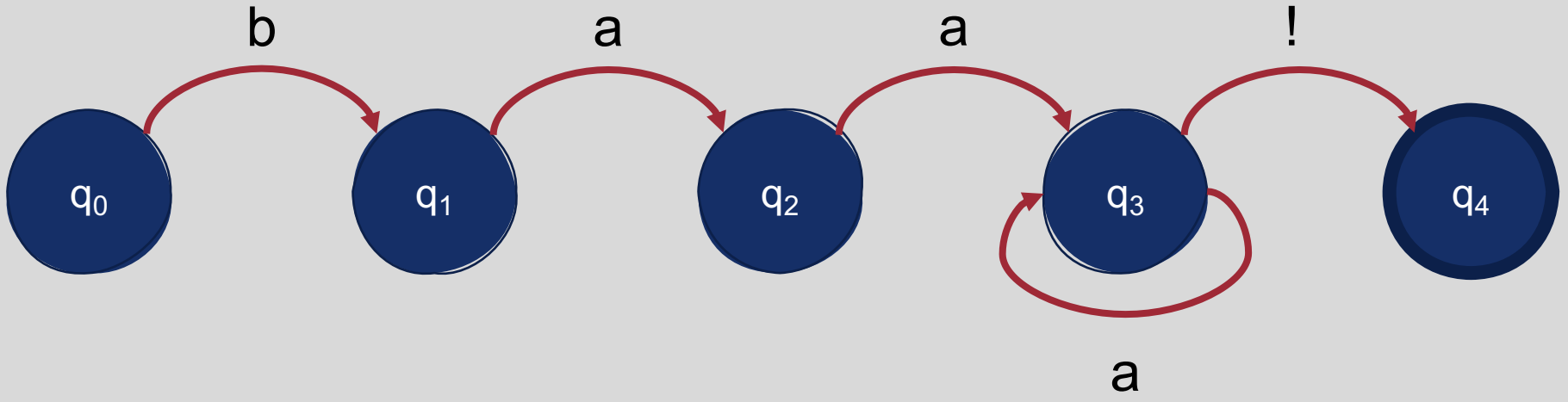
Regex that this FSA matches: baa+!



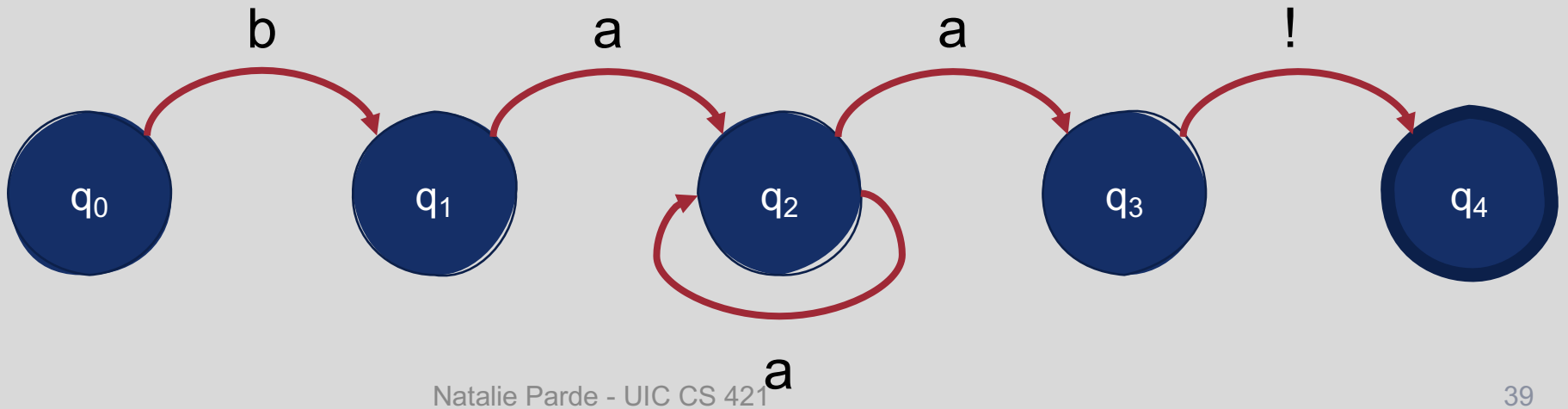
Test String: baabaa!

Note: More than one FSA can correspond to the same regular language!

Test String:
baaa!



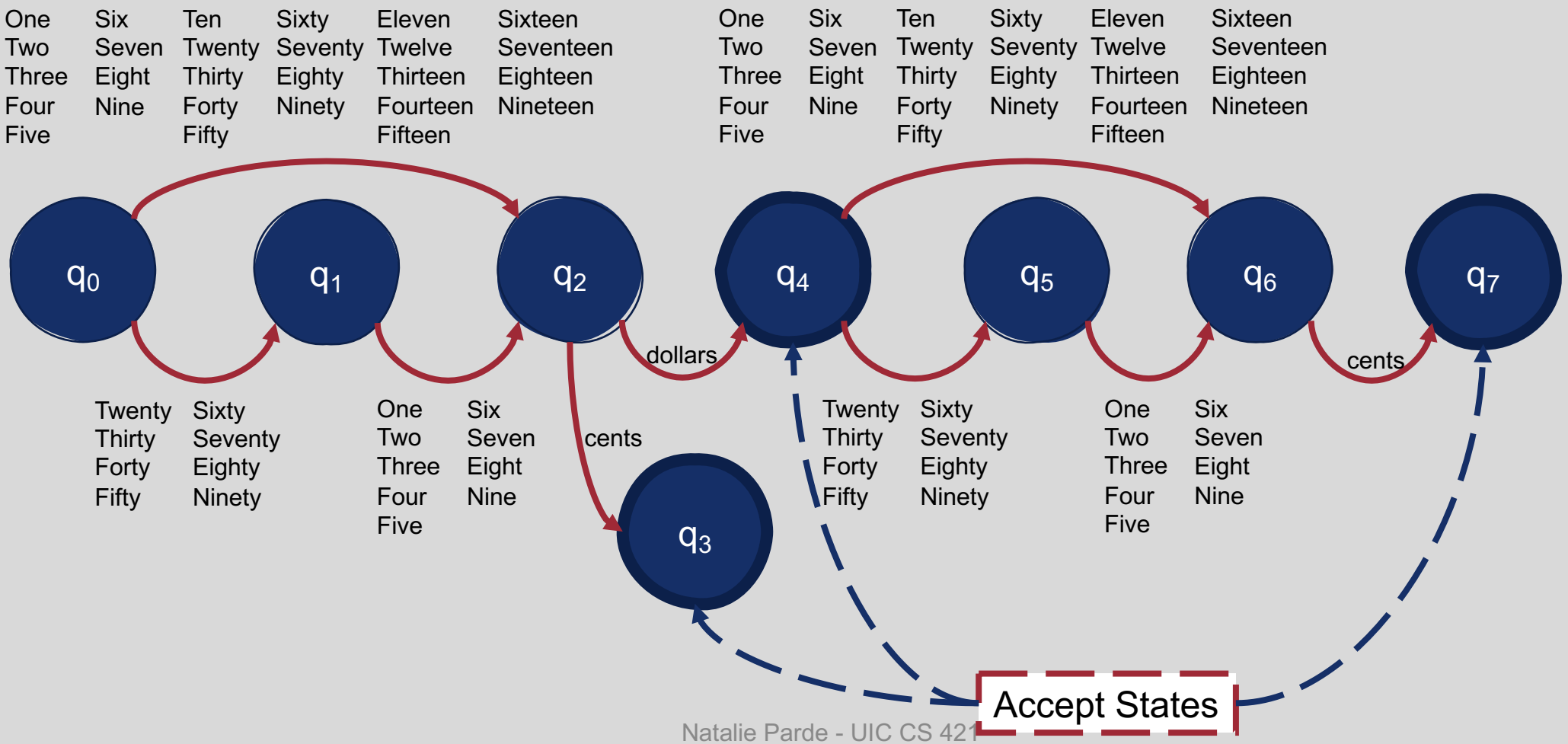
Test String:
baaa!



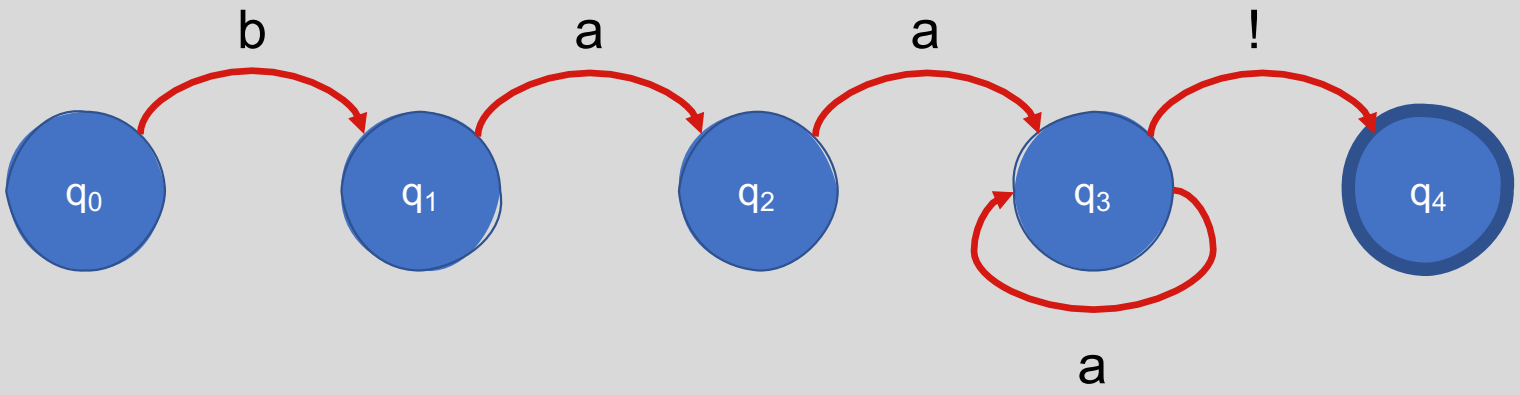
Formal Definition

- A finite state automaton can be specified by enumerating the following properties:
  - The set of states, ***Q***
  - A finite alphabet, ***Σ***
  - A start state, q_0
  - A set of accept/final states, $F \subseteq Q$
  - A transition function or transition matrix between states, **δ(*q*,*i*)**
- δ(q,i): Given a state q∈Q and input i∈Σ, **δ(*q*,*i*) returns a new state q'∈Q**.

Example: FSA for Dollar Amounts



State transitions in FSAs can be represented using tables.

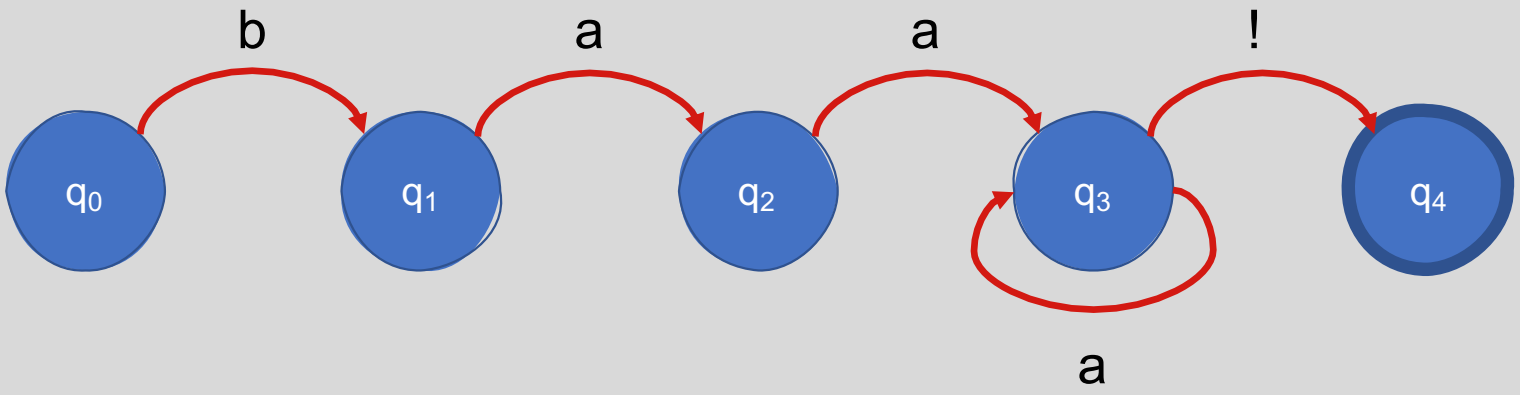


Next Item in Sequence

Currently in State

	b	a	!	<end>
q_0				
q_1				
q_2				
q_3				
q_4				

State transitions in FSAs can be represented using tables.



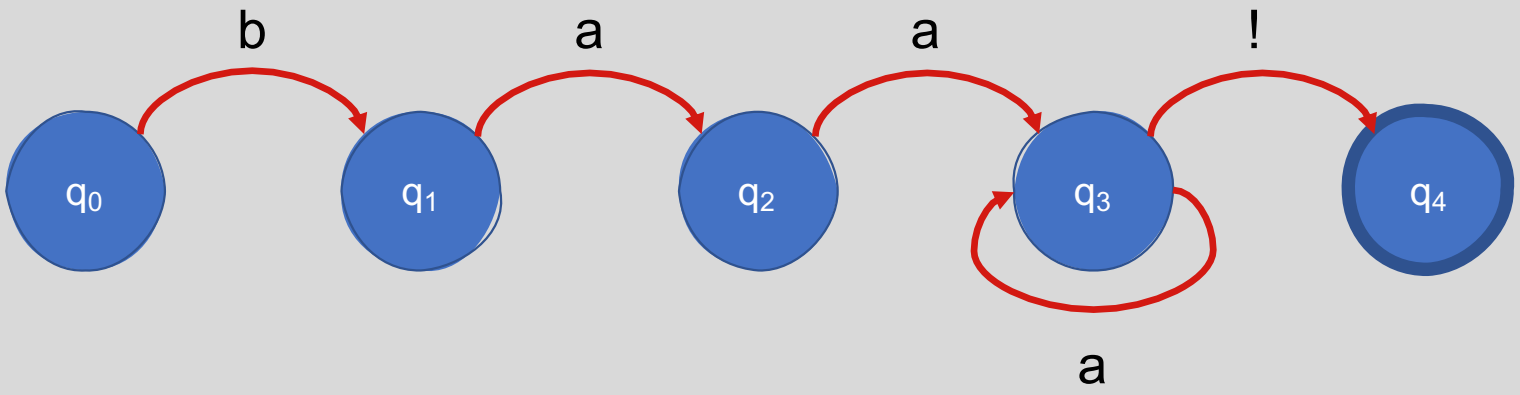
Next Item in Sequence

Currently in State

	b	a	!	<end>
q_0	q_1			
q_1				
q_2				
q_3				
q_4				

Go to State

State transitions in FSAs can be represented using tables.



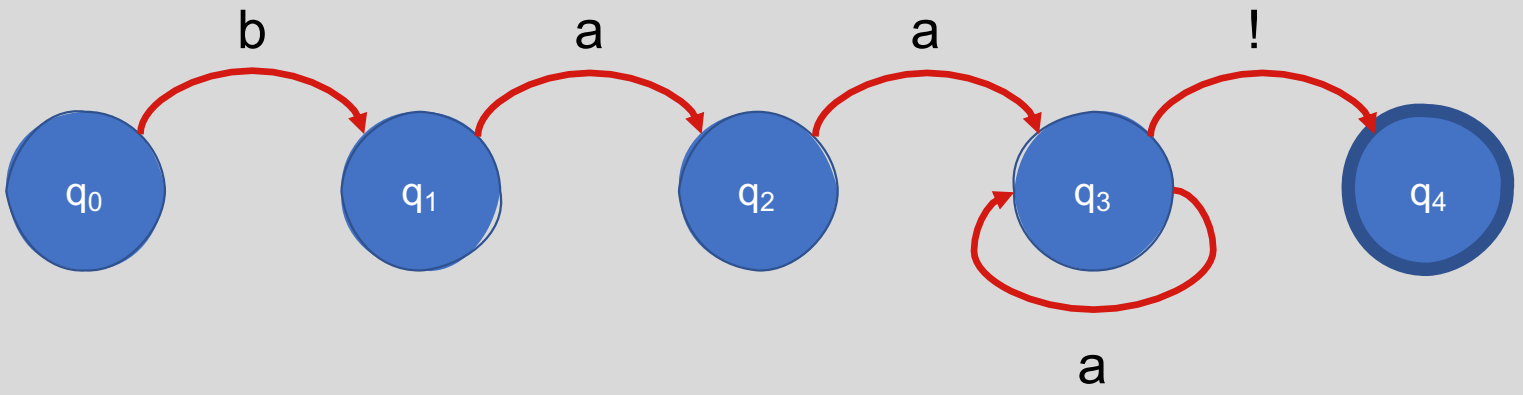
Next Item in Sequence

Currently in State

	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1				
q_2				
q_3				
q_4				

Go to State

State transitions in FSAs can be represented using tables.

