Finite State Machines

Roger Levy

Massachusetts Institute of Technology
Department of Brain & Cognitive Sciences

Regular expressions – a minimal characterization

- Given a finite alphabet Σ, any character sequence from $\Sigma \cup \{*, |, (, \ )\}$ with matching parentheses is a valid regular expression.

Regular expressions – a minimal characterization

- Given a finite alphabet Σ, any character sequence from $\Sigma \cup \{*, |, (, \ )\}$ with matching parentheses is a valid regular expression.
- For example, if $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, the following are valid regular expressions:

`a`	`bab`	`b*a`
`a*`	`ab*`	`(ab)*`
`(a|b)`	`(a*|b)`	`(ab)|(b(a*))`

Regular expressions – a minimal characterization

- Given a finite alphabet Σ, any character sequence from $\Sigma \cup \{*, |, (, \ )\}$ with matching parentheses is a valid regular expression.
- For example, if $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, the following are valid regular expressions:

`a`	`bab`	`b*a`
`a*`	`ab*`	`(ab)*`
`(a|b)`	`(a*|b)`	`(ab)|(b(a*))`

- The empty string is also a valid regular expression

Regular expressions – a minimal characterization

- Given a finite alphabet Σ, any character sequence from $\Sigma \cup \{*, |, (, \ )\}$ with matching parentheses is a valid regular expression.
- For example, if $\Sigma = \{\texttt{a}, \texttt{b}\}$, the following are valid regular expressions:

`a`	`bab`	`b*a`
`a*`	`ab*`	`(ab)*`
`(a|b)`	`(a*|b)`	`(ab)|(b(a*))`

- The empty string is also a valid regular expression
- Semantics of what each part of a regular expression **matches**:

The empty string	A zero-character sequence
Any character from Σ	That character
Concatenation: XY	what X matches followed by what Y matches
$X*$	0 or more repetitions of X ($*$ is the "Kleene star")
$X \mid Y$	X or Y
()	determine operator precedence

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)
- Conveniences that **don't** change formal power:

`[]`	**Character classes**, e.g. `[a-f]` for `a|b|c|d|e|f`
`^` and `$`	**Anchors** requiring certain in-string position for partial matches
X`+`	"Kleene plus", equiv. $XX*$: 1+ repetitions of X
X`{`m`,`n`}`	**Counters**: between m and n repetitions of X
`.`	**Wildcard** (matches any symbol; a kind of character class)

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)
- Conveniences that **don't** change formal power:

`[]`	**Character classes**, e.g. `[a-f]` for `a|b|c|d|e|f`
`^` and `$`	**Anchors** requiring certain in-string position for partial matches
X`+`	"Kleene plus", equiv. $XX*$: 1+ repetitions of X
X`{`m`,`n`}`	**Counters**: between m and n repetitions of X
`.`	**Wildcard** (matches any symbol; a kind of character class)

- **Lookahead** and **lookbehind** guide the regex engine's matcher:

X`(?=`Y`)`	Require that Y follows X in order to match X
X`(?!`Y`)`	Require that Y not follow X in order to match X

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)
- Conveniences that **don't** change formal power:

`[]`	**Character classes**, e.g. `[a-f]` for `a|b|c|d|e|f`
`^` and `$`	**Anchors** requiring certain in-string position for partial matches
X`+`	"Kleene plus", equiv. $XX*$: 1+ repetitions of X
X`{`m`,`n`}`	**Counters**: between m and n repetitions of X
`.`	**Wildcard** (matches any symbol; a kind of character class)

- **Lookahead** and **lookbehind** guide the regex engine's matcher:

X`(?=`Y`)`	Require that Y follows X in order to match X
X`(?!`Y`)`	Require that Y not follow X in order to match X

- One common extension **does** change formal power: **backreferences**.

`(`X`)`Y`\1`	X then Y then a repetition of the string matching X, e.g.:
`([ab]*)b\1`	Matches `b`, `aba`, `bbb`, `abbba`, ...

  Matching regexes with arbitrary backreferences is NP-complete (**aho-1990:algorithms**)!

Additional machinery in many regex implementations

- **Partial** (cf. **total**) matching (e.g., Python's `re`'s `search()` vs. `fullmatch()`)
- Conveniences that **don't** change formal power:

`[]`	**Character classes**, e.g. `[a-f]` for `a|b|c|d|e|f`
`^` and `$`	**Anchors** requiring certain in-string position for partial matches
X`+`	"Kleene plus", equiv. $XX*$: 1+ repetitions of X
X`{`m`,`n`}`	**Counters**: between m and n repetitions of X
`.`	**Wildcard** (matches any symbol; a kind of character class)

- **Lookahead** and **lookbehind** guide the regex engine's matcher:

X`(?=`Y`)`	Require that Y follows X in order to match X
X`(?!`Y`)`	Require that Y not follow X in order to match X

- One common extension **does** change formal power: **backreferences**.

`(`X`)`Y`\1`	X then Y then a repetition of the string matching X, e.g.:
`([ab]*)b\1`	Matches `b`, `aba`, `bbb`, `abbba`, ...

