

Lecture 8

Ben Rosenberg

June 17, 2021

Non-deterministic finite automata

Definition: A **non-deterministic finite automaton** is a structure

$$M = (Q, \Sigma, \delta, q_0, F)$$

where Q is a finite set of states, Σ is an alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final (accepting) states, and the transition function δ is a *partial function*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q).$$

The initial configuration is (q_0, w_0) for a given input $w_0 \in \Sigma^*$. From a configuration (q, w) , M may transition to a next configuration (p, x) if:

$$w = ax \text{ where } x \in \Sigma \cup \{\lambda\}$$

and

$$[q, a, p] \in \delta.$$

M accepts a given input w_0 if there exists a sequence of transition that takes M from the initial configuration to an accepting configuration: (t, λ) for some $t \in F$.

Important distinctions between NFAs and DFAs:

- λ -transition: the automaton can move without receiving input
- δ is a partial function: it is possible that δ does not have an output for a given (state, input) pair
- δ maps to $\mathcal{P}(Q)$, the power set of Q – the machine may be at multiple states at the same time
 - $\delta : (q, a) \mapsto \{p_1, p_2, \dots, p_n\}, a \in \Sigma \cup \{\lambda\}$

Note that in the above definition, we say *a* next configuration rather than *the* next configuration. A machine can have multiple configurations at the same time. We use the *existential* quantifier here.

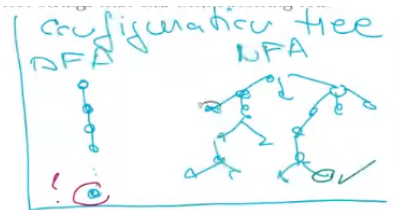


Figure 1: Configuration tree: difference between DFAs and NFAs

Example: Set of strings that end with `bba`.

“How to fail”: The complement theorem from DFAs cannot be applied to NFAs.

Example NFA: Set of exactly those strings that contain substring `babbabbba`.

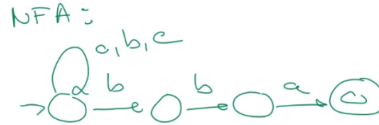


Figure 2: NFA for strings that end with bba

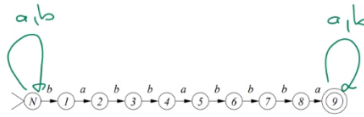


Figure 3: NFA for set of exactly those strings that contain substring babbabba

Kleene's Theorem

Theorem: The classes of languages defined by:

- regular expressions
- deterministic finite automata
- non-deterministic finite automata

are equivalent. These languages are called **regular languages**.

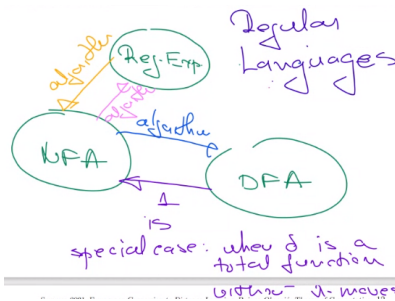


Figure 4: Relationship between NFAs, DFAs, and regular expressions

Algorithms

Algorithm ①: **Regular operations on finite state automata**

Input: finite state automata M_1 and M_2 , which satisfy the following form:

- initial state with zero in-degree
- single final state with zero out-degree

and accept languages L_1 and L_2 respectively:

Output: finite state automaton M that accepts language:

$$L = L_1 \cup L_2$$

$$L = L_1 \cdot L_2$$

$$L = L^*$$

In order to get the initial and final states correct for an NFA, simply add a λ -transitions from a new initial state with zero in-degree and from accept states to a new accept state with zero-out degree.

Case 1: $L_1 \cup L_2$

Take the two original machines, and add the following (red) transitions and states.

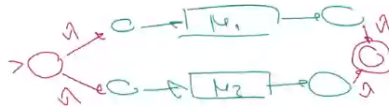


Figure 5: Union

Case 2: $L_1 \circ L_2$

Simply add a λ -transition from the accept state of one to the initial state of another.



Figure 6: Concatenation

Case 3: L_1^*

Add λ -transitions from the initial state to the accept state, and from the accept state to the initial state.

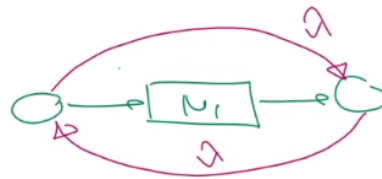


Figure 7: Kleene star

And so, the class of finite automata is closed under these operations.

Algorithm (2): **Conversion of regular expressions to finite state automata**

Input: regular expression e

Output: finite state automaton M that accepts the language defined by e

Construction: Recursion on number of operators in e .