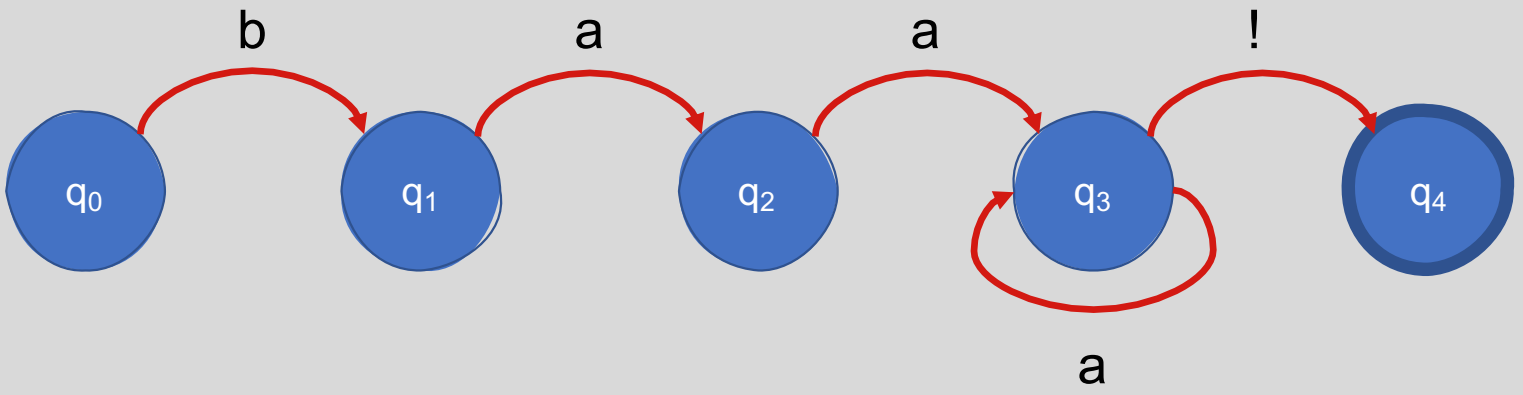


Next Item in Sequence

Currently in State	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1	☹	q_2		
q_2				
q_3				
q_4				

Go to State

State transitions in FSAs can be represented using tables.

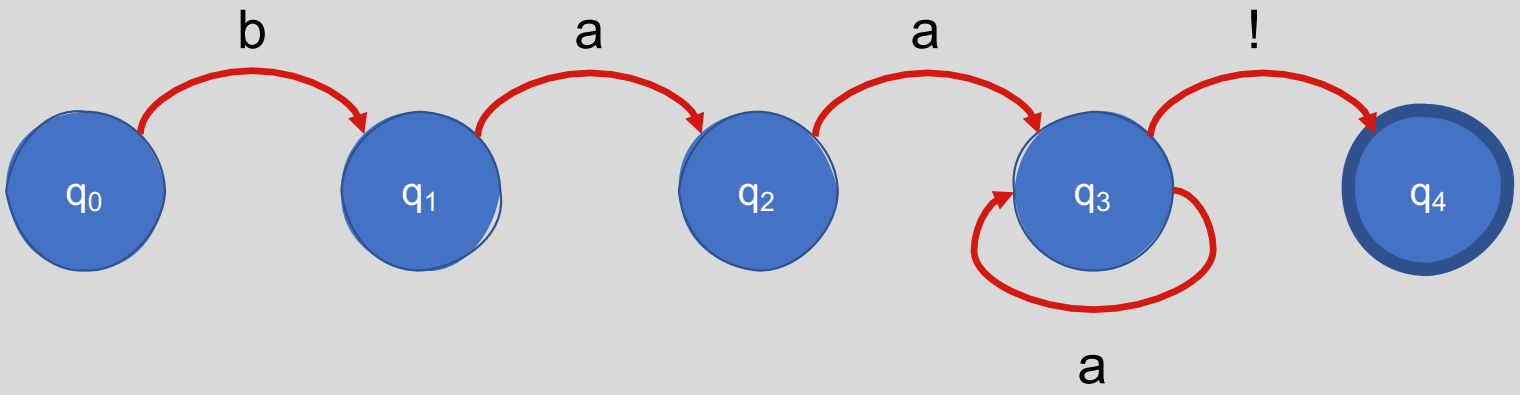


Next Item in Sequence

Currently in State	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1	☹	q_2	☹	☹
q_2	☹	q_3		
q_3				
q_4				

Go to State

State transitions in FSAs can be represented using tables.

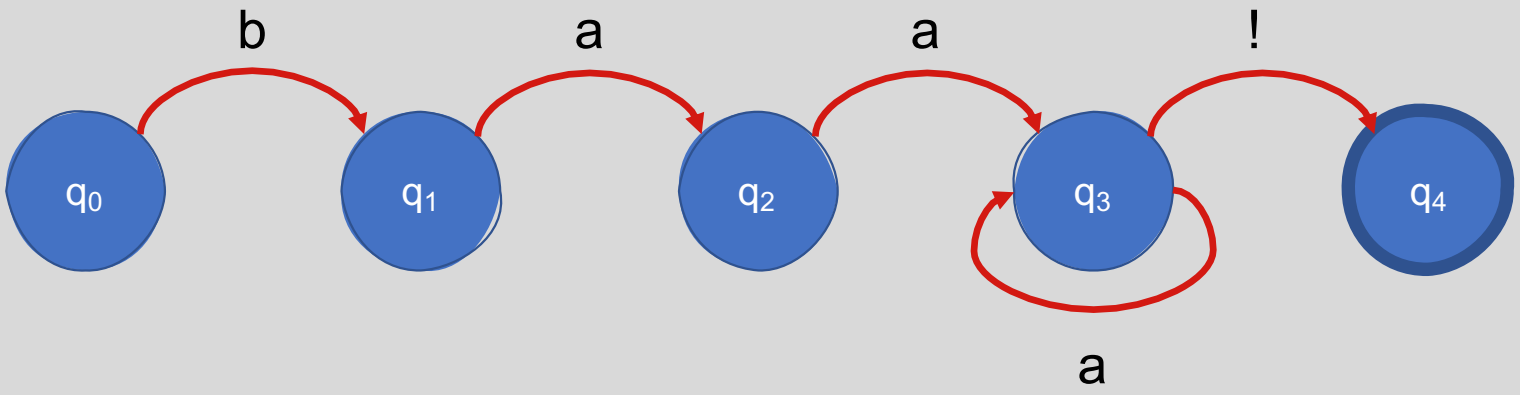


Next Item in Sequence

Currently in State	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1	☹	q_2	☹	☹
q_2	☹	q_3	☹	☹
q_3	☹	q_3		
q_4				

Go to State

State transitions in FSAs can be represented using tables.



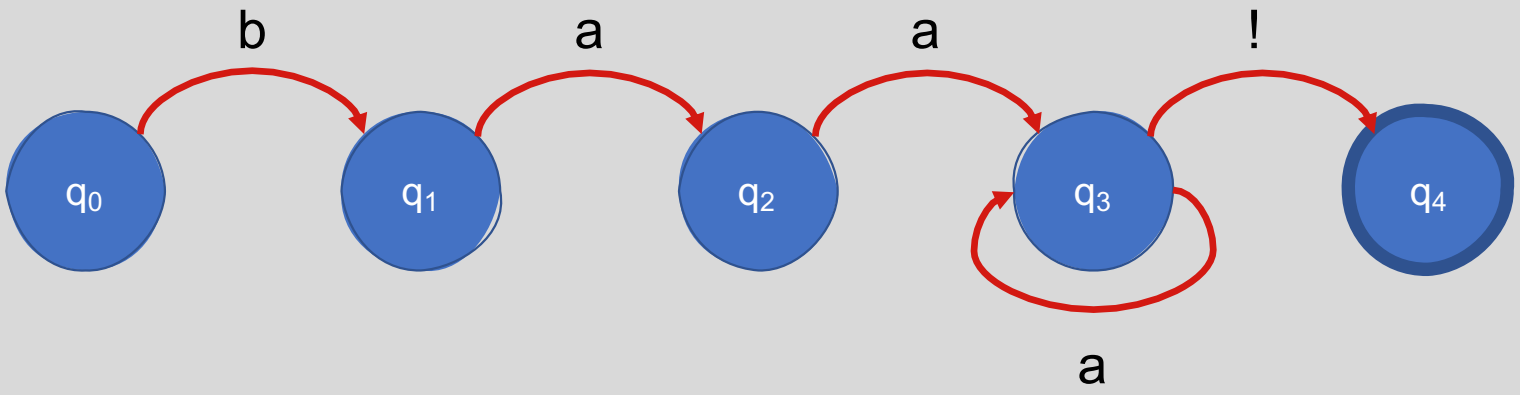
Next Item in Sequence

Currently in State

	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1	☹	q_2	☹	☹
q_2	☹	q_3	☹	☹
q_3	☹	q_3	q_4	
q_4				

Go to State

State transitions in FSAs can be represented using tables.



Next Item in Sequence

Currently in State

	b	a	!	<end>
q_0	q_1	☹	☹	☹
q_1	☹	q_2	☹	☹
q_2	☹	q_3	☹	☹
q_3	☹	q_3	q_4	☹
q_4	☹	☹	☹	☺

Accept!

State transition tables simplify the process of determining whether your input will be accepted by the FSA.

- For a given sequence of items to match, **begin in the start state** with the first item in the sequence
- **Consult the table** …is a transition to any other state permissible with the current item?
- If so, **move to the state indicated by the table**
- If you make it to the end of your sequence and to a final state, **accept**

Formal Algorithm

```
index ← beginning of sequence
current_state ← initial state of FSA
loop:
        if end of sequence has been reached:
                if current_state is an accept state:
                        return accept
                else:
                        return reject
        else if transition_table[current_state, sequence[index]] is empty:
                return reject
        else:
                current_state ← transition_table[current_state, sequence[index]]
                index ← index + 1
end
```

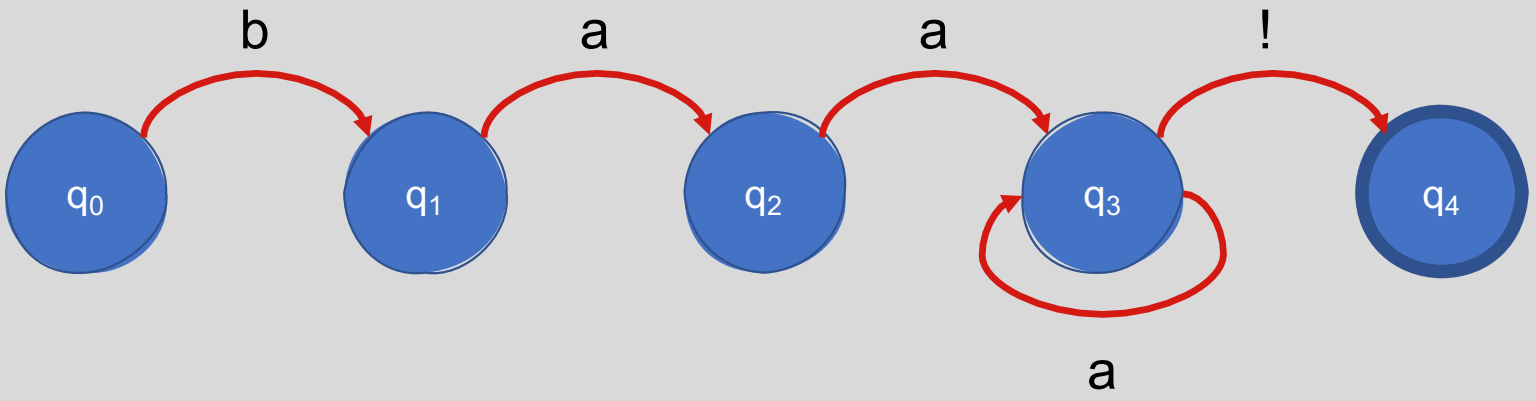
Deterministic vs. Non-Deterministic FSAs

Deterministic FSA: At each point in processing a sequence, there is one unique thing to do (no choices!)

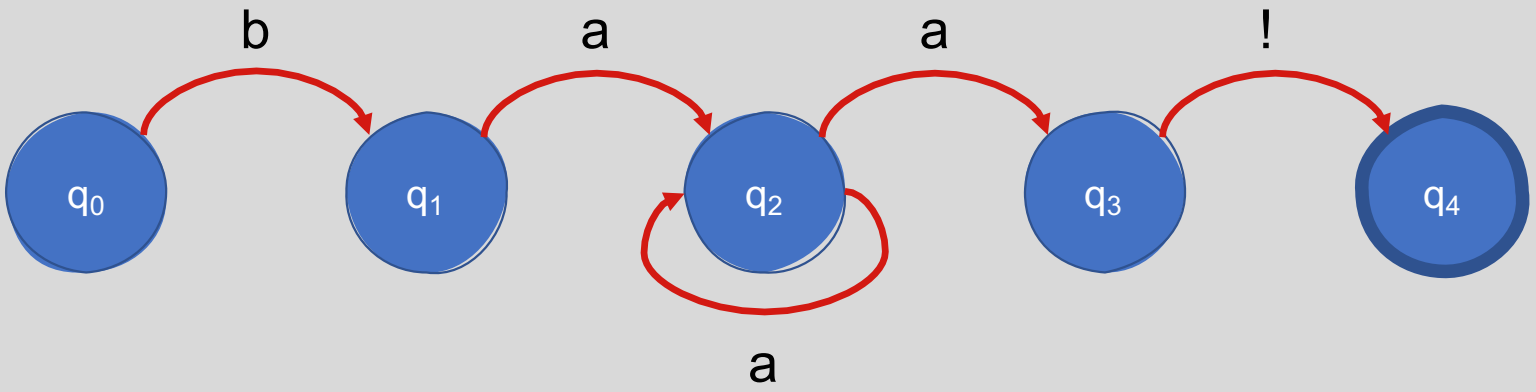
Non-Deterministic FSA: At one or more points in processing a sequence, there are multiple permissible next steps (choices!)

Deterministic or Non-Deterministic?

?

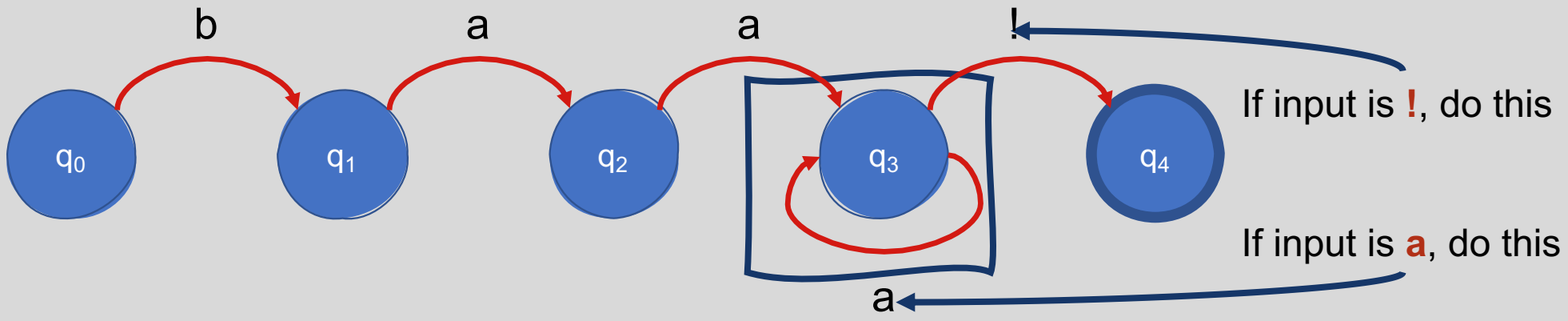


?