  Matching regexes with arbitrary backreferences is NP-complete (**aho-1990:algorithms**)!
- In this discussion going forward, we cover regexes *without* backreferences.

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms
- Example:

  prick A word of English

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms
- Example:

  prick A word of English

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms
- Example:

prick	A word of English
plick	Not a word of English, but *could in principle be one*

To finite-state automata through phonotactics

- **Phonotactics** are the language-specific rules of what sound sequences constitute licit vs. illicit wordforms
- Example:

prick	A word of English
plick	Not a word of English, but *could in principle be one*
pnick	Not a word of English, and “could not be”

Language specificity of phonotactic rules

- *zvoon* (in the International Phonetic Alphabet: /zvun/) would be hard-pressed to be a word of English

Language specificity of phonotactic rules

- *zvoon* (in the International Phonetic Alphabet: /zvun/) would be hard-pressed to be a word of English
- But it would be a very natural word in Russian!

Language specificity of phonotactic rules

- *zvoon* (in the International Phonetic Alphabet: /zvun/) would be hard-pressed to be a word of English
- But it would be a very natural word in Russian!
- The individual sounds are all in English:

z	as in	"zebra"
v	as in	"victory"
u	as in	"hoop"
n	as in	"can"

  but the arrangement into *this sequence* is not OK in English

Language specificity of phonotactic rules

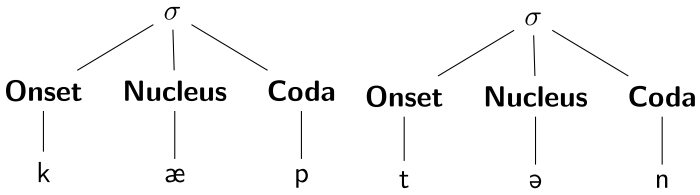
- *zvoon* (in the International Phonetic Alphabet: /zvun/) would be hard-pressed to be a word of English
- But it would be a very natural word in Russian!
- The individual sounds are all in English:

z	as in "zebra"
v	as in "victory"
u	as in "hoop"
n	as in "can"

  but the arrangement into *this sequence* is not OK in English
- Can you think of similar examples involving English and another language you know?

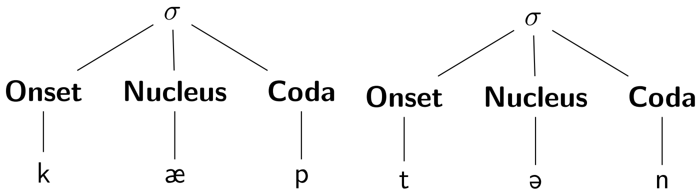
The grammar of English onsets

- Basic syllable structure of English, e.g. for *captain* /kæptən/:



The grammar of English onsets

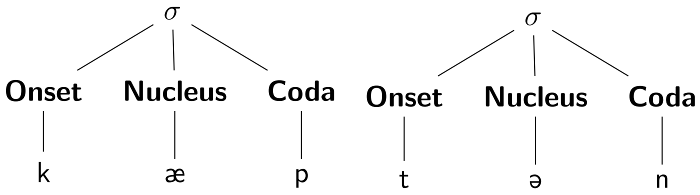
- Basic syllable structure of English, e.g. for *captain* /kæptən/:



- The badness of *zvoon* has to do with the **onset**

The grammar of English onsets

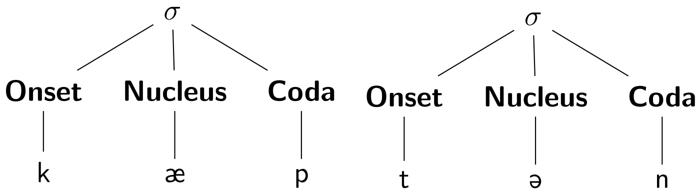
- Basic syllable structure of English, e.g. for *captain* /kæptən/:



- The badness of *zvoon* has to do with the **onset**
- Some examples of prohibitions on English onsets:

The grammar of English onsets

- Basic syllable structure of English, e.g. for *captain* /kæptən/:

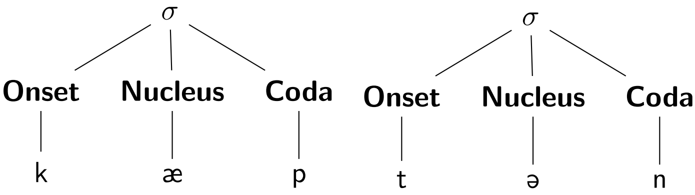


- The badness of *zvoon* has to do with the **onset**
- Some examples of prohibitions on English onsets:
  - **sonorants** (nasals like n, and liquids like l and r) can appear only at the *end* of an onset, e.g.:

OK:	net, ring, bring, plain
Not OK:	nzap, rwell, lroom

The grammar of English onsets

- Basic syllable structure of English, e.g. for *captain* /kæptən/:



- The badness of *zvoon* has to do with the **onset**
- Some examples of prohibitions on English onsets:
  - **sonorants** (nasals like n, and liquids like l and r) can appear only at the *end* of an onset, e.g.:

