Nondeterministic Finite Automata (NFA)

Jay Bagga

1 Introduction

In the module on Deterministic Finite Automata (DFA) you learned that a DFA is a finite state machine that accepts or rejects finite strings of symbols and produces the same unique computation for each unique input string. This unique computation is what the term *deterministic* means. In particular, for any state of a DFA, the state transition function specifies exactly one state for each input symbol in the alphabet. We now introduce a generalization of this concept. In this module, we will study *Nondeterministic* Finite Automata. In a non-deterministic finite automation, for a given state and input symbol, none or several choices may be available for the next state. Nondeterminism is a powerful concept in computation which also has many practical applications. Let us motivate this concept with an example.

2 Pattern matching

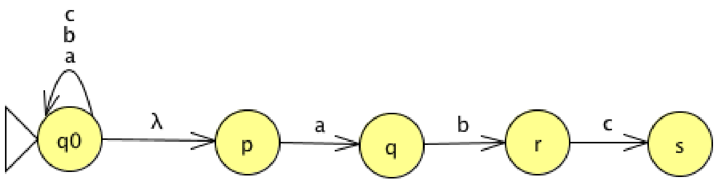
A common everyday task that many of you do is to search for a pattern in some text. Specifically, suppose we have a set of words and we want all the words that contain a string w as a substring with or without a specified location. To simplify our discussion, suppose our alphabet is $\{a, b, c\}$ and we are interested in the set of all words that contain the string abc or that end with the string ac. Thus a word w is in our set if the string abc occurs anywhere in w or if w is a word of any length that ends with ac. Some examples of such words are $w_1 = aaabc$, $w_2 = babcabcac$, $w_3 = aaac$. On the other hand, the string $w_4 = abbccaab$ is not in the desired set.

Note that w_2 has the substring abc in multiple locations and also ends with ac. We try to construct a finite machine that may accept such words. First, suppose we want the string abc anywhere in the word. In terms of states and transitions, we want the configuration shown below to be a part of our finite machine.

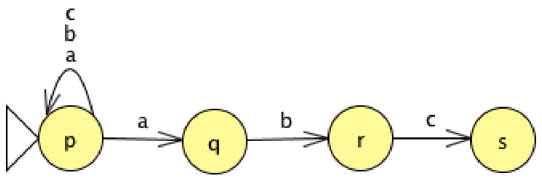
$$\begin{tikzcd} p \arrow[r, "a"] & q \arrow[r, "b"] & r \arrow[r, "c"] & s \end{tikzcd}$$

Thus we can move from the state p to the state s with the string abc. Since this occurrence of abc can be anywhere in the word, we want to be able to *reach* the state p at any step. Also, once we reach s, we have found the substring abc and now we want to accept the word with any combination of input symbols following this occurrence of abc. The concept of nondeterminism comes into play here and one way we can accomplish this task is as follows.

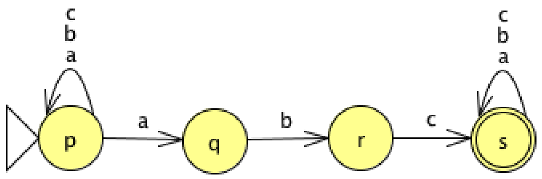
We introduce an initial state $q0$ with transitions shown as below. The transitions labeled a, b and c at $q0$ can read any combination of the symbols in the alphabet, while the transition labeled λ allows one to "*jump*" from $q0$ to p without reading an input symbol. Thus we may "*guess*" where an abc might occur to make this jump.



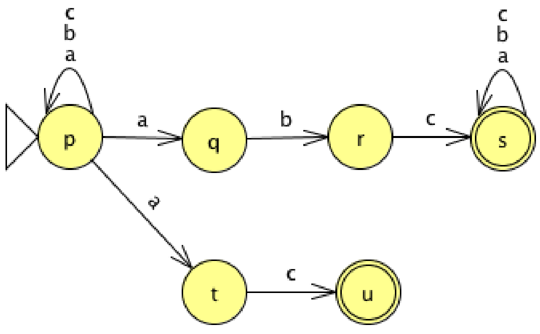
An equivalent way to achieve this without the λ transition is as shown below. Here the state p is the initial state. We observe that there are two transitions labeled a from the state p. One stays in p while the other goes to q. This nondeterministic behavior allows one to make the jump to the state q when looking for the substring abc.



Next, we introduce transitions a, b and c at state s and make s an accept state, as shown below. Thus any word containing the string abc is accepted.



Finally, to accept words that end with ac, we introduce two more states and transitions as shown below. This is a Nondeterministic Finite Automaton (NFA) that accepts our desired set.



We note several properties of this NFA.

- There are three transitions from state p labeled a. Another way of saying this is that the transition labeled a from state p goes to the set $\{p, q, t\}$.
- There are no transitions labeled a or c from the state q. Another way of saying this is that the transitions labeled a and c from state q go to the empty set ϕ.
- There can be transitions labeled λ, as discussed above.