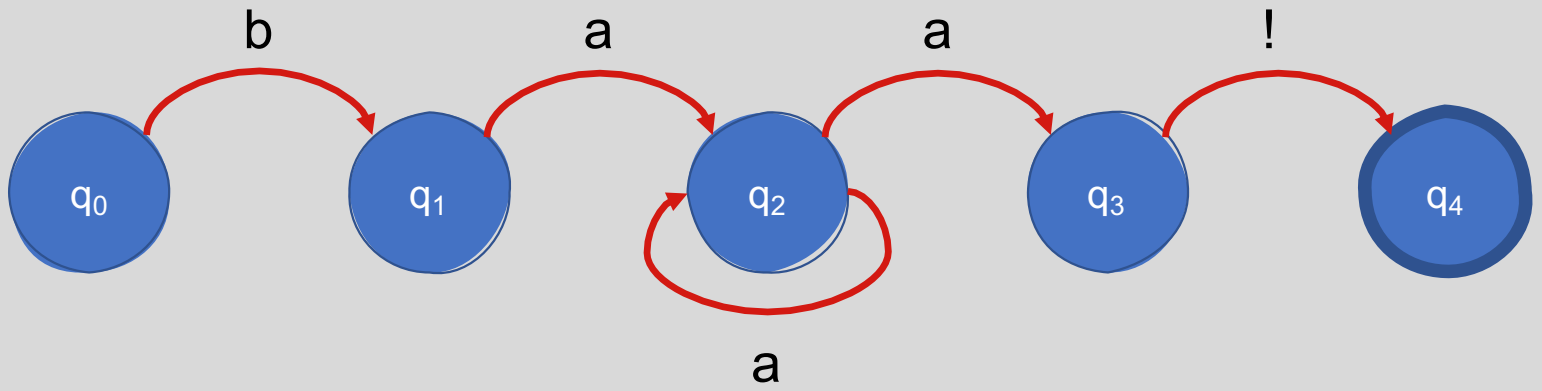


Deterministic or Non-Deterministic?

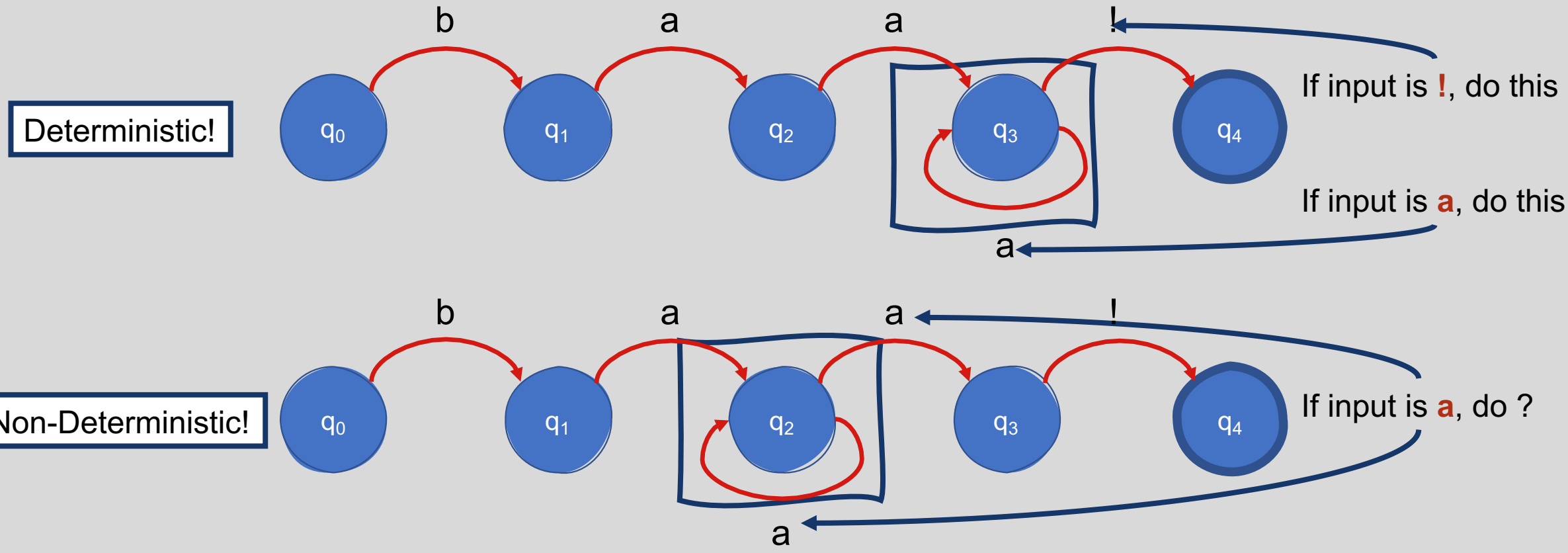
Deterministic!



?



Deterministic or Non-Deterministic?



Every non-deterministic FSA can be converted to a deterministic FSA.

- This means that both are equally powerful!
- Deterministic FSAs can accept as many languages as non-deterministic ones

Non-Deterministic FSAs: How to check for input acceptance?

- Two approaches:
  1. Convert the non-deterministic FSA to a deterministic FSA and then check that version
  2. Manage the process as a state-space search

Non-Deterministic FSA Search Assumptions

There exists at least one path through the FSA for an item that is part of the language defined by the machine

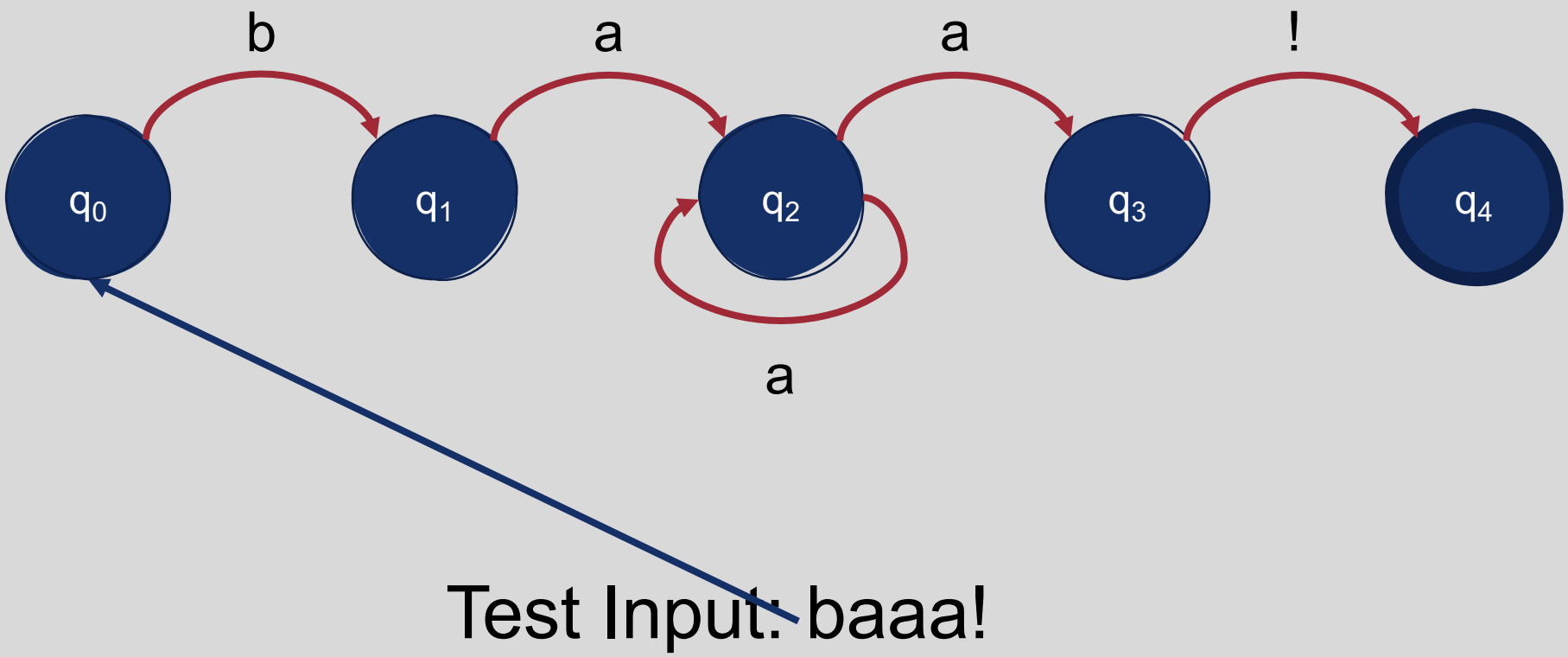
Not all paths directed through the FSA for an accept item lead to an accept state

No paths through the FSA lead to an accept state for an item not in the language

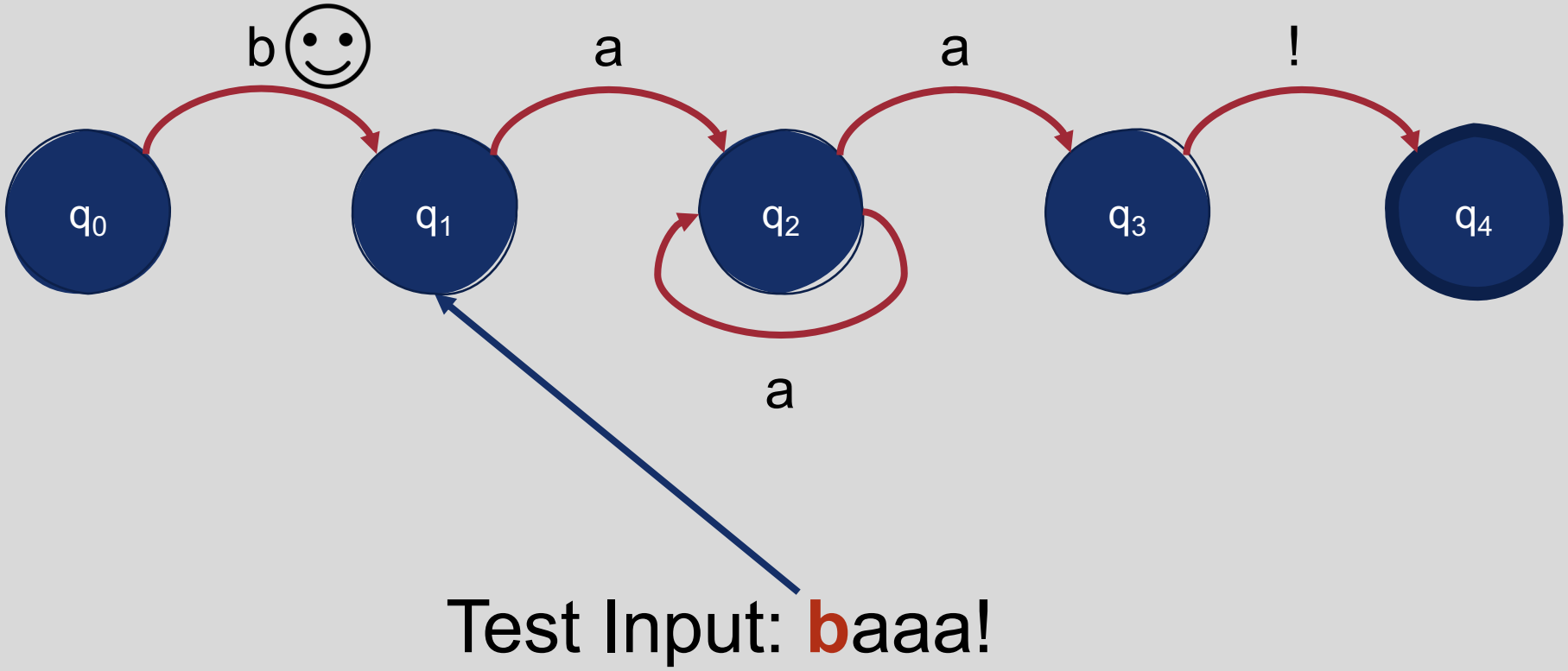
Non-Deterministic FSA Search Assumptions

- **Success:** Path is found for a given item that ends in an accept
- **Failure:** All possible paths for a given item lead to failure