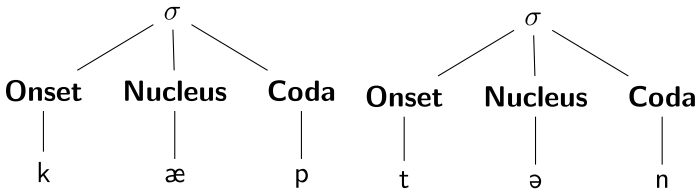
OK:	net, ring, bring, plain
Not OK:	nzap, rwell, lroom

  - The only fully acceptable sound that can precede a nasal or **obstruent** (stops like p, fricatives like z, and affricates like ch /tʃ/) is s, e.g.:

OK:	spill, snout
Not OK:	shpill, znout

The grammar of English onsets

- Basic syllable structure of English, e.g. for *captain* /kæptən/:



- The badness of *zvoon* has to do with the **onset**
- Some examples of prohibitions on English onsets:
  - **sonorants** (nasals like n, and liquids like l and r) can appear only at the *end* of an onset, e.g.:

OK:	net, ring, bring, plain
Not OK:	nzap, rwell, lroom

  - The only fully acceptable sound that can precede a nasal or **obstruent** (stops like p, fricatives like z, and affricates like ch /tʃ/) is s, e.g.:

OK:	spill, snout
Not OK:	shpill, znout

- Note that these prohibitions vary in generality!

Sound classes in phonology

- Similar to character classes in regexes, e.g., `[A-Fa-f]`, phonologists write sound classes using **phonological features**

Sound classes in phonology

- Similar to character classes in regexes, e.g., `[A-Fa-f]`, phonologists write sound classes using **phonological features**
- For any feature, a phoneme's value can be $+$, $-$, or unspecified

Feature	$+$ phonemes	$-$ phonemes
Voice	b, d, ð (first sound of *the*), g, v, z, j, ʒ (second consonant of *measure*)	tʃ ("ch"), f, k, p, s, ʃ ("sh"), t, θ ("th"), h
Labial	b, p, f, v, m, w	none
Sonorant	l, m, n, ŋ ("ng"), r, w, y	all others
Strident	tʃ, j, s, ʃ, z, ʒ	d, ð, t, θ, n, l, r
Continuant	ð, f, s, ʃ, θ, v, z, ʒ, h	b, tʃ, d, g, j, k, p, t

Sound classes in phonology

- Similar to character classes in regexes, e.g., `[A-Fa-f]`, phonologists write sound classes using **phonological features**
- For any feature, a phoneme's value can be $+$, $-$, or unspecified

Feature	$+$ phonemes	$-$ phonemes
Voice	b, d, ð (first sound of *the*), g, v, z, j, ʒ (second consonant of *measure*)	tʃ ("ch"), f, k, p, s, ʃ ("sh"), t, θ ("th"), h
Labial	b, p, f, v, m, w	none
Sonorant	l, m, n, ŋ ("ng"), r, w, y	all others
Strident	tʃ, j, s, ʃ, z, ʒ	d, ð, t, θ, n, l, r
Continuant	ð, f, s, ʃ, θ, v, z, ʒ, h	b, tʃ, d, g, j, k, p, t

- We can now write phonotactic prohibitions as little regular expressions:

Constraint	Regex	Explanation
$*\begin{bmatrix}+\text{Son}\end{bmatrix}\begin{bmatrix}\quad\end{bmatrix}$	`[mnŋlrwy].`	Sonorants may only be onset-final
$*\begin{bmatrix}\quad\end{bmatrix}\begin{bmatrix}+\text{Cont}\end{bmatrix}$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$*\begin{bmatrix}+\text{Cont}\\+\text{Voice}\end{bmatrix}\begin{bmatrix}\quad\end{bmatrix}$	`[ðvzʒ].`	Voiced fricatives can't precede anything

Toward a unified phonotactic grammar

Constraint	Regex	Explanation
$*\begin{bmatrix} +\text{Son} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[mnŋlrwy].`	Sonorants may only be onset-final
$*\begin{bmatrix} \quad \end{bmatrix}\begin{bmatrix} +\text{Cont} \end{bmatrix}$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$*\begin{bmatrix} +\text{Cont} \\ +\text{Voice} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[ðvzʒ].`	Voiced fricatives can't precede anything

- We can write each phonotactic **constraint** as a simple regex

Toward a unified phonotactic grammar

Constraint	Regex	Explanation
$*\begin{bmatrix} +\text{Son} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[mnŋlrwy].`	Sonorants may only be onset-final
$*\begin{bmatrix} \quad \end{bmatrix}\begin{bmatrix} +\text{Cont} \end{bmatrix}$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$*\begin{bmatrix} +\text{Cont} \\ +\text{Voice} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[ðvzʒ].`	Voiced fricatives can't precede anything

- We can write each phonotactic **constraint** as a simple regex
- How can we combine a set of phonotactic constraints into a unified **phonotactic grammar**?