Properties such as above distinguish an NFA from a DFA. Another consequence of nondeterminism is that there may be several possible paths for a given input string. For example, for the input $w = ababc$, there are several configurations in the NFA above. For example, let us look at the configuration below.

$$p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{a} q \xrightarrow{b} r \xrightarrow{c} s$$

.

Since s is an accept state, the above configuration shows that the input w is accepted. However, we also have,

$$p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{c} p$$

.

This configuration ends in p which is not an accept state. We observe that an NFA accepts an input if there is at least one accepting configuration. Now consider the following configuration for the same input $w = ababc$.

$$p \xrightarrow{a} p \xrightarrow{b} p \xrightarrow{a} t \xrightarrow{b} \phi \xrightarrow{c} \phi$$

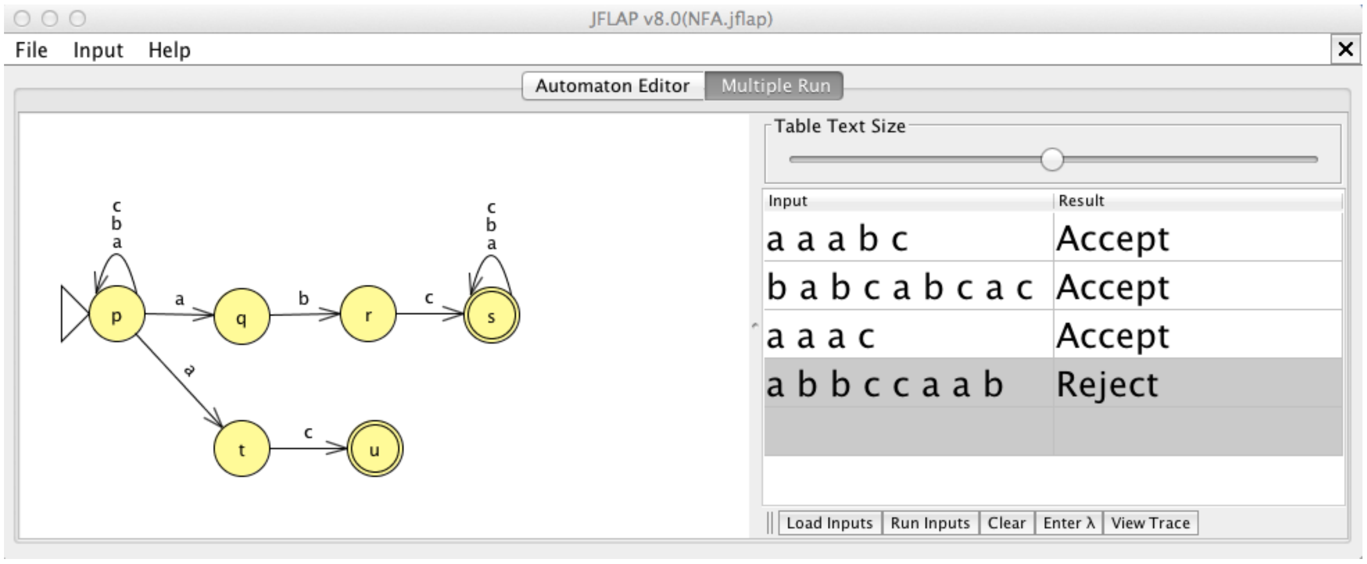
.

Since there is no transition from t labeled b, we interpret this is as a transition going to the empty set ϕ. This state is generally not shown in the state transition diagram. Any transition from ϕ stays in ϕ. Can you see any other configurations for the input $w = ababc$?

Before studying the formal definition of an NFA given below, you should practice by testing the NFA discussed above with several inputs.

Question 1.

1. Load the NFA in the file NF-1.jflap.
2. Enter inputs w_1, w_2, w_3 and w_4 in the discussion above and verify that the first three inputs are accepted and the last is rejected, as shown in the diagram below.
3. List all configurations for the inputs w_1, w_2, w_3 and w_4.
4. Enter six more inputs of your own, three of which are accepted and the other three are rejected.



Input	Result
a a a b c	Accept
b a b c a b c a c	Accept
a a a c	Accept
a b b c c a a b	Reject

3 Nondeterministic Finita Automata (NFA)

We can now describe NFA formally. An NFA is a 5-*tuple* $(Q, \Sigma, \delta, q0, F)$, where

- Q is a finite set of states
- Σ is a finite set of input symbols (the alphabet)
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q)$ is the state transition function, where $\mathcal{P}(Q)$ is the power set of Q, the set of all subsets of Q
- $q0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

An input w is accepted by an NFA if there is at least configuration for w that ends in an accept state, that is a state that belongs to F. As mentioned above, there may be several configurations for an input. The state transition function can also be represented as a table, as shown next for the NFA above. We have filled this table partially and you are asked to complete the table in an exercise below.

