Finite state automata

Data Structures and Algorithms for Computational Linguistics III
(ISCL-BA-07)

Çağrı Çöltekin
`ccoltekin@sfs.uni-tuebingen.de`

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

version: 090cabc @2021-02-03

Why study finite-state automata?

- Unlike some of the abstract machines we discussed, finite-state automata are efficient models of computation
- There are many applications
  - Electronic circuit design
  - Workflow management
  - Games
  - Pattern matching
  - ...

But more importantly ;-)
  - Tokenization, stemming
  - Morphological analysis
  - Shallow parsing/chunking
  - ...

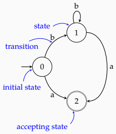
Finite-state automata (FSA)

- A finite-state machine is in one of a finite-number of states in a given time
- The machine changes its state based on its input
- Every regular language is generated/recognized by an FSA
- Every FSA generates/recognizes a regular language
- Two flavors:
  - *Deterministic finite automata* (DFA)
  - *Non-deterministic finite automata* (NFA)

Note: the NFA is a superset of DFA.

DFA as a graph

- States are represented as nodes
- Transitions are shown by the edges, labeled with symbols from an alphabet
- One of the states is marked as the *initial state*
- Some states are accepting states



DFA: formal definition

Formally, a finite state automaton, M, is a tuple $(\Sigma, Q, q_0, F, \Delta)$ with

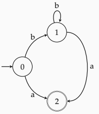
Σ is the alphabet, a finite set of symbols
Q a finite set of states
q_0 is the start state, $q_0 \in Q$
F is the set of final states, $F \subseteq Q$
Δ is a function that takes a state and a symbol in the alphabet, and returns another state $(\Delta : Q \times \Sigma \to Q)$

> At any given time, for any input,
> a DFA has a single well-defined action to take.

DFA: formal definition

an example

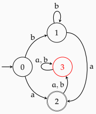
$$\begin{aligned}
\Sigma &= \{a, b\} \\
Q &= \{q_0, q_1, q_2\} \\
q_0 &= q_0 \\
F &= \{q_2\} \\
\Delta &= \{(q_0, a) \to q_2, \quad (q_0, b) \to q_1, \\
&\quad (q_1, a) \to q_2, \quad (q_1, b) \to q_1\}
\end{aligned}$$



Another note on DFA

error or sink state

- Is this FSA deterministic?
- To make all transitions well-defined, we can add a sink (or error) state
- For brevity, we skip the explicit error state
  - In that case, when we reach a dead end, recognition fails

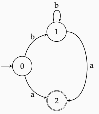


DFA: the transition table

transition table

		symbol	
		a	**b**
state	→**0**	2	1
	1	2	1
	***2**	∅	∅

→ marks the start state
* marks the accepting state(s)

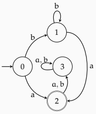


DFA: the transition table

transition table

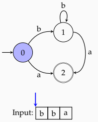
		symbol	
		a	**b**
state	→**0**	2	1
	1	2	1
	***2**	3	3
	3	3	3

→ marks the start state
* marks the accepting state(s)



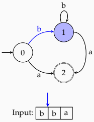
DFA recognition

1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input



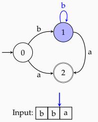
DFA recognition

1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input



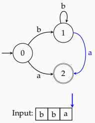
DFA recognition

1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input



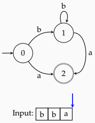
DFA recognition

1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input



DFA recognition

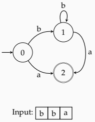
1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input



DFA recognition

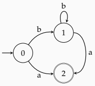
1. Start at q_0
2. Process an input symbol, move accordingly
3. Accept if in a final state at the end of the input

- What is the complexity of the algorithm?
- How about inputs:
  - bbbb
  - aa



A few questions

- What is the language recognized by this FSA?
- Can you draw a simpler DFA for the same language?
- Draw a DFA recognizing strings with even number of ‘a’s over $\Sigma = \{a, b\}$



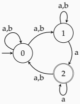
Non-deterministic finite automata

Formal definition

A non-deterministic finite state automaton, M, is a tuple $(\Sigma, Q, q_0, F, \Delta)$ with

Σ is the alphabet, a finite set of symbols
Q a finite set of states
q_0 is the start state, $q_0 \in Q$
F is the set of final states, $F \subseteq Q$
Δ is a function from (Q, Σ) to $P(Q)$, power set of Q $(\Delta : Q \times \Sigma \to P(Q))$

An example NFA



transition table

		symbol	
		a	**b**
state	→**0**	0,1	0,1
	1	1,2	1
	***2**	0,2	0

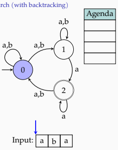
- We have nondeterminism, e.g., if the first input is a, we need to choose between states 0 or 1
- Transition table cells have *sets* of states

Dealing with non-determinism

- Follow one of the links, store alternatives, and *backtrack* on failure
- Follow all options in parallel

NFA recognition

as search (with backtracking)

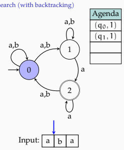


1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)

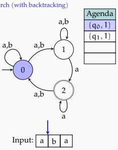


1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)

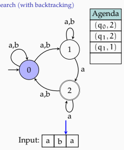


1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)

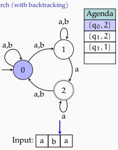


1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)

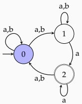


1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_0,3)$
$(q_1,3)$
$(q_1,2)$
$(q_1,1)$