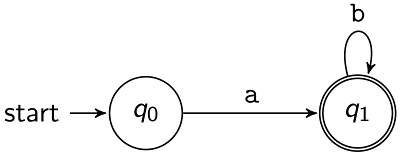
Toward a unified phonotactic grammar

Constraint	Regex	Explanation
$*\begin{bmatrix} +\text{Son} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[mnŋlrwy].`	Sonorants may only be onset-final
$*\begin{bmatrix} \quad \end{bmatrix}\begin{bmatrix} +\text{Cont} \end{bmatrix}$	`.[ðfsʃθvzʒh]`	Fricatives can't have anything before them
$*\begin{bmatrix} +\text{Cont} \\ +\text{Voice} \end{bmatrix}\begin{bmatrix} \quad \end{bmatrix}$	`[ðvzʒ].`	Voiced fricatives can't precede anything

- We can write each phonotactic **constraint** as a simple regex
- How can we combine a set of phonotactic constraints into a unified **phonotactic grammar**?
- A natural way to do this turns out to be through the formalism of **finite-state machines**

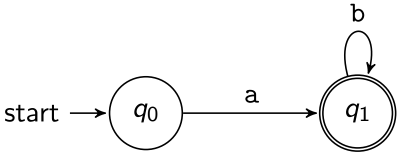
Finite-state automata: an example

- Example **finite-state automaton** (FSA):



Finite-state automata: an example

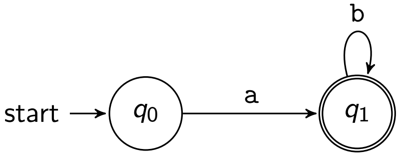
- Example **finite-state automaton** (FSA):



- Accepts all and only those strings that begin with `a` and then have nothing but `b`

Finite-state automata: an example

- Example **finite-state automaton** (FSA):



- Accepts all and only those strings that begin with `a` and then have nothing but `b`
- More precisely, it accepts all and only the strings accepted by the regular expression `ab*`

Finite-state automata, formally defined

A **finite-state automaton** consists of:

- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;

Finite-state automata, formally defined

A **finite-state automaton** consists of:

- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;
- A finite **input alphabet** Σ of symbols (the symbols that comprise strings, like in regexes);

Finite-state automata, formally defined

A **finite-state automaton** consists of:

- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;
- A finite **input alphabet** Σ of symbols (the symbols that comprise strings, like in regexes);
- A set of **final states** $F \subseteq Q$;

Finite-state automata, formally defined

A **finite-state automaton** consists of:

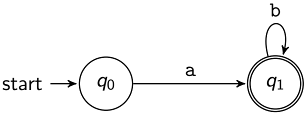
- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;
- A finite **input alphabet** Σ of symbols (the symbols that comprise strings, like in regexes);
- A set of **final states** $F \subseteq Q$;
- The transition relation Δ, comprised ofa finite set of TRANSITIONS each of the form $q \overset{i}{\leadsto} q'$, with $i \in \Sigma$ or $i = \epsilon$ (*epsilon* is the **empty string**). Informally, "if you are in state q and have input i available next, you can consume it and move to state q'".

Finite-state automata, formally defined

A **finite-state automaton** consists of:

- A finite set of N **states** $Q = \{q_0, q_1, \ldots, q_{N-1}\}$, with q_0 the **start state**;
- A finite **input alphabet** Σ of symbols (the symbols that comprise strings, like in regexes);
- A set of **final states** $F \subseteq Q$;
- The transition relation Δ, comprised ofa finite set of TRANSITIONS each of the form $q \overset{i}{\rightsquigarrow} q'$, with $i \in \Sigma$ or $i = \epsilon$ (*epsilon* is the **empty string**). Informally, "if you are in state q and have input i available next, you can consume it and move to state q'".

Equivalent specifications of the same FSA



$$Q = \{q_0, q_1\}$$
$$\Sigma = \{\mathtt{a}, \mathtt{b}\}$$
$$F = \{q_1\}$$
$$\Delta = \{q_0 \overset{\mathtt{a}}{\rightsquigarrow} q_1,\ q_1 \overset{\mathtt{b}}{\rightsquigarrow} q_1\}$$

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\leadsto}} \underset{2}{\overset{i_2}{\leadsto}} \dots \underset{N-1}{\overset{i_{N-1}}{\leadsto}} \underset{N}{\overset{i_N}{\leadsto}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \ldots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

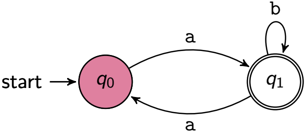
where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



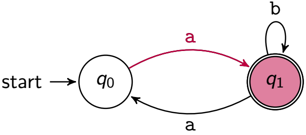
a b b
b
a b a

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \ldots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.