Zero operators:

Recursively, if e has $n + 1$ operators, then we have 3 cases:

- $e = e_1 \cup e_2$
- $e = e_1 \circ e_2$
- $e = (e_1)^*$

where e_1, e_2 have n or fewer operators.

We apply the previous algorithm:

$$(a \cup bc^*)(b(a \cup b)^*c) \cup (bc \cup a)^*$$

“How to fail”: Be wary of assuming that automata are equivalent intuitively when they may not be.

Another example: Given $(a^*b)^*$, we know that $a \notin L$.

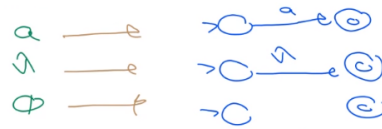


Figure 8: Zero-operator regular expressions

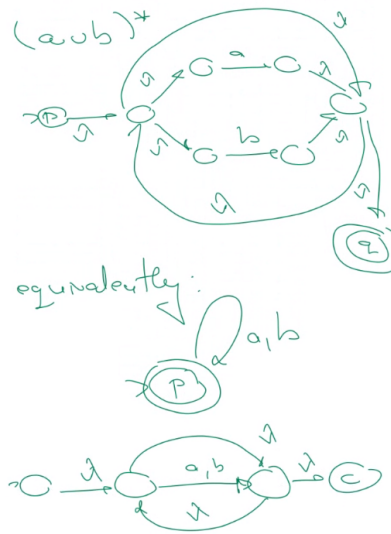


Figure 9: Three different ways to write $(a \cup b)^*$

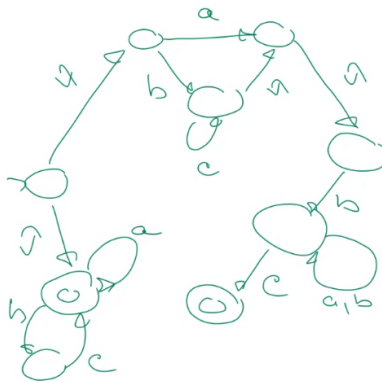


Figure 10: NFA for $(a \cup bc^*)(b(a \cup b)^*c) \cup (bc \cup a)^*$

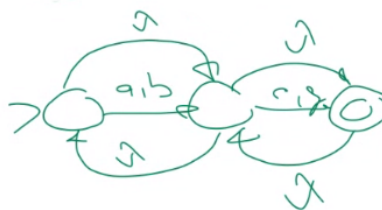


Figure 11: Example of **incorrect** NFA for $(a \cup b)^*(c \cup g)^*$ (should be λ -transition between parts)

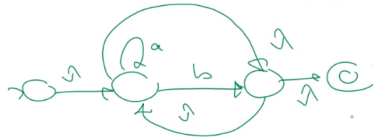


Figure 12: Example of **incorrect** NFA for $(a^*b)^*$ (it accepts $a \notin L$)

Algorithm ③: Conversion of NFAs to DFAs

Input: *non-deterministic* finite state automaton M that accepts the language L

Output: *deterministic* finite automaton M_1 that accepts the language L

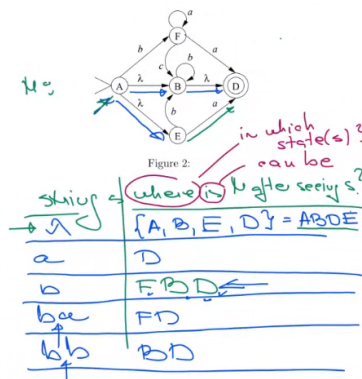


Figure 13: Example of algorithm #3

For every string $w \in \Sigma^*$, we know exactly the set of strings where M can be after w . There is nothing nondeterministic.

Idea: **States of the new deterministic automaton M_1 are sets of states of the old deterministic automaton M** , such that for every string $w \in \Sigma^*$, M_1 is in a state $X \subseteq Q$ after processing w if and only if M can be in any one of the elements of X after processing w , but in no other state.

Construction:

- $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
- $Q_1 = \mathcal{P}(Q)$

Note: our construction will construct only reachable states.

	a	b	c	λ
A	\emptyset	$\{F\}$	\emptyset	$\{B, E\}$
B	\emptyset	$\{B\}$	\emptyset	$\{D\}$
E	$\{D\}$	$\{B\}$	\emptyset	\emptyset
F	$\{F, D\}$	\emptyset	$\{B\}$	\emptyset
D	\emptyset	\emptyset	\emptyset	\emptyset

Figure 14: A suitable representation of the old δ for the NFA

Definition: The λ -closure of a state $X \in Q$ is defined recursively as follows:

Base case: $X \in \mathcal{C}(x)$ (X always belongs to its λ -closure)

Recursive case: If $y \in \mathcal{C}(x)$ and $[y, \lambda, z] \in \delta$ then $z \in \mathcal{C}(x)$.