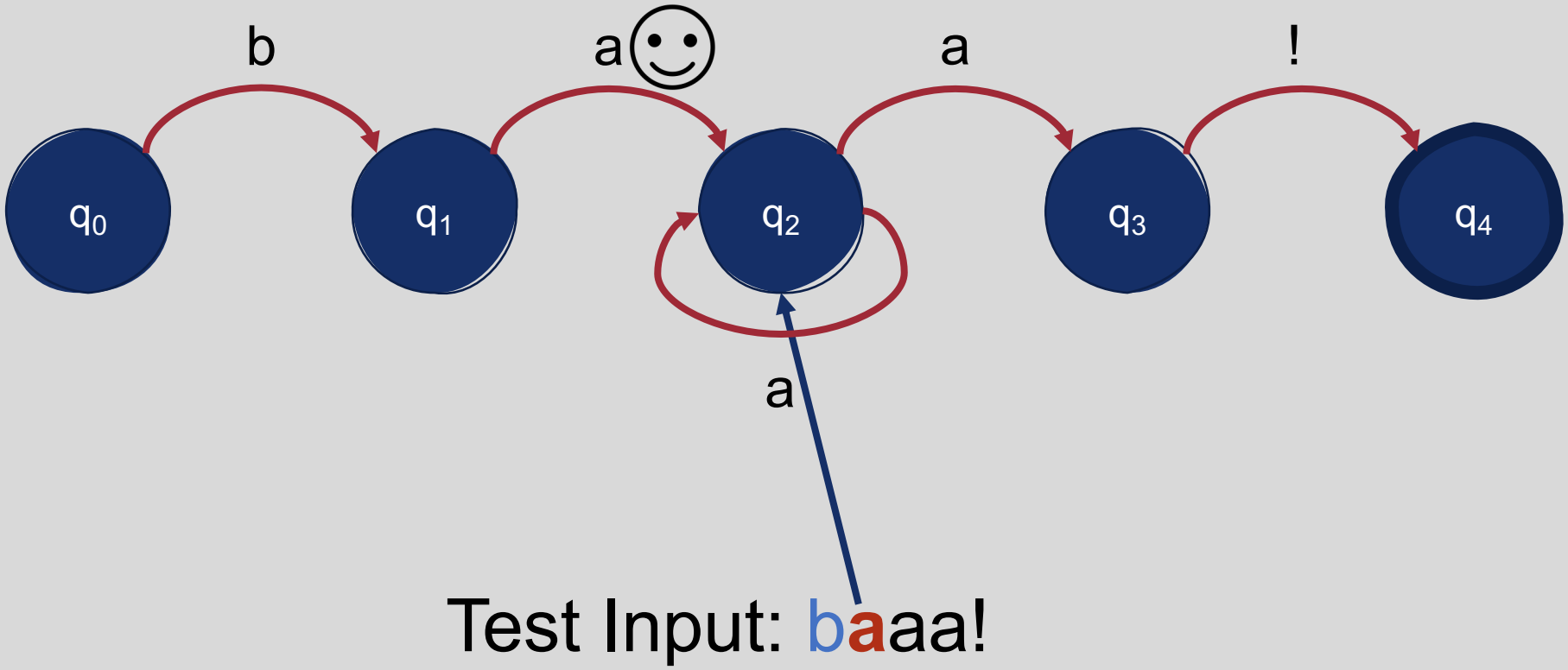
Example: Non-Deterministic FSA Search



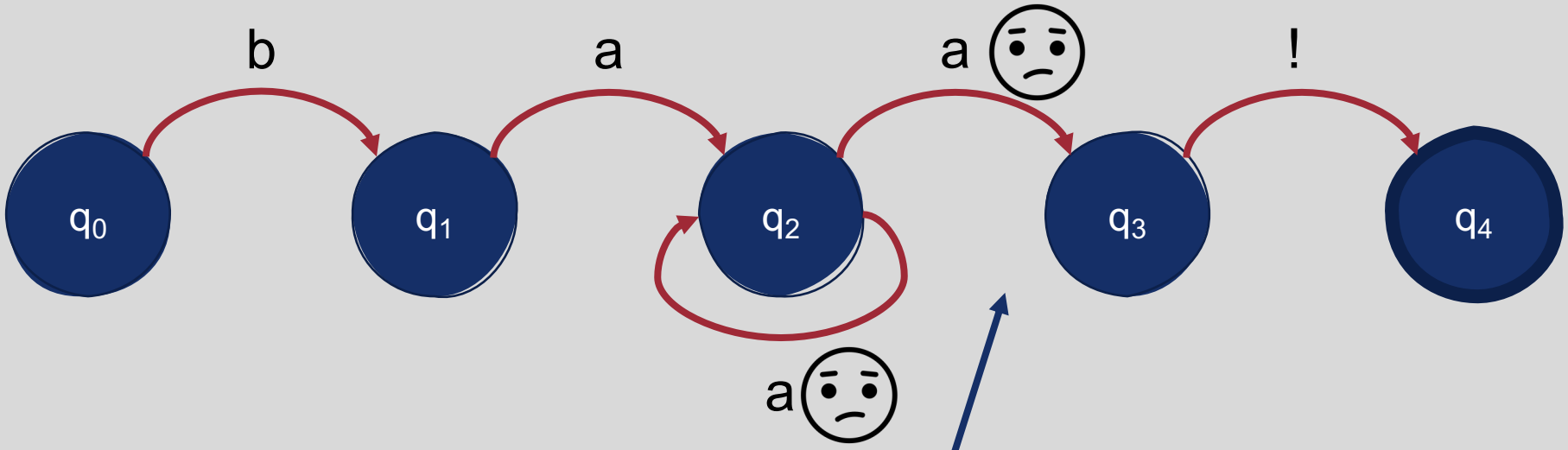
Example: Non-Deterministic FSA Search



Example: Non-Deterministic FSA Search

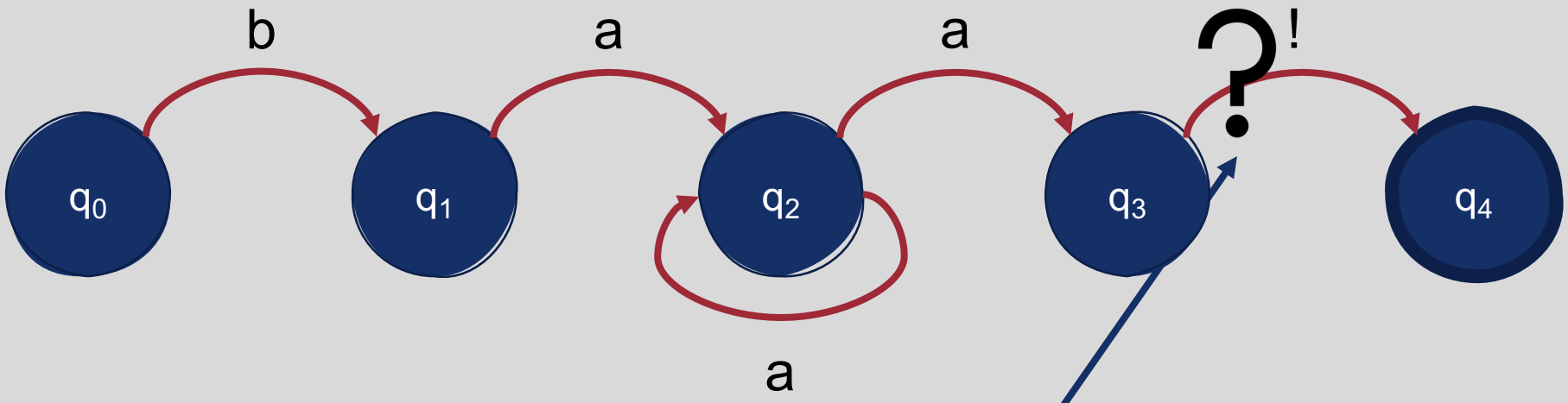


Example: Non-Deterministic FSA Search



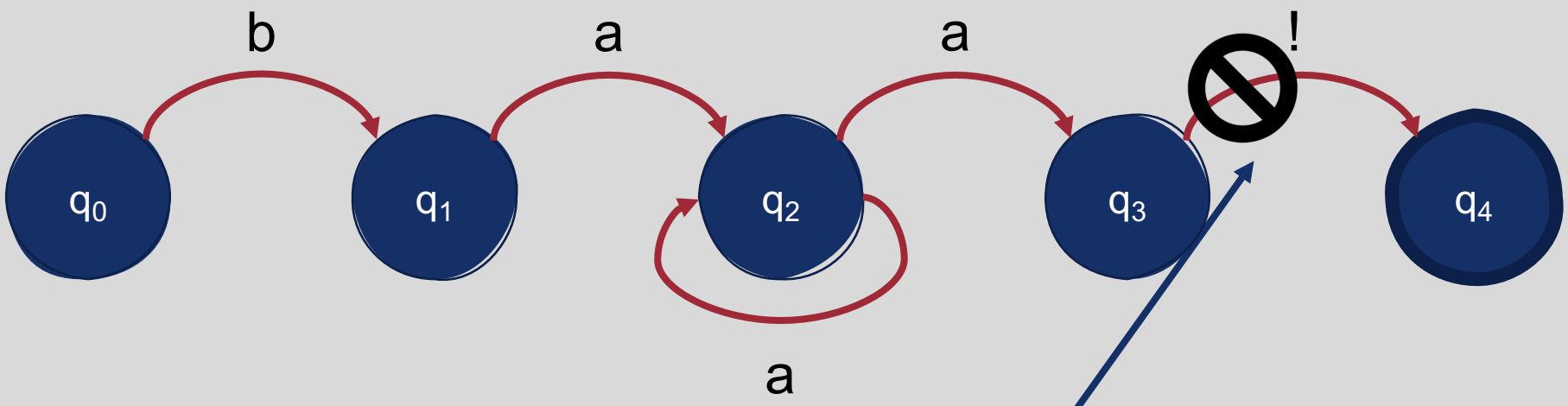
Test Input: baaa!

Example: Non-Deterministic FSA Search



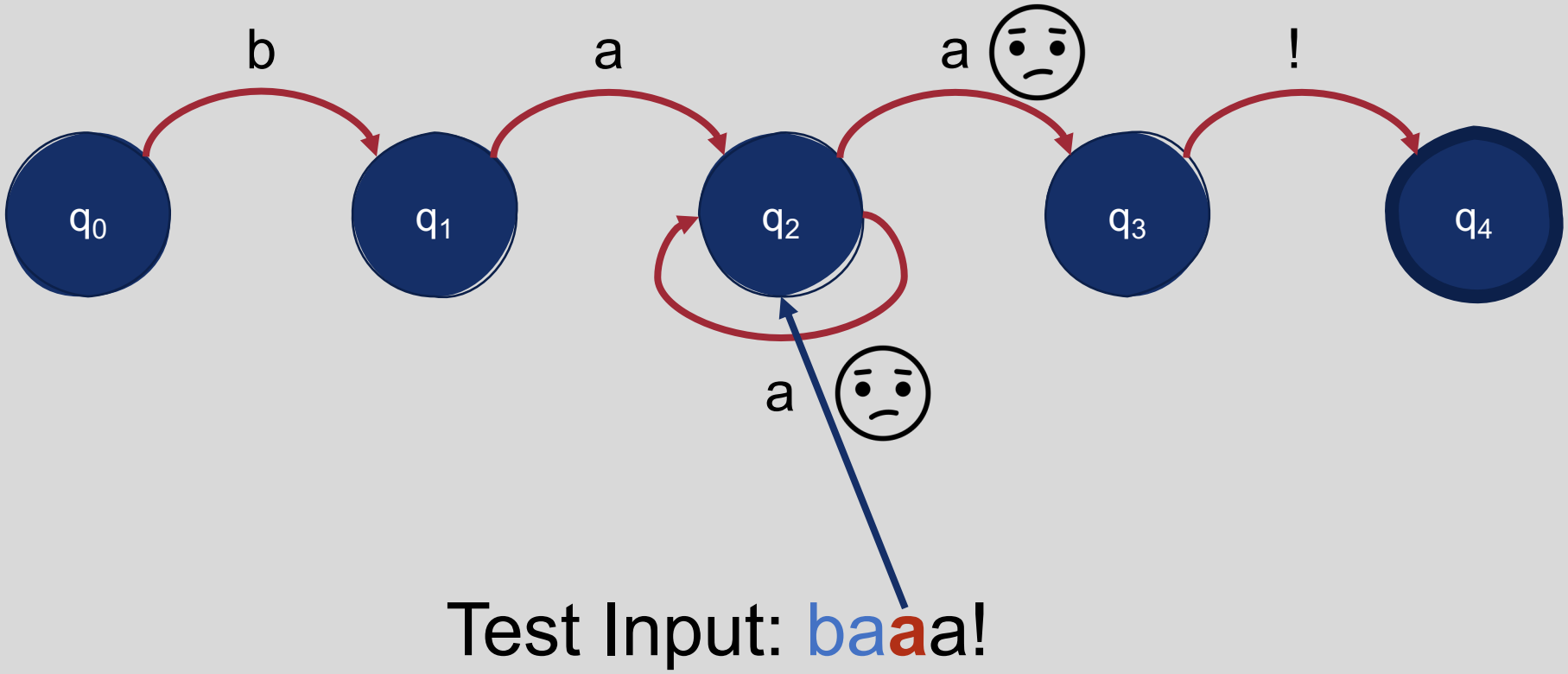
Test Input: baaa!

Example: Non-Deterministic FSA Search

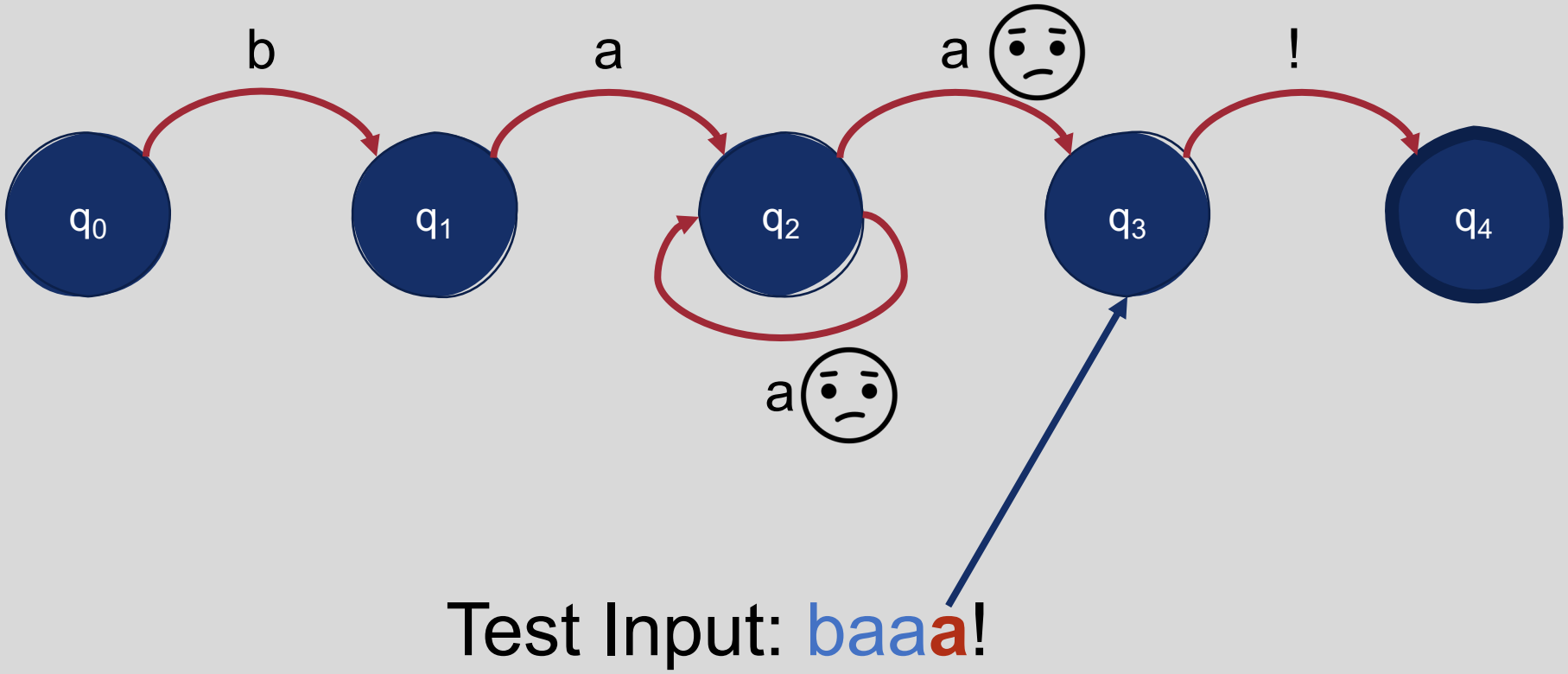


Test Input: baaa! ☹

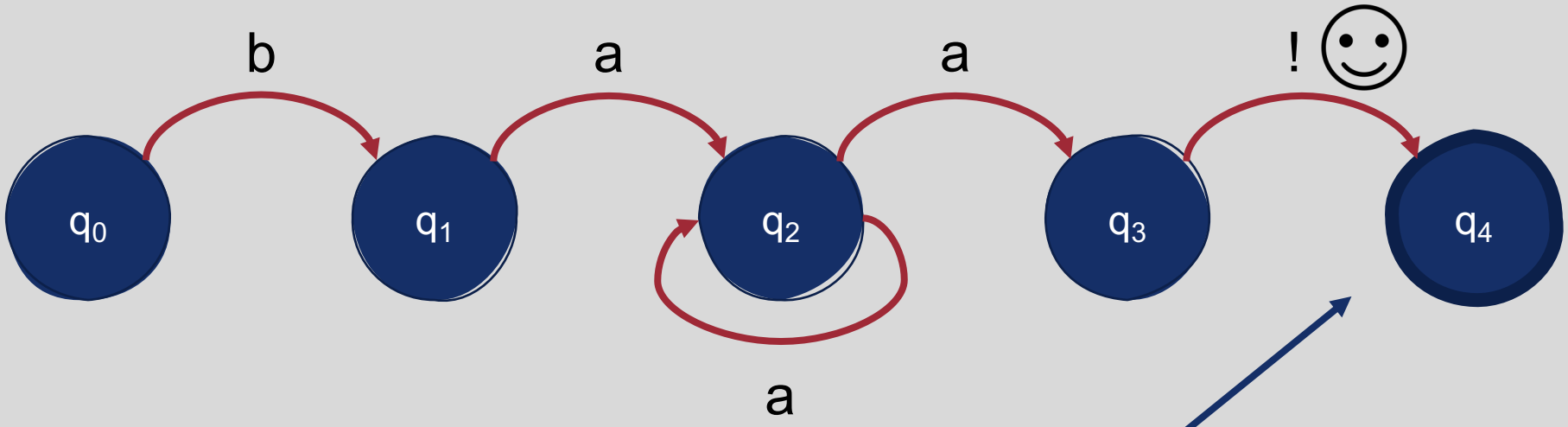
Example: Non-Deterministic FSA Search



Example: Non-Deterministic FSA Search

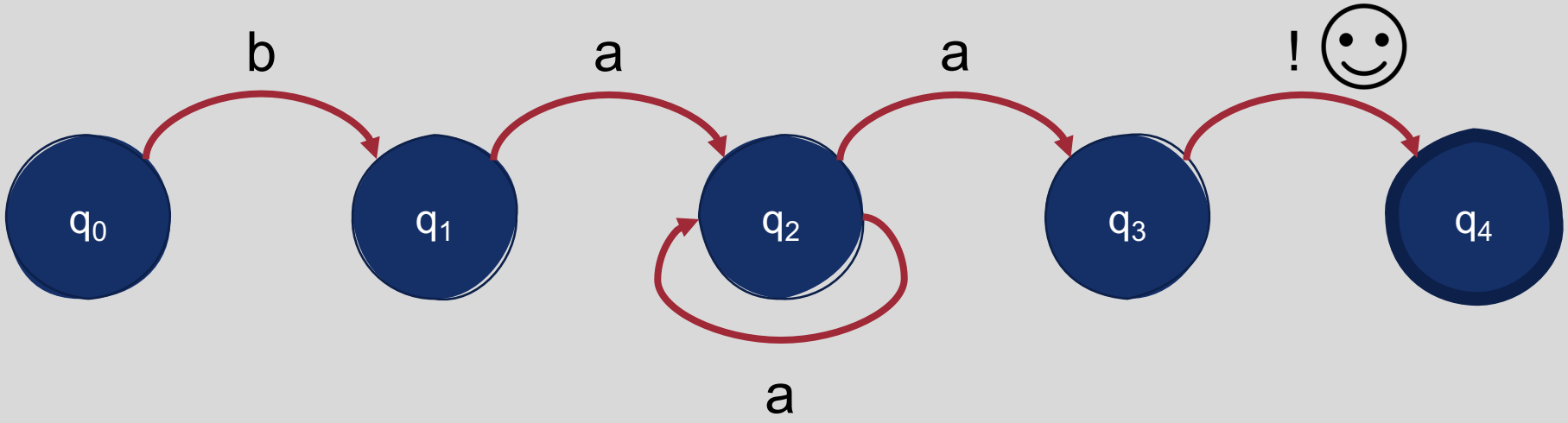


Example: Non-Deterministic FSA Search



Test Input: baaa!