a|b b

b

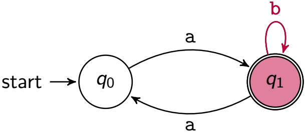
a b a

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



a b|b

b

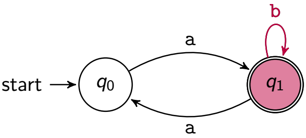
a b a

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

  where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



a b b|

b

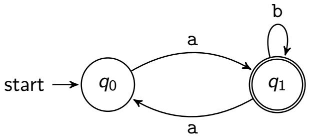
a b a

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\leadsto}} \underset{2}{\overset{i_2}{\leadsto}} \dots \underset{N-1}{\overset{i_{N-1}}{\leadsto}} \underset{N}{\overset{i_N}{\leadsto}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



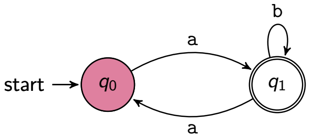
a b b	**Accepted**
b	
a b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \overset{i_1}{\underset{1}{\rightsquigarrow}} \overset{i_2}{\underset{2}{\rightsquigarrow}} \dots \overset{i_{N-1}}{\underset{N-1}{\rightsquigarrow}} \overset{i_N}{\underset{N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



a b b **Accepted**

b

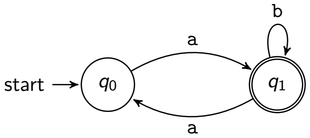
a b a

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \ldots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.



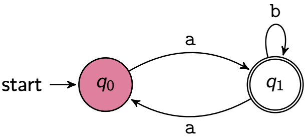
a b b	**Accepted**
b	**Rejected**
a b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



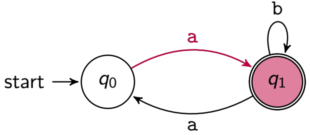
a b b	**Accepted**
b	**Rejected**
a b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \ldots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.



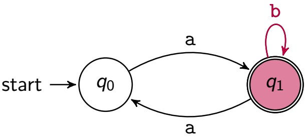
a b b	**Accepted**
b	**Rejected**
a|b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



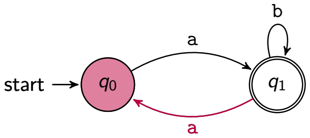
a b b	**Accepted**
b	**Rejected**
a b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



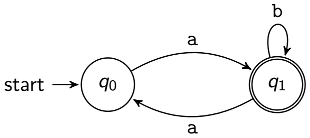
a b b	**Accepted**
b	**Rejected**
a b a	

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \ldots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \ldots, i_N$ are the appropriately sequenced inputs from w.



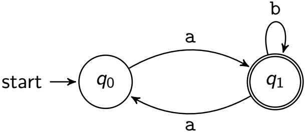
a b b	**Accepted**
b	**Rejected**
a b a	**Rejected**

Acceptance criterion in FSAs (slightly informal)

- An FSA **accepts** a string if you can **recursively** apply the transition relation to the current state (initializing at q_0) and the current position in the string (initializing at the beginning of the string) and get to a final state with the string completely consumed.
- If the sequence of transitions is of length N we may depict a **path through the automaton** that accepts w as

$$q_0 \underset{1}{\overset{i_1}{\rightsquigarrow}} \underset{2}{\overset{i_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{i_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{i_N}{\rightsquigarrow}} q^*$$

where $q^* \in F$ and $i_1, \dots, i_N$ are the appropriately sequenced inputs from w.



a b b	**Accepted**
b	**Rejected**
a b a	**Rejected**

- **For every regex there is an FSA that accepts all and only the strings the regex matches, and vice versa!**

Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):

Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ

Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.

Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.
- As a result, in a deterministic FSA, there is never more than one transition possible given a state and the current position in the string.

Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.
- As a result, in a deterministic FSA, there is never more than one transition possible given a state and the current position in the string.
- In a **non-deterministic** finite-state automaton (NFSA), neither of those constraints hold.

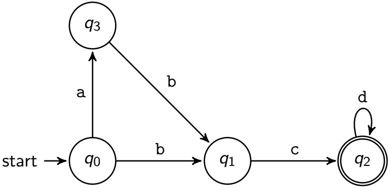
Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.
- As a result, in a deterministic FSA, there is never more than one transition possible given a state and the current position in the string.
- In a **non-deterministic** finite-state automaton (NFSA), neither of those constraints hold.

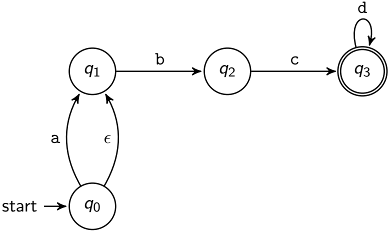
Deterministic versus non-deterministic FSAs

- In a **deterministic** finite-state automaton (DFSA):
  - Each transition's input symbol i must be a symbol in Σ, and cannot be ϵ
  - The transition relation is a **function**.
- As a result, in a deterministic FSA, there is never more than one transition possible given a state and the current position in the string.
- In a **non-deterministic** finite-state automaton (NFSA), neither of those constraints hold.