$\mathcal{C}(x)$ is the set of states reachable from x using just λ -arcs.

	$\mathcal{C}(x)$
A	ABED
B	BD
E	E
F	F
D	D

Figure 15: Table of $\mathcal{C}(x)$'s

We say that the new initial state is the λ -closure of the old initial state: that is, $q_1 = \mathcal{C}(q_0)$.

New transition function δ_1 : we say that

$$\delta_1(x, a) = \bigcup_{p \in x} \delta(p, a)$$

Note that

$$\mathcal{C}(_ \cup _) = \mathcal{C}(_) \cup \mathcal{C}(_),$$

which we will use in determining

$$\delta_1(x, a) = \mathcal{C}\left(\bigcup_{p \in x} \delta(p, a)\right).$$

δ_1	a	b	c
$\{A, B, D, E\}$	$\{D\}$	$\{B, D, F\}$	\emptyset
$\{D\}$	\emptyset	\emptyset	\emptyset
$\{B, D, F\}$	$\{D, F\}$	$\{B, D\}$	$\{B, D\}$
\emptyset	\emptyset	\emptyset	\emptyset
$\{B, D\}$	\emptyset	$\{B, D\}$	\emptyset
$\{D, F\}$	$\{D, F\}$	\emptyset	$\{B, D\}$

Figure 16: New transition function (of dubious correctness)

Final (accept) states: any state that contains at least one of the old final states

Algorithm (4): **Conversion of finite automaton to regular expression**

Input: finite automaton M with zero in-degree and single final state with zero out-degree

Output: regular expression e


Def: A **generalized expression graph** (GEG) is *not* a finite automaton – its arcs are labelled by regular expressions.

Idea: transform the original automaton through a sequence of equivalent generalized expression graphs until a trivial one is obtained, which will be read off the result.

The NFA is a special case of a generalized expression graph, where the arc label is either a single symbol or λ .



Figure 17: Example G.E.G.

We want to keep modifying our NFA until we get something of the form .

We need to arrive at 2 nodes, which means that we will need to eliminate some number of nodes. For an automaton with k nodes, we will need to eliminate $k - 2$ nodes.

Example: Odd number of b's

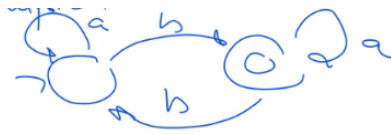


Figure 18: Odd number of b's

We need to make this into a compliant automaton.



Figure 19: Compliant automaton with odd number of b's

Then, we make a table that is kind of an adjacency matrix. For the above automaton, it looks like:

For every pair of nodes (p, q) , we need a new matrix entry.

Back to the previous example: which nodes do we need to eliminate? We should eliminate 2 and 3, as they are in the middle of our good nodes 1 and 4. We start by trying to eliminate node 2.

So, we make a table as before, but without node 2:

Note that if any of the expressions are \emptyset , then the arc does not change.

And so, the regular expression for an odd number of b's is $a^*b(a \cup ba^*b)^*$.

	1	2	3	4
1	\emptyset	a	b	\emptyset
2	\emptyset	a	b	\emptyset
3	\emptyset	b	a	a
4	\emptyset	\emptyset	\emptyset	\emptyset

Figure 20: Adjacency matrix-like construct

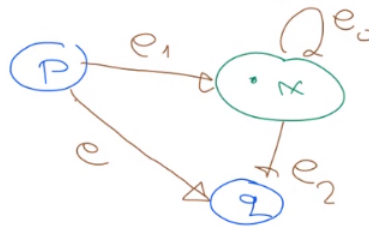


Figure 21: Before x is eliminated

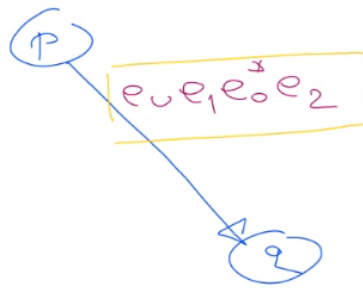


Figure 22: After x is eliminated

	1	3	4
1	\emptyset	a^*b	\emptyset
3	\emptyset	au ba^*b	a
4	\emptyset	\emptyset	\emptyset

Figure 23: Table without node #2

	1	4
1	\emptyset	$a^*b(aub^*a^*b)^*$
4	\emptyset	\emptyset

Figure 24: Table without node #3