Example: Non-Deterministic FSA Search



Test Input: baaa! 

Non-Deterministic FSA Search

- States in the search space are pairings of sequence indices and states in the FSA
- By keeping track of which states have and have not been explored, we can systematically explore all the paths through an FSA given an input

Compositional FSAs

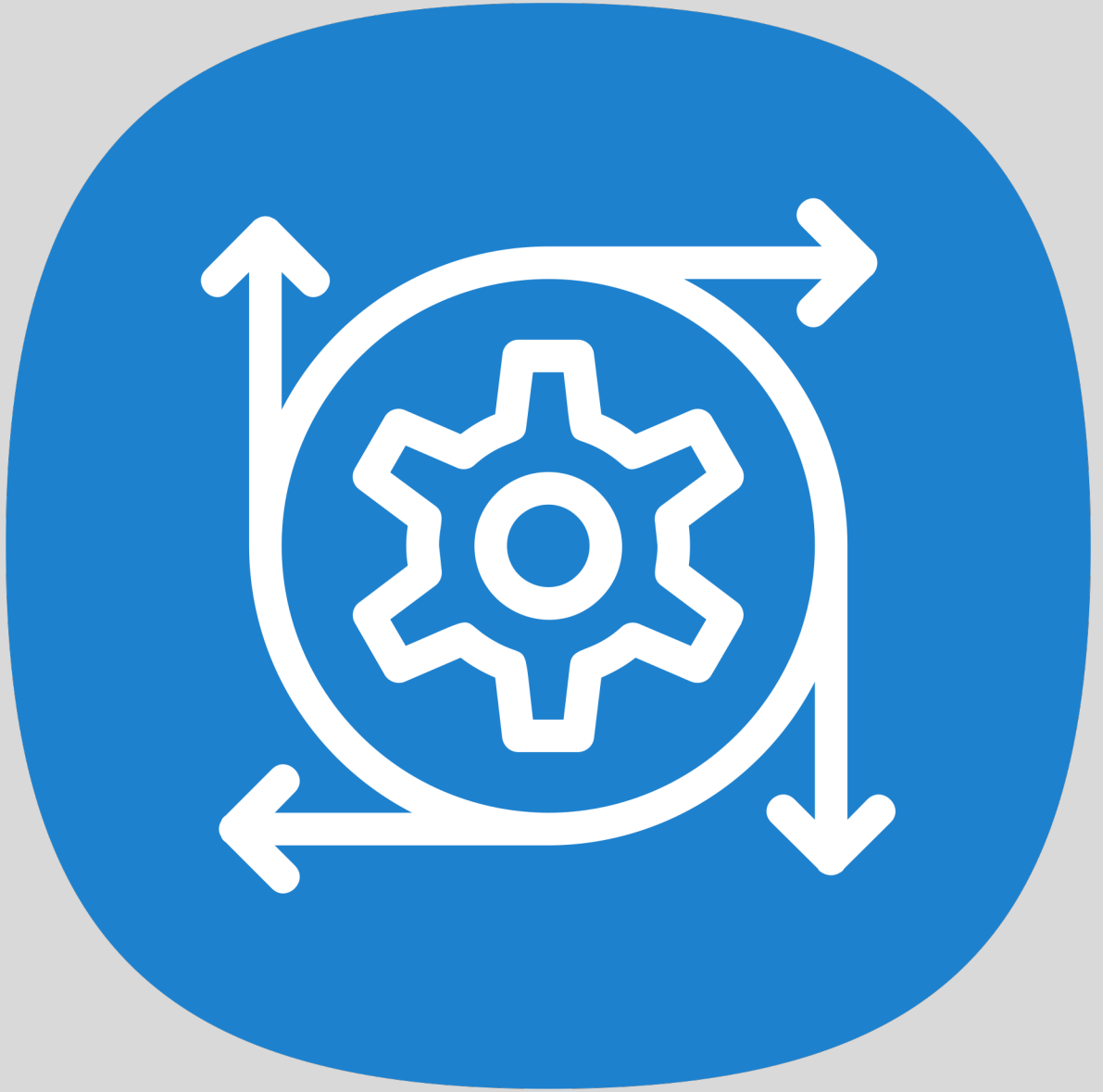
- You can apply set operations to any FSA
  - Union
  - Concatenation
  - Negation
    - For non-deterministic FSAs, first convert to a deterministic FSA
  - Intersection
- To do so, you may need to utilize an ϵ transition
  - ϵ transition: Move from one state to another without consuming an item from the input sequence

Summary: Finite State Automata

- FSAs are computational models that describe regular languages
- To determine whether an input item is a member of an FSA's language, you can process it sequentially from the start to (hopefully) the final state
- State transitions in FSAs can be represented using tables
- FSAs can be either deterministic or non-deterministic

What's the next natural step?

Finite state transducers



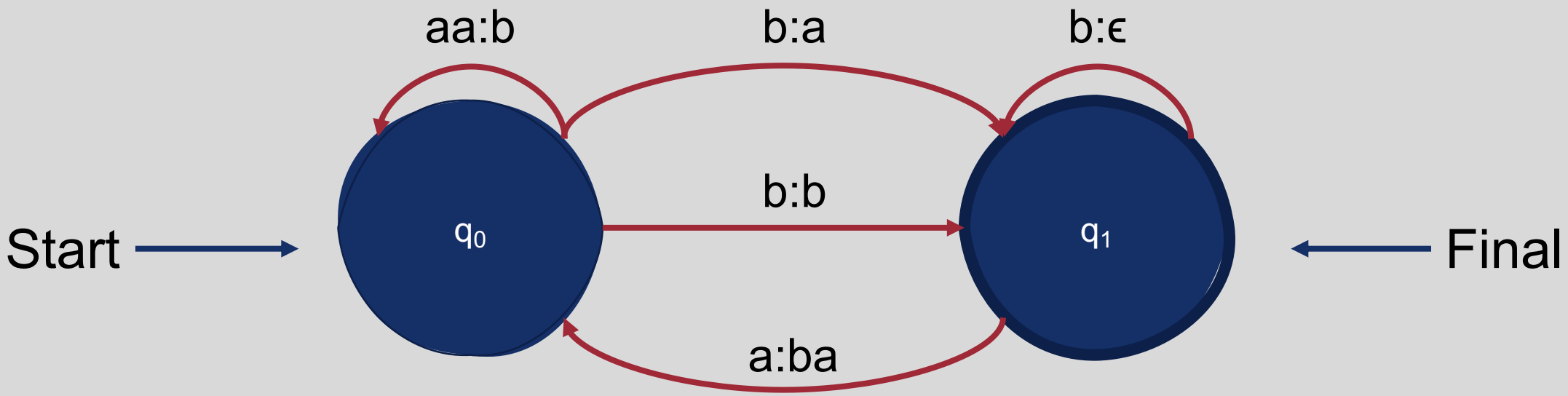
What are finite state transducers?

Finite State Transducer (FST): A type of FSA that describes mappings between two sets of items

This means that FSTs recognize or generate pairs of items

FSAs can be converted to FSTs by labeling each arc with two items (e.g., **a**:**b** for an input of **a** and and an output of **b**)

Example: Simple FST



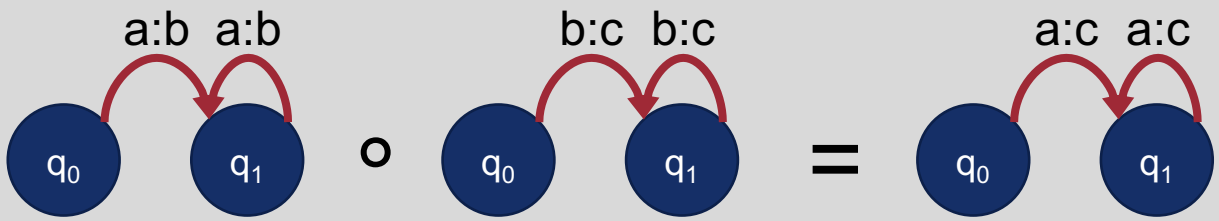
Formal Definition

- A finite state transducer can be specified by enumerating the following properties:
  - The set of states, ***Q***
  - A finite input alphabet, ***Σ***
  - A finite output alphabet, ***Δ***
  - A start state, q_0
  - A set of accept/final states, $F \subseteq Q$
  - A transition function or transition matrix between states, **δ(*q*,*i*)**
  - An output function giving the set of possible outputs for each state and input, **σ(*q*,*i*)**
- δ(q,i): Given a state q∈Q and input i∈Σ, **δ(*q*,*i*) returns a new state q'∈Q**.

Formal Properties

Composition: Letting T_1 be an FST from I_1 to O_1 and letting T_2 be an FST from I_2 to O_2, the two FSTs can be composed such that the resulting FST maps directly from I_1 to O_2.

Inversion: Letting T be an FST that maps from I to O, its inversion (T^{-1}) will map from O to I.



Deterministic vs. Non-Deterministic FSTs

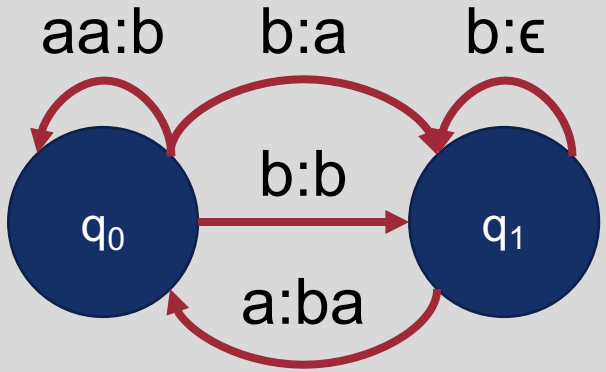
Just like FSAs, **FSTs can be non-deterministic** …one input can be translated to many possible outputs!

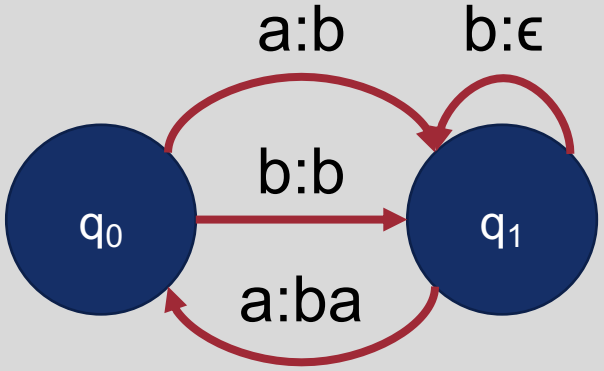
Unlike FSAs, **not all non-deterministic FSTs can be converted to deterministic FSTs**

FSTs with underlying deterministic FSAs (at any state, a given input maps to at most one transition out of the state) are called **sequential transducers**

Examples: Non-Deterministic and Sequential Transducers

Non-Deterministic





Sequential

Morphology

- **Morphemes**:
  - Small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Bits and pieces that adhere to stems and add additional information
    - -ed
    - -ing
    - -s
- Morphological parsing is a classic use case for FSTs

Morphological Parsing

- The task of recognizing the component morphemes of words (e.g., foxes → fox + es) and building structured representations of those components

Why is morphological parsing necessary?

Morphemes can be productive

- Example: -ing attaches to almost every verb, including brand new words
  - "Why are you Instagramming that?"

Some languages are very morphologically complex

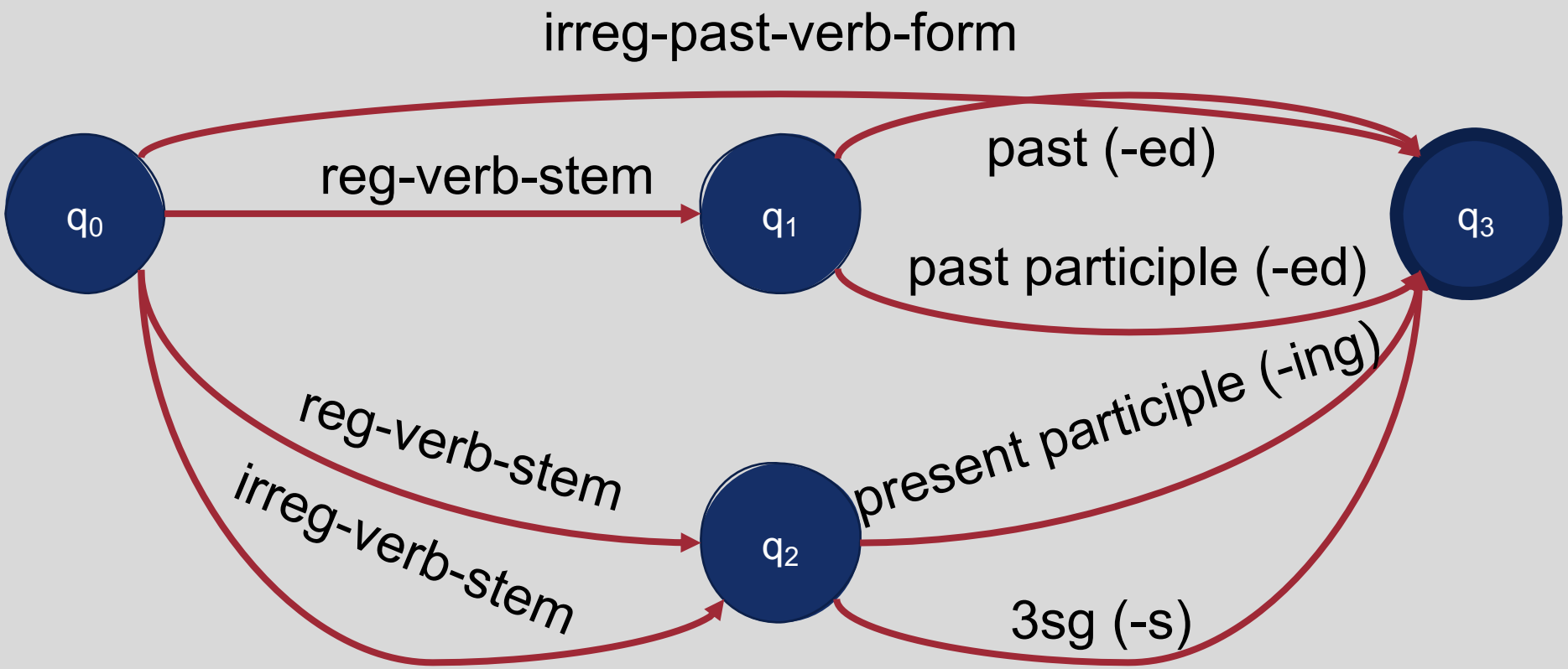
- Uygarlastiramadiklarimizdanmissinizcasina
  - Uygar 'civilized' + las 'become'
  - + tir 'cause' + ama 'not able'
  - + dik 'past' + lar 'plural'
  - + imiz 'p1pl' + dan 'abl'
  - + mis 'past' + siniz '2pl' + casina 'as if'

Surface Text	Morphological Parse
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	goose +N +SG
merging	merge +V +PresPart
caught	catch +V +Past