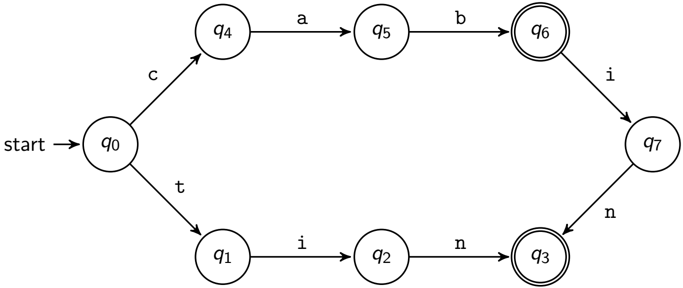
DFSA

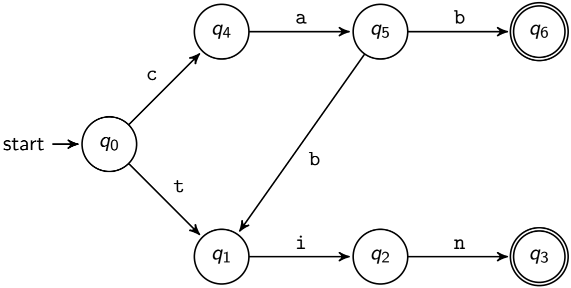


Equivalent NFSA

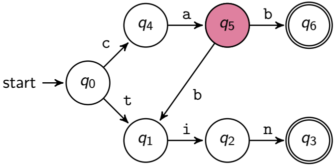


Another equivalent DFSA/NFSA pair





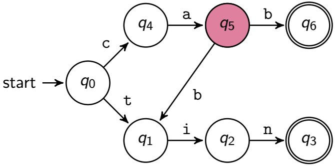
Checking string acceptance/rejection in an NFSA



c a|b...

- Checking for acceptance is harder for NFSAs than for DFSAs, due to choicepoints!

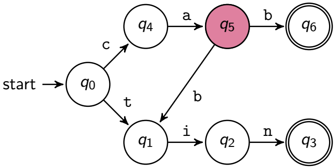
Checking string acceptance/rejection in an NFSA



`c a|b...`

- Checking for acceptance is harder for NFSAs than for DFSAs, due to choicepoints!
- Above, there are two possible outward transitions in q_5 for input symbol `b`.

Checking string acceptance/rejection in an NFSA



c a|b...

- Checking for acceptance is harder for NFSAs than for DFSAs, due to choicepoints!
- Above, there are two possible outward transitions in q_5 for input symbol b.
- Algorithmic options:
  - **Backup:** whenever we encounter a choicepoint, generate a list of transition options and mark our position. Try one. If we fail, go back to the last choicepoint and try the next option on the list. If we run out of options, then the string is rejected.
  - **Lookahead:** look forward in the string to guide choice.
  - **Parallelism:** Instead of maintaining and updating a single state, build a *set* of possible states for each string position.

FSA determinization

- Every NFSA can be **determinized** to create a DFSA that accepts and rejects the same strings

FSA determinization

- Every NFSA can be **determinized** to create a DFSA that accepts and rejects the same strings
- NFSAs and DFSAs are expressively equivalent

Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\rightsquigarrow} q' \in \Delta$ for some $q' \in Q$.

Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\rightsquigarrow} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.

Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\leadsto} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.
- If an FSA is not total, then it is PARTIAL.

Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\rightsquigarrow} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.
- If an FSA is not total, then it is PARTIAL.
- For every partial FSA, there is a total FSA that accepts and rejects exactly the same string set.

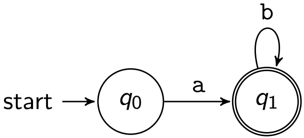
Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\leadsto} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.
- If an FSA is not total, then it is PARTIAL.
- For every partial FSA, there is a total FSA that accepts and rejects exactly the same string set.

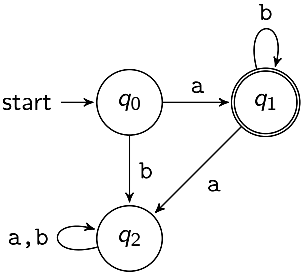
Partial vs total FSAs

- In a TOTAL FSA, state $q \in Q$ and every symbol $i \in \Sigma$, there is a transition $q \overset{i}{\rightsquigarrow} q' \in \Delta$ for some $q' \in Q$.
- Graphically: in every state, for every symbol in Σ there is at least one outgoing arc labeled with that symbol.
- If an FSA is not total, then it is PARTIAL.
- For every partial FSA, there is a total FSA that accepts and rejects exactly the same string set.