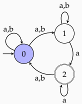
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_0,3)$
$(q_1,3)$
$(q_1,2)$
$(q_1,1)$

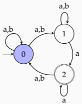
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_1,3)$
$(q_1,2)$
$(q_1,1)$

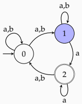
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_1,3)$
$(q_1,2)$
$(q_1,1)$

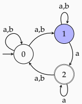
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_1,2)$
$(q_1,1)$

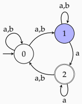
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_1,2)$
$(q_1,1)$

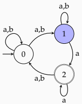
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_2,3)$
$(q_1,3)$
$(q_1,1)$

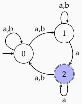
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_2,3)$
$(q_1,3)$
$(q_1,1)$

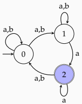
Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition

as search (with backtracking)



Agenda
$(q_1,3)$
$(q_1,1)$

Input: a b a

1. Start at q_0
2. Take the next input, place all possible actions to an *agenda*
3. Get the next action from the agenda, act
4. At the end of input

Accept if in an accepting state
Reject not in accepting state & agenda empty
Backtrack otherwise

NFA recognition as search

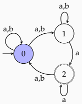
summary

- Worst time complexity is exponential
  - Complexity is worse if we want to enumerate all derivations
- We used a stack as *agenda*, performing a depth-first search
- A queue would result in breadth-first search
- If we have a reasonable heuristic A* search may be an option
- Machine learning methods may also guide finding a fast or the best solution

---

NFA recognition

parallel version



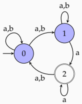
Input: a b a b

1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

---

NFA recognition

parallel version

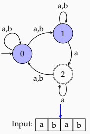


Input: a b a b

1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

NFA recognition

parallel version

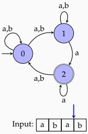


1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

---

NFA recognition

parallel version

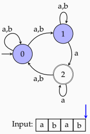


1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

---

NFA recognition

parallel version

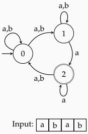


1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

---

NFA recognition

parallel version

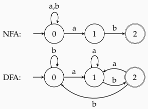


1. Start at q_0
2. Take the next input, mark all possible next states
3. If an accepting state is marked at the end of the input, accept

Note: the process is *deterministic*, and *finite-state*.

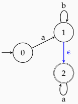
An exercise

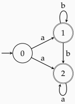
Construct an NFA and a DFA for the language over $\Sigma = \{a, b\}$ where all sentences end with ab.



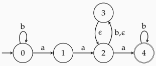
One more complication: ϵ transitions

- An extension of NFA, ϵ-NFA, allows moving without consuming an input symbol, indicated by an ϵ-transition (sometimes called a λ-transition)
- Any ϵ-NFA can be converted to an NFA





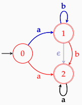
ϵ-transitions need attention



- How does the (depth-first) NFA recognition algorithm we described earlier work on this automaton?
- Can we do without ϵ transitions?

ϵ removal

- We start with finding the ϵ-*closure* of all states
  - ϵ-closure$(q_0) = \{q0\}$
  - ϵ-closure$(q_1) = \{q1, q2\}$
  - ϵ-closure$(q_2) = \{q2\}$
- Replace each arc to each state with arc(s) to all states in the ϵ-closure of the state

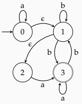


ϵ removal

a(nother) solution with the transition table

transition table

		symbol			
		a	b	ϵ	ϵ^*
state	**→0**	0	∅	1	0,1,2
	1	∅	1,3	2	1,2
	2	3	∅	∅	2
	***3**	3	1	∅	3



ϵ removal

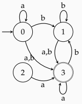
a(nother) solution with the transition table

transition table

		symbol			
		a	b	ϵ	ϵ^*
state	**→0**	0	∅	1	0,1,2
	1	∅	1,3	2	1,2
	2	3	∅	∅	2
	***3**	3	1	∅	3

⇒

	symbol	
	a	b
→0	0,3	1,3
1	3	1,3
2	3	∅
***3**	3	1



NFA–DFA equivalence

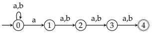
- The language recognized by every NFA is recognized by some DFA
- The set of DFA is a subset of the set of NFA (a DFA is also an NFA)
- The same is true for ϵ-NFA
- All recognize/generate regular languages
- NFA can automatically be converted to the equivalent DFA

Why do we use an NFA then?

- NFA (or ϵ-NFA) are often easier to construct
  - Intuitive for humans (cf. earlier exercise)
  - Some representations are easy to convert to NFA rather than DFA, e.g., regular expressions
- NFA may require less memory (fewer states)

A quick exercise – and a not-so-quick one

1. Construct (draw) an NFA for the language over $\Sigma = \{a, b\}$, such that 4th symbol from the end is an a



2. Construct a DFA for the same language

Summary

- FSA are efficient tools with many applications
- FSA have two flavors: DFA, NFA (or maybe three: ε-NFA)
- DFA recognition is linear, recognition with NFA may require exponential time
- Reading suggestion: **hopcroft1979** (and its successive editions), Jurafsky and Martin (2009, Ch. 2)

Next:

- FSA determinization, minimization
- Reading suggestion: **hopcroft1979** (and its successive editions), Jurafsky and Martin (2009, Ch. 2)

Acknowledgments, credits, references

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second edition. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

---

blank

---

blank

---

blank

---

blank