Finite State Morphological Parsing

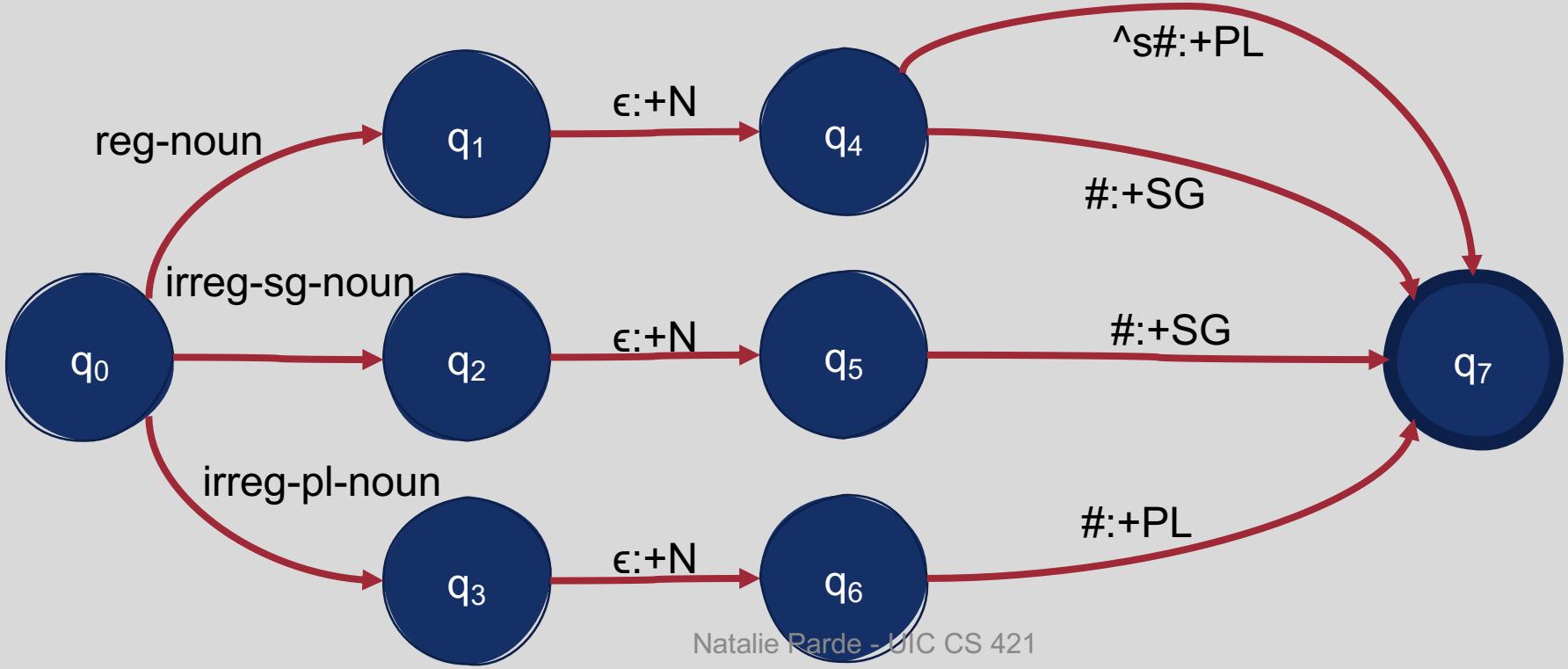
Goal: Take input surface realizations and produce morphological parses as output

Example Morphological Lexicon



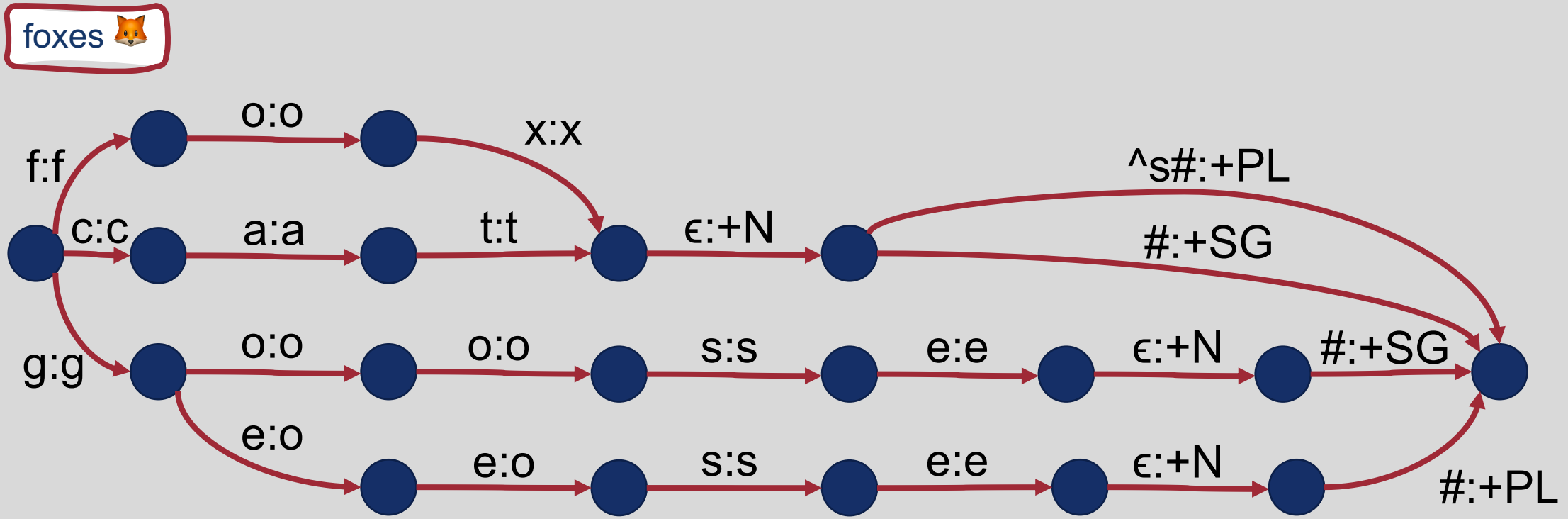
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



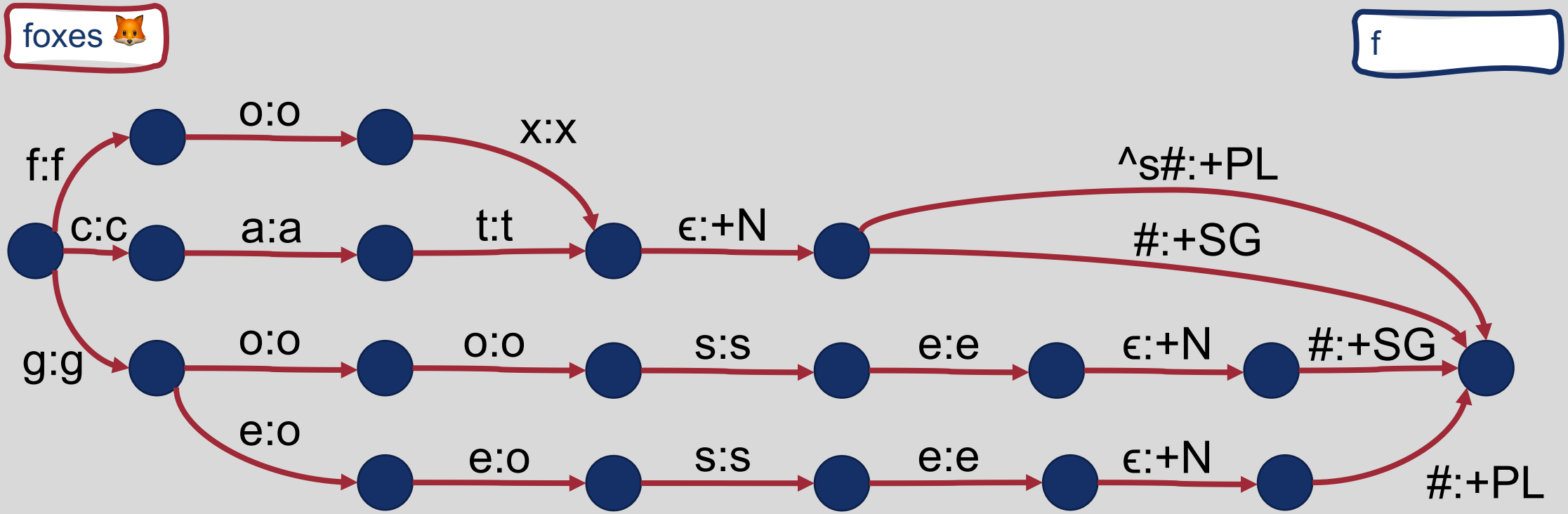
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



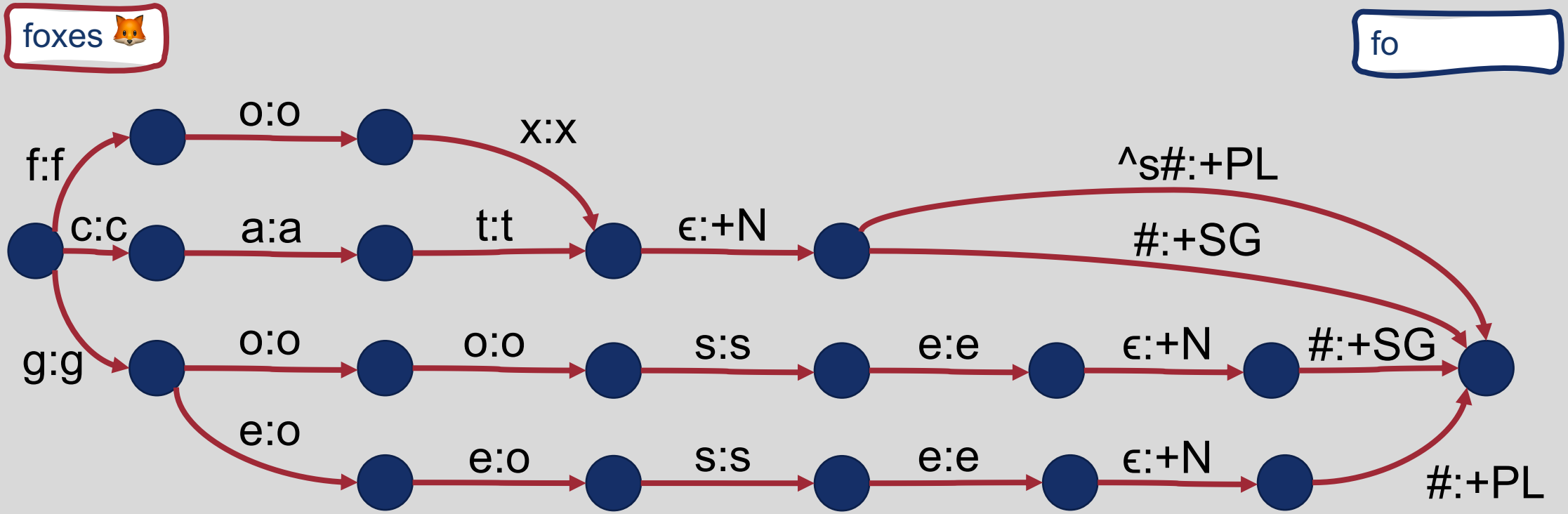
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



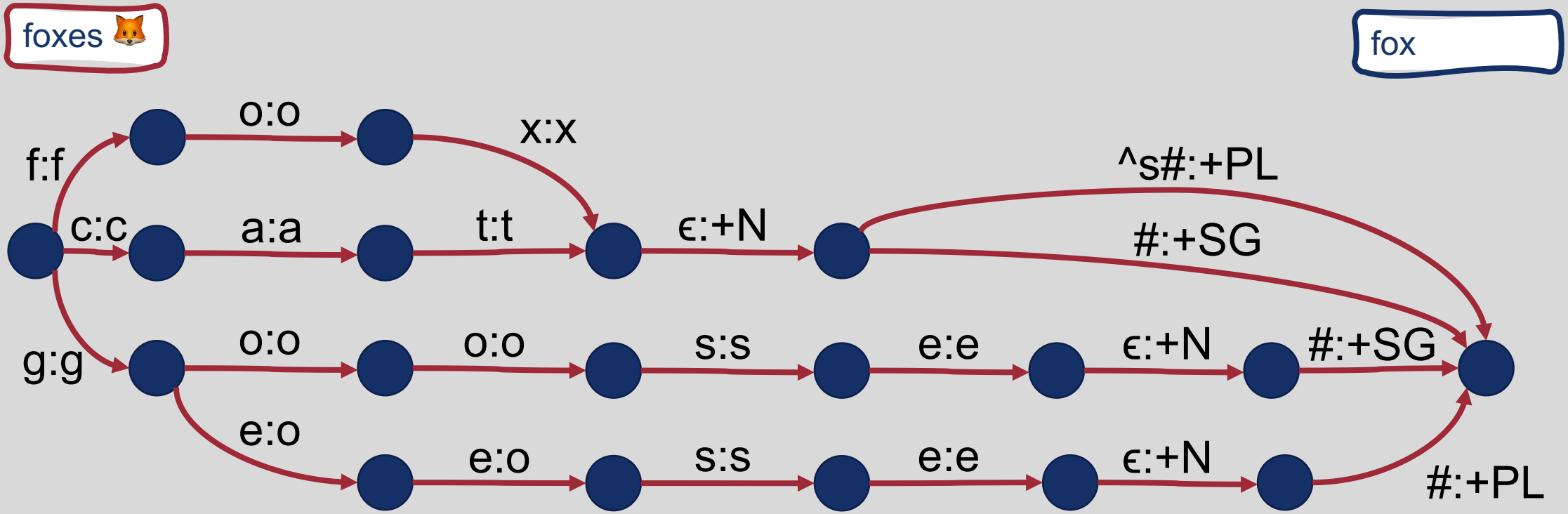
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



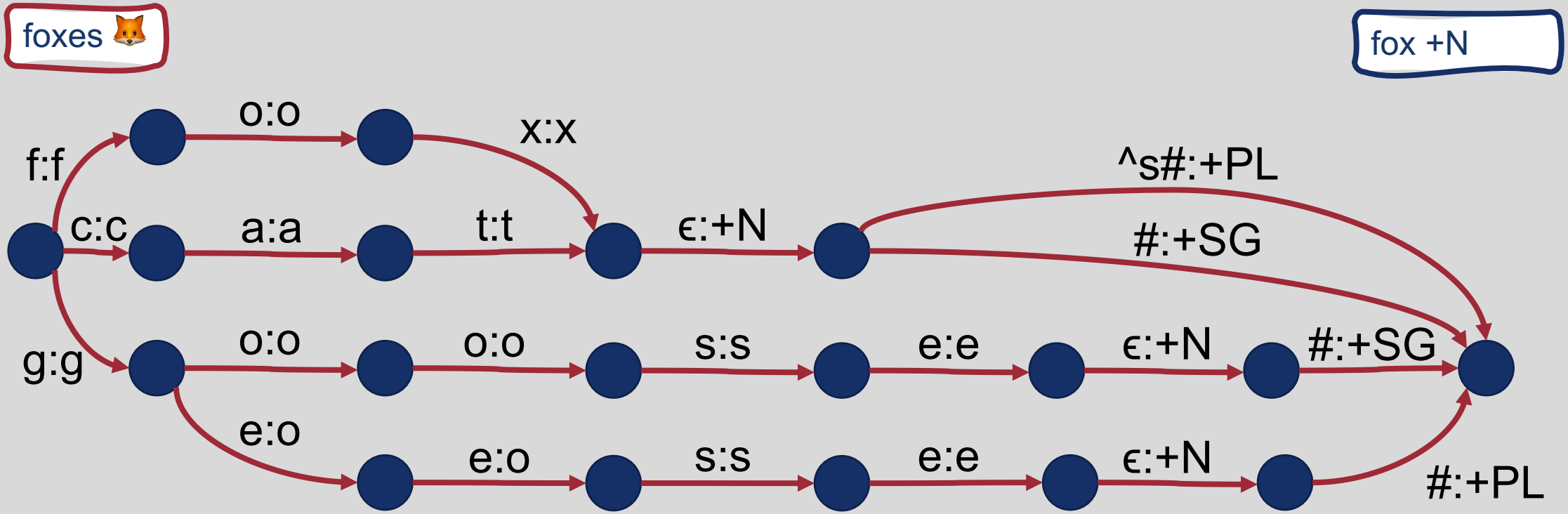
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



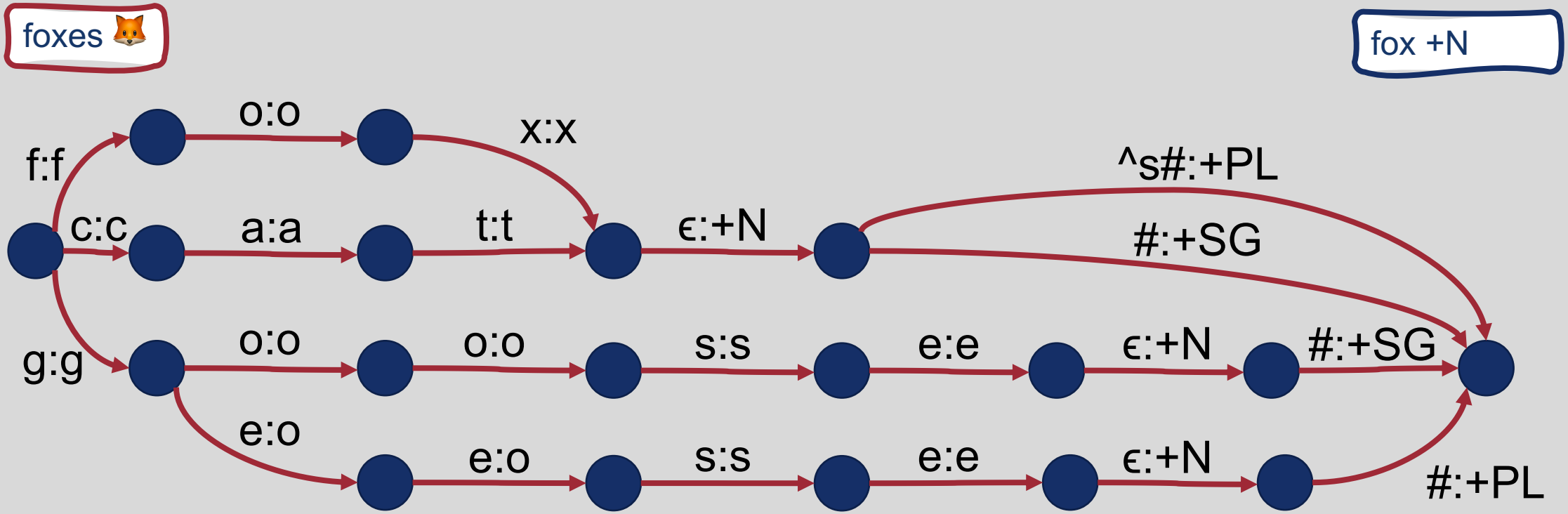
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