Partial FSA



Equivalent total FSA



Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars

Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars
- A grammar fragment is not a complete description of a language and its structure

Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars
- A grammar fragment is not a complete description of a language and its structure
- Rather, a fragment targets *part* of a language and should capture insights about the structure of that part

Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars
- A grammar fragment is not a complete description of a language and its structure
- Rather, a fragment targets *part* of a language and should capture insights about the structure of that part
- Grammar fragments can target any of a number of levels of linguistic structure: lexicon, phonology, morphology, syntax, semantics

Writing grammar fragments

- With FSAs in hand, we will start to explore writing **fragments** of natural language grammars
- A grammar fragment is not a complete description of a language and its structure
- Rather, a fragment targets *part* of a language and should capture insights about the structure of that part
- Grammar fragments can target any of a number of levels of linguistic structure: lexicon, phonology, morphology, syntax, semantics
- Broad goal: as we accumulate grammar fragments for a language, we should obtain an increasingly close approximation of the *true* characterization of the language and its structure

Fragment example 1: Finnish word-level phonotactics

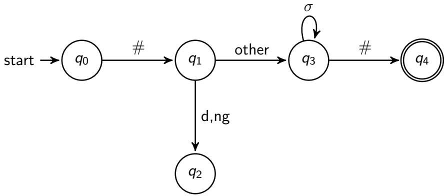
- In Finnish, the possible syllable structures are V, CV, VC, CVC, CCVC (where C=consonant, V=vowel).

Fragment example 1: Finnish word-level phonotactics

- In Finnish, the possible syllable structures are V, CV, VC, CVC, CCVC (where C=consonant, V=vowel).
- Constraint 1: Word-initially, any consonant can appear except for d and ng.

Fragment example 1: Finnish word-level phonotactics

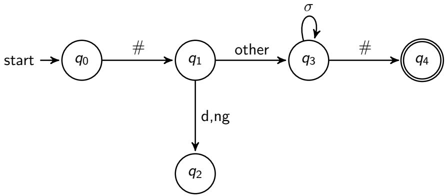
- In Finnish, the possible syllable structures are V, CV, VC, CVC, CCVC (where C=consonant, V=vowel).
- Constraint 1: Word-initially, any consonant can appear except for d and ng.



#	=	word boundary
"other"	=	symbols in Σ not appearing in another outgoing edge from the state

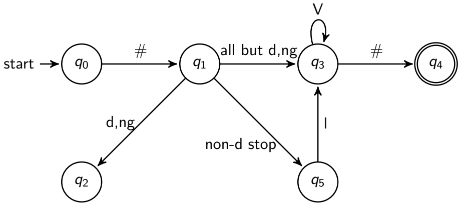
Fragment example 1: Finnish word-level phonotactics

- In Finnish, the possible syllable structures are V, CV, VC, CVC, CCVC (where C=consonant, V=vowel).
- Constraint 1: Word-initially, any consonant can appear except for d and ng.



# =	word boundary
"other" =	symbols in Σ not appearing in another outgoing edge from the state

- Add constraint 2: Word-initially, CC sequences must be stop+liquid(=l).



Fragment example 2: English adjective ordering

- Consider the following English adjective classes:

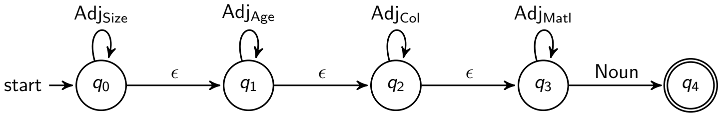
Class	**Examples**
Size	*big*, *short*, *wide*, *heavy*, *voluminous*
Age	*old*, *new*, *recent*
Color	*blue*, *black*, *white*, *colorless*
Material	*wooden*, *organic*, *metal*, *stone*

Fragment example 2: English adjective ordering

- Consider the following English adjective classes:

Class	**Examples**
Size	*big*, *short*, *wide*, *heavy*, *voluminous*
Age	*old*, *new*, *recent*
Color	*blue*, *black*, *white*, *colorless*
Material	*wooden*, *organic*, *metal*, *stone*

- The following FSA captures their relative ordering preferences:

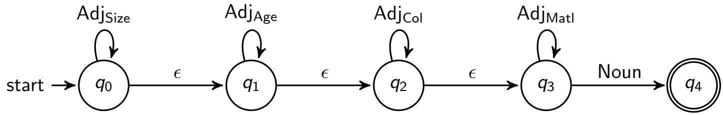


Fragment example 2: English adjective ordering

- Consider the following English adjective classes:

Class	**Examples**
Size	*big*, *short*, *wide*, *heavy*, *voluminous*
Age	*old*, *new*, *recent*
Color	*blue*, *black*, *white*, *colorless*
Material	*wooden*, *organic*, *metal*, *stone*

- The following FSA captures their relative ordering preferences:



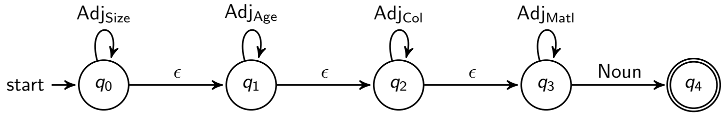
footnotesize	**Acceptable**	**Unacceptable**
	table	*old* (missing a noun!)
	old table	?*table old*
	big blue building	?*blue big building*
	voluminous organic produce	?*organic voluminous produce*

Fragment example 2: English adjective ordering

- Consider the following English adjective classes:

Class	**Examples**
Size	*big*, *short*, *wide*, *heavy*, *voluminous*
Age	*old*, *new*, *recent*
Color	*blue*, *black*, *white*, *colorless*
Material	*wooden*, *organic*, *metal*, *stone*

- The following FSA captures their relative ordering preferences:



footnotesize

Acceptable	**Unacceptable**
table	*old* (missing a noun!)
old table	?*table old*
big blue building	?*blue big building*
voluminous organic produce	?*organic voluminous produce*

- **Note:** the fuller picture is more complicated! For example, contrastive stress can help bring an adjective leftward (e.g., "the **WOODEN** old door, not the **STONE** one"). But this simple description captures some major trends.

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings
- We can write **fragments** of natural language grammars using these formalisms for at least some kinds of phonotactics and bits of English syntax

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings
- We can write **fragments** of natural language grammars using these formalisms for at least some kinds of phonotactics and bits of English syntax
- **Looking ahead:**

Summary thus far

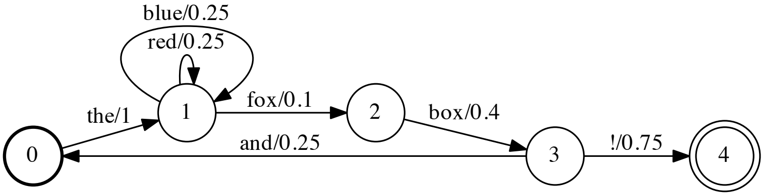
- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings
- We can write **fragments** of natural language grammars using these formalisms for at least some kinds of phonotactics and bits of English syntax
- **Looking ahead:**
  - Mechanisms to combine finite-state grammar fragments into a single unified fragment

Summary thus far

- Regular expressions are an expressive, but constrained, formalism for defining sets of strings
- Finite-state automata are an *expressively equivalent* formalism for defining sets of strings
- We can write **fragments** of natural language grammars using these formalisms for at least some kinds of phonotactics and bits of English syntax
- **Looking ahead:**
  - Mechanisms to combine finite-state grammar fragments into a single unified fragment
  - What parts of natural language structure can and cannot be captured by these formalisms?

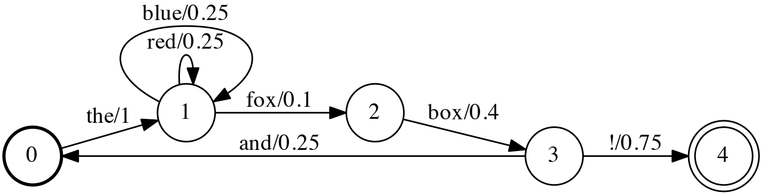
Weighted Finite-State Automata (WFSAs)

- An example of a WFSA:



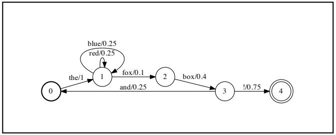
Weighted Finite-State Automata (WFSAs)

- An example of a WFSA:



- A useful exercise: write regular expressions that are equivalent to each of the automata on this page, and on the previous page

Weighted Finite-State Automata (WFSAs)



A WEIGHTED FINITE-STATE AUTOMATON (WFSA) consists of a tuple (Q, V, S, R) such that:

- Q is a finite set of STATES $q_0 q_1 \ldots q_N$, with q_0 the designated START STATE;
- Σ is a finite set of terminal symbols;
- $F \subseteq Q$ is the set of FINAL STATES;
- Δ is a finite set of TRANSITIONS each of the form $q \stackrel{i}{\rightsquigarrow} q'$, meaning that “if you are in state q and see symbol i you can consume it and move to state q'”;
- λ is a function mapping transitions to real numbers (weights);
- ρ is a function mapping final states to real numbers (weights).

Weighted Finite-State Automata (WFSAs)

- Q is a finite set of STATES $q_0 q_1 \dots q_N$, with q_0 the designated START STATE;
- Σ is a finite set of terminal symbols;
- $F \subseteq Q$ is the set of FINAL STATES;
- Δ is a finite set of TRANSITIONS each of the form $q \overset{i}{\rightsquigarrow} q'$, meaning that "if you are in state q and see symbol i you can consume it and move to state q'";
- λ is a function mapping transitions to real numbers (weights);
- ρ is a function mapping final states to real numbers (weights).

- $w_{1 \dots N} \in \Sigma^N$ is ACCEPTED or RECOGNIZED by an automaton iff there is a PATH of transitions $\underset{1 \dots N}{\rightsquigarrow}$ to a final state $q^* \in F$ such that

$$q_0 \underset{1}{\overset{w_1}{\rightsquigarrow}} \underset{2}{\overset{w_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{w_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{w_N}{\rightsquigarrow}} q^*$$

- The WEIGHT of such a path $\underset{1 \dots N}{\rightsquigarrow}$ is the product of the weights of each of the transitions, together with the weight of the final state:

$$P(q_0 \underset{1}{\overset{w_1}{\rightsquigarrow}} \underset{2}{\overset{w_2}{\rightsquigarrow}} \dots \underset{N-1}{\overset{w_{N-1}}{\rightsquigarrow}} \underset{N}{\overset{w_N}{\rightsquigarrow}} q^*) = \rho(q^*) \prod_{i=1}^{N} \lambda(\underset{i}{\rightsquigarrow}) \quad (1)$$