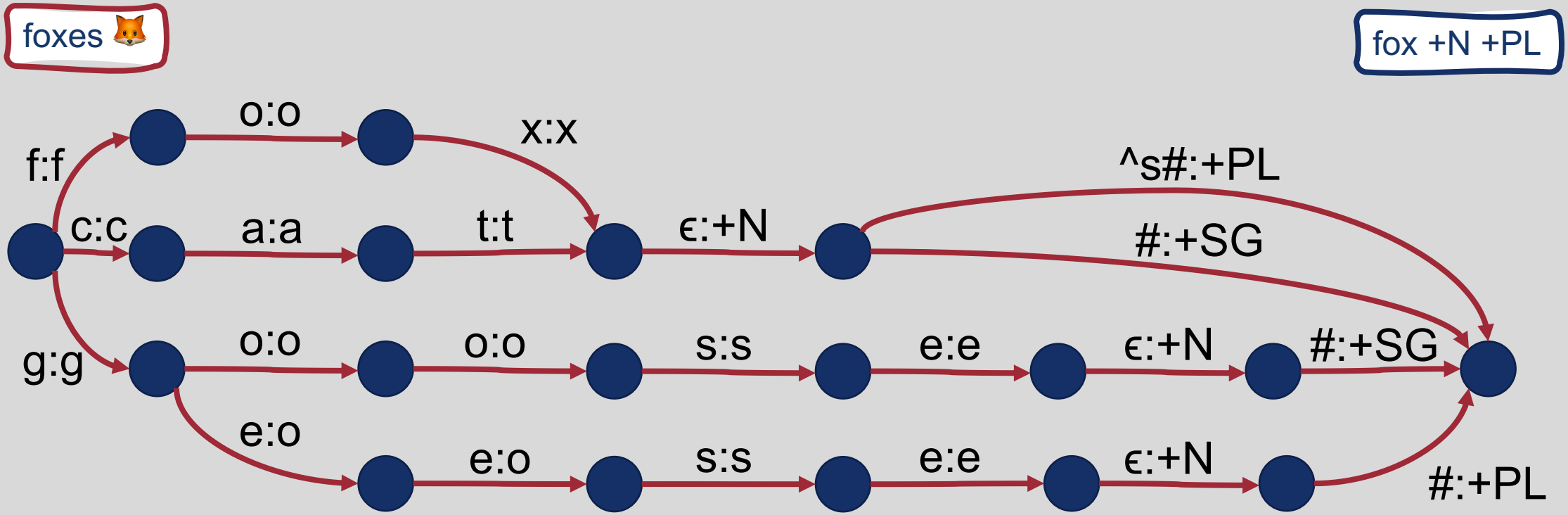
Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



Finite State Morphological Parsing

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat		



Summary: Finite State Transducers

- FSTs are FSAs that describe mappings between two sets
- Although all non-deterministic FSAs can be converted to deterministic versions, all non-deterministic FSTs cannot
- FSTs with underlying deterministic FSAs are called sequential transducers
- FSTs are particularly useful for morphological parsing

Moving back to more practical details....

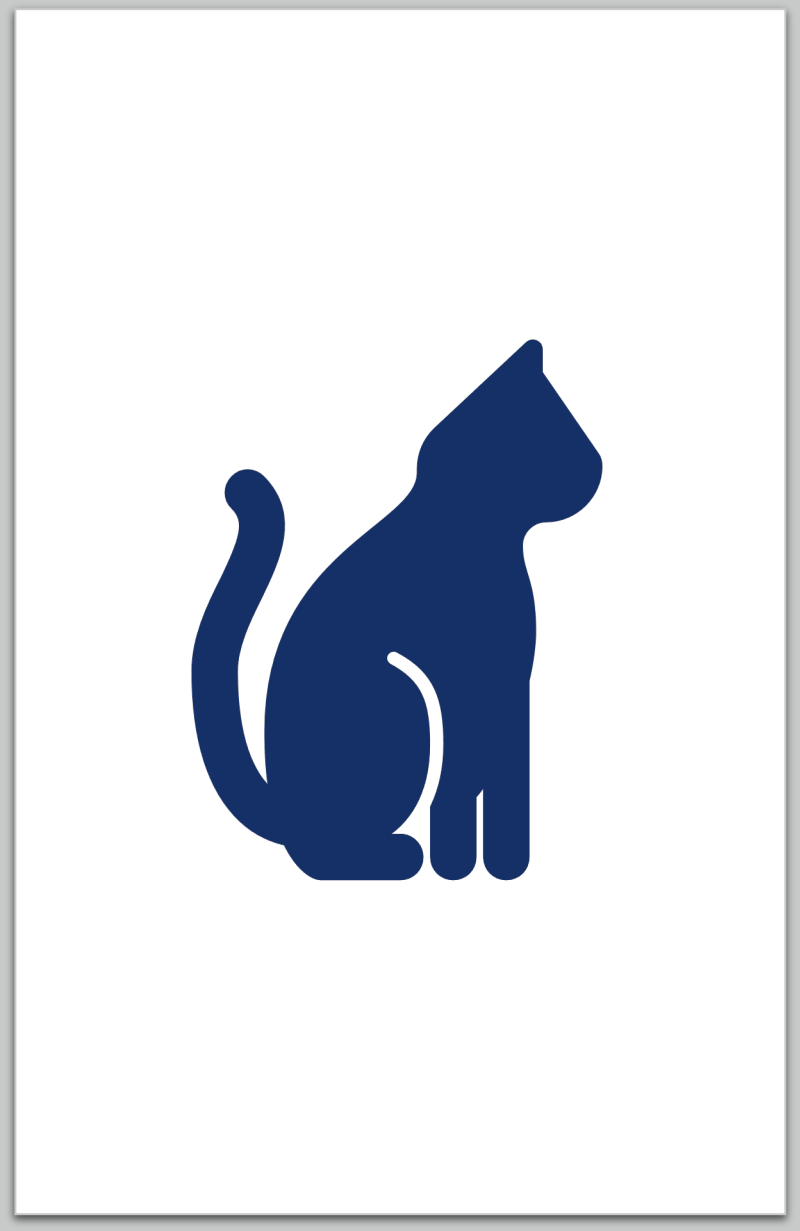
- Text tokenization is an important first step for most NLP tasks
- It is often implemented using regular expressions

Text Tokenization: Step #1 in Most NLP Pipelines

- A typical NLP pipeline begins by:
  - Segmenting sentences in running text
  - Separating words in running text
  - Normalizing word formats (e.g., favourite = favorite)

How many words?

- I do uh main- mainly business data processing
  - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats**!
  - **Lemma**: Words with the same stem, coarse-grained part of speech, and general word sense
    - cat and cats = same lemma
  - **Wordform**: The full inflected surface form of a word
    - cat and cats = different wordforms



How many words?

Alice looked all round the table, but there was nothing on it but tea.

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 14 tokens (or 16?)
  - 13 types (or 15?)

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Dataset	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20K
Shakespeare	884K	31K
Google N-grams	1 trillion	13 million

Issues in Tokenization

- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not ?
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → one token or two?
- a.m., Ph.D. → ??

Tokenization: Language Issues

Contractions

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

Tokens Not Delineated by Whitespace

- ***Lebensversicherungsgesellschaftsangestellter***
  - life insurance company employee
- 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Sharapova now lives in Florida in the southeastern United States.

Maximum Matching Word Segmentation Algorithm

Given a wordlist and a string:

1) Start a pointer at the beginning of the string
2) Find the longest word in dictionary that matches the string starting at pointer
3) Move the pointer over the word in string
4) Go to 2

莎拉波娃现在居住在美国东南部的佛罗里达。

莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

Doesn't generally work well with English....

Thecatinthehat → the cat in the hat

Thetabledownthere → ? theta bled own there

the table down there

- Nice Python tokenizers:
  - NLTK: http://www.nltk.org/api/nltk.tokenize.html
  - spaCy: https://spacy.io/api/tokenizer
  - StanfordNLP: https://github.com/stanfordnlp/stanza/

Text Normalization

- **Normalization:** Manipulating text such that all **forms** of the same word match (e.g., U.S.A. = USA, flavour = flavor, etc.)
- To normalize text, you must define **equivalence classes**
  - Example: Periods in a term → not important
- Words with the same characters but different capitalization are often considered equivalent to one another (**case folding**)
  - Example: Hello = hello
  - Not a perfect strategy!
    - US != us
- Useful equivalence classes vary depending on task
  - Capitalization can be very important in sentiment analysis

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be differ color*
- Tricky because you need to find the correct dictionary headword form

Stemming

- Automatically reduces words to their stems using simple rules
  - language dependent
  - Example: {automate(s), automatic, automation} → automat
- Pros: Very quick, simple to implement
- Cons: Groups together some words that don't really mean the same thing, and doesn't group together some words that do mean the same thing
  - {meanness, meaning} → mean
  - {goose} → goos, {geese} → gees

Porter Stemming

- Step 1a
  - sses → ss caresses → caress
  - ies → i ponies → poni
  - ss → ss caress → caress
  - s → ø cats → cat
- Step 1b
  - (*v*)ing → ø walking → walk
  - sing → sing
  - (*v*)ed → ø plastered → plaster
  - …
- Step 2 (for long stems)
  - ational→ ate relational→ relate
  - izer→ ize digitizer → digitize
  - ator→ ate operator → operate
  - …
- Step 3 (for longer stems)
  - al → ø revival → reviv
  - able → ø adjustable → adjust
  - ate → ø activate → activ
  - …

Much like tokenization, stemming methods are difficult to transfer across languages!

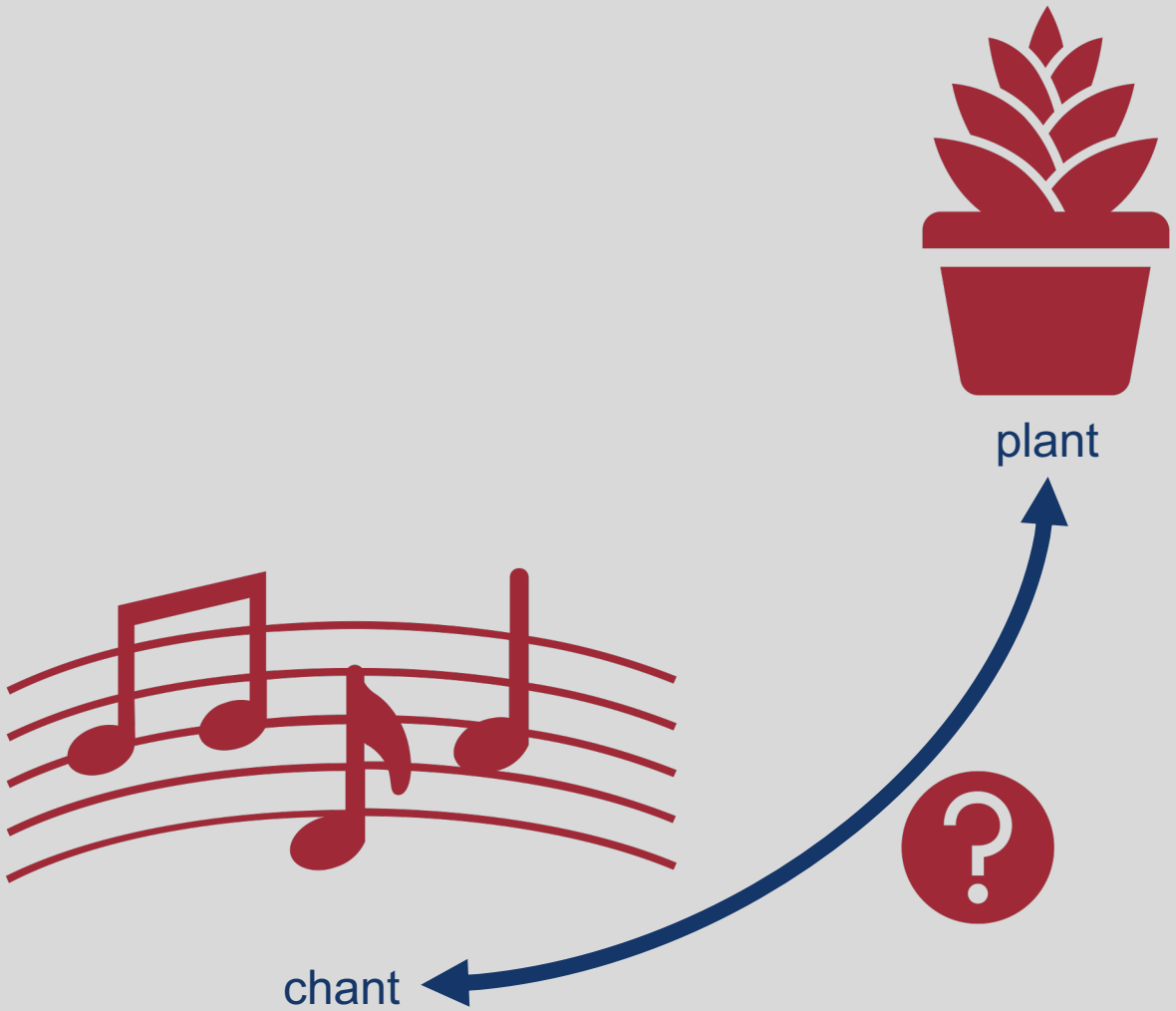
Sentence Segmentation

- !, ? are relatively unambiguous
- . is more ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Simple sentence segmentation:
  - Build a binary **classifier** that checks for "."
    - Classifier: A model that predicts labels for unseen test input
  - At each token, decides EndOfSentence/NotEndOfSentence

We know how to preprocess strings now …but how can we find the distance between them?

Popular string (or other sequence) comparison technique:

- Minimum edit distance



Edit Distance

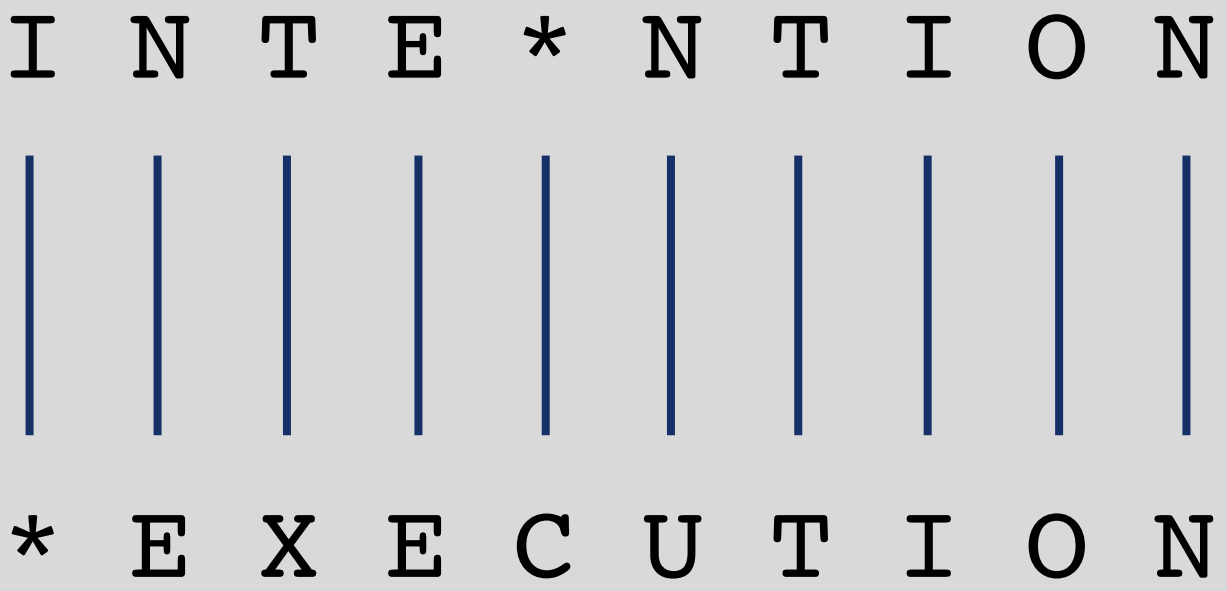
Simple way to answer the question: How similar are two strings?

Minimum Edit Distance

- Minimum number of editing operations needed to transform one string into another
- Possible editing operations:
  - Insertion
  - Deletion
  - Substitution

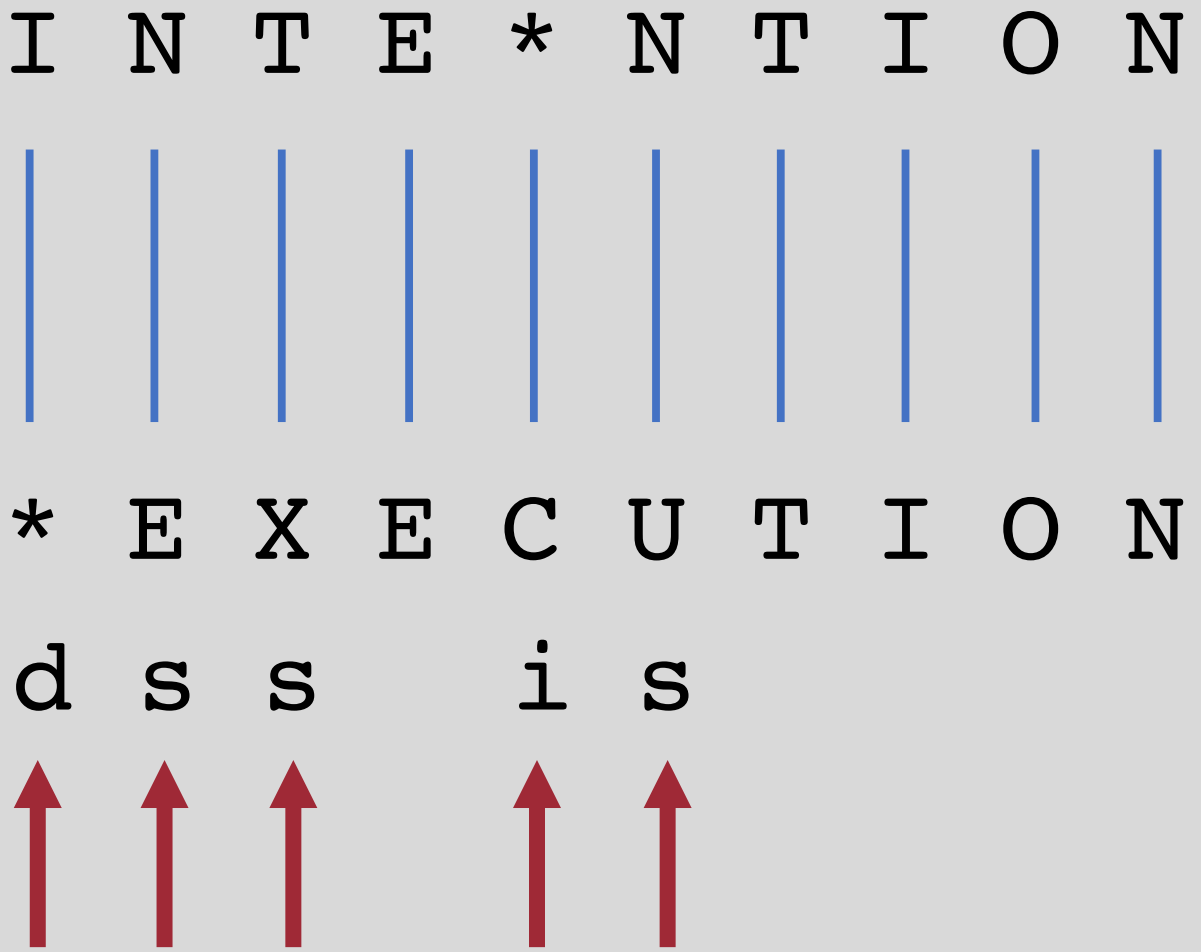
Minimum Edit Distance

- Two strings and their **alignment**:



Minimum Edit Distance

- If each operation has a cost of 1 (Levenshtein distance)
  - Distance between these is 5
- If substitutions cost 2 (alternative also proposed by Levenshtein)
  - Distance between them is 8



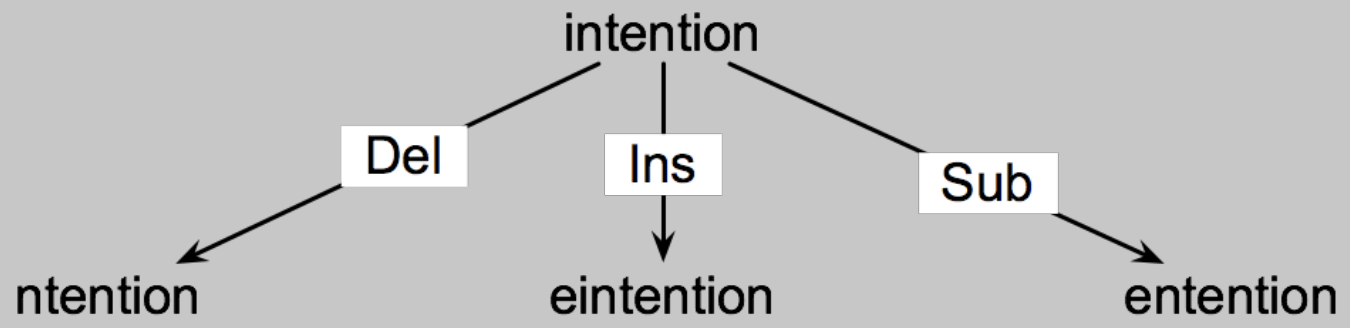
Other Uses of Edit Distance in NLP

- Evaluating machine translation and speech recognition using **word error rate**

```
Spokesman confirms    senior government adviser was shot
Spokesman said    the senior           adviser was shot dead
            S       I              D                    I
```

- Named entity extraction and entity coreference
  - **Meta Platforms, Inc.** announced today
  - **Meta** profits

How to find the minimum edit distance?



- Search for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize (the number of edits)

However, the search space of all edit sequences is huge!

- We can't afford to navigate naïvely
- Lots of distinct paths wind up at the same state
  - We don't have to keep track of all of them (just the shortest paths)

Formal Definition: Minimum Edit Distance

- For two strings
  - X of length *n*
  - Y of length *m*
- We define D(*i*,*j*) as the edit distance between X[1..*i*] and Y[1..*j*]
  - X[1..*i*] = the first *i* characters of X
- The edit distance between X and Y is thus D(*n*,*m*)

Intuition: Dynamic Programming

- Minimum edit distance can be solved using **dynamic programming**
  - Stores intermediate outputs in a table
  - Intuition: If some string B is in the optimal path from string A to string C, then that path must also include the optimal path from A to B
- D(n,m) is computed tabularly, combining solutions to subproblems
- Bottom-up
  - We compute D(i,j) for small *i,j*
  - And compute larger D(i,j) based on previously computed smaller values
    - i.e., compute D(*i,j*) for all *i* (0 < *i* < n) and *j* (0 < j < m)

Formal Definition: Minimum Edit Distance

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Algorithm:

  For each $i = 1 \ldots n$

  For each $j = 1 \ldots m$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; \text{ if } X(i) \neq Y(j) \\ 0; \text{ if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance

The Edit Distance Table

N										
O										
I										
T										
N										
E										
T										
N										
I										
#										
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2								
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7			
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7	6		
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

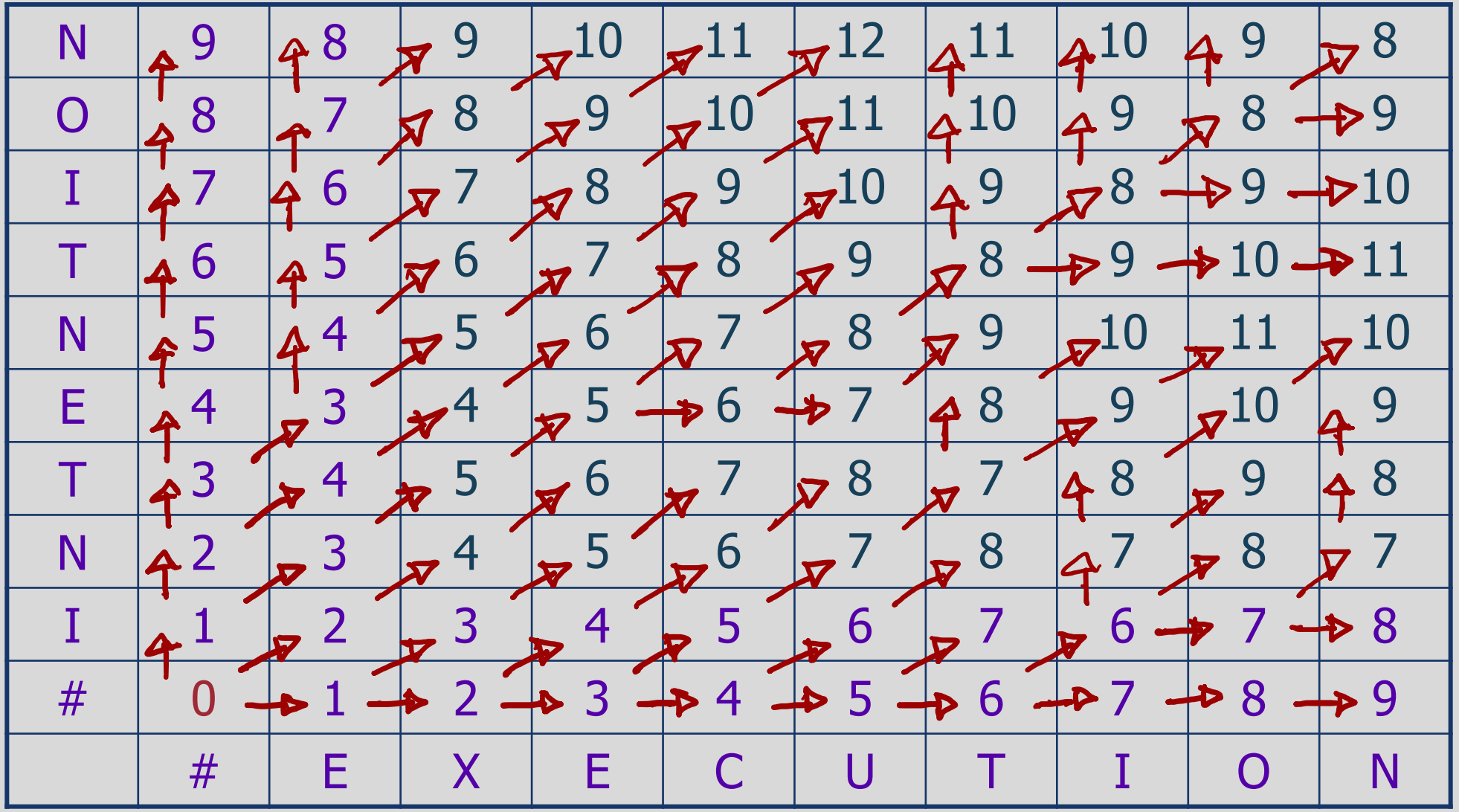
The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

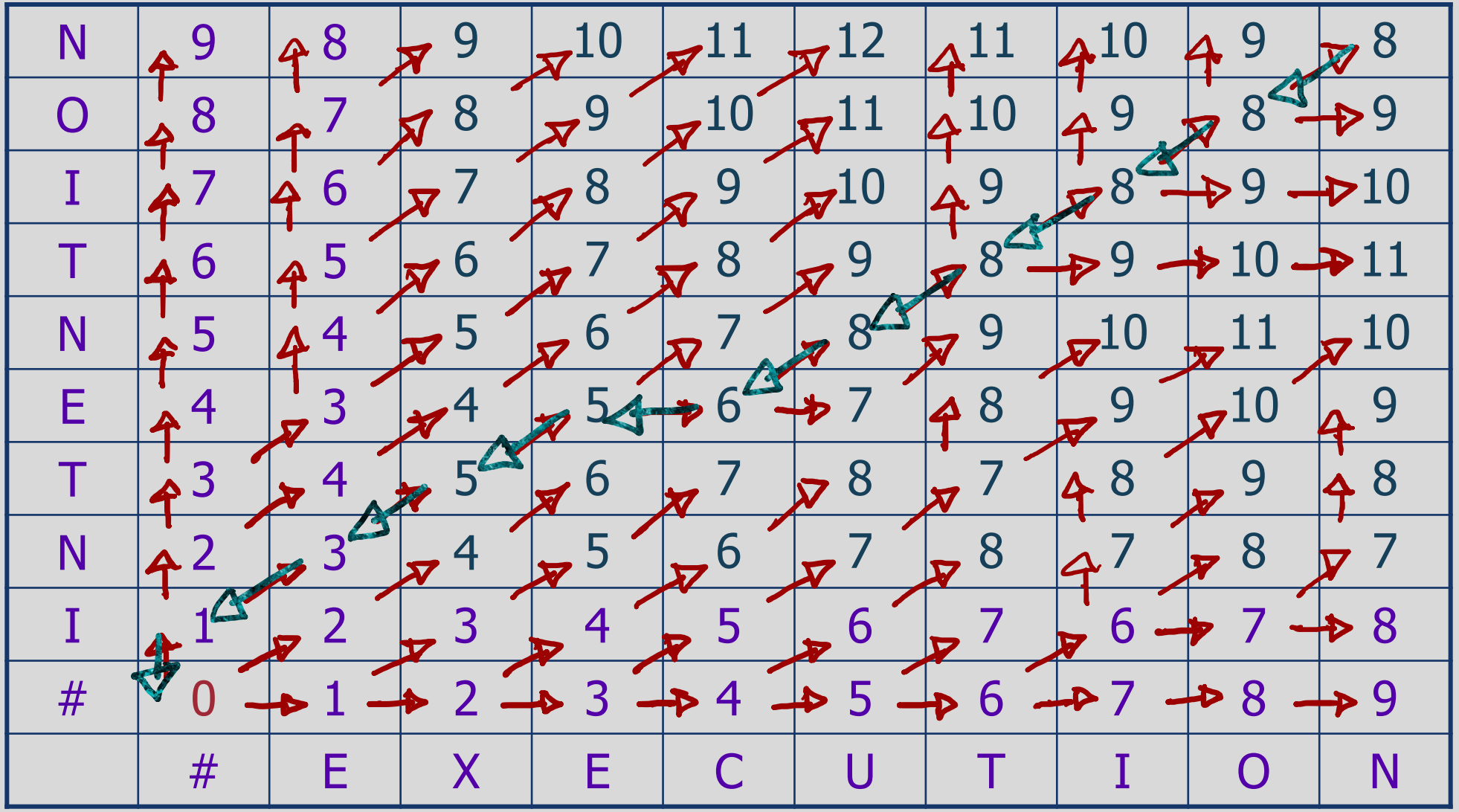
Backtrace for Computing Alignments

- We know the minimum edit distance now …but what is the alignment between the two strings?
- We can figure this out by maintaining a **backtrace**
  - For each new cell, remember where we came from!
    - D(i-1,j) ?
    - D(i,j-1) ?
    - D(i-1,j-1) ?
- Once we reach the end of the table (upper right corner), we can trace backward using these pointers to figure out the alignment

The Edit Distance Table



The Edit Distance Table



Formal Definition: Minimum Edit Distance with Backtrace

- Base conditions:

  D(i,0) = i D(0,j) = j

  Termination:

  D(N,M) is distance

- Algorithm:

```
For each  i = 1…n
      For each  j = 1…m
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} \\ \text{DOWN} \\ \text{DIAG} \end{cases}$$

Summary

- **Text Preprocessing:** Preparing text for downstream language processing tasks
  - Tokenization
  - Normalization
  - Lemmatization
  - Stemming
- Regular expressions are a powerful tool for text preprocessing!
- **Edit Distance:** Determining the similarity between two strings based on the number of insertions, deletions, and substitutions needed to transform one to another
- Minimum edit distance, computed using dynamic programming, allows you to find the smallest number of edits needed to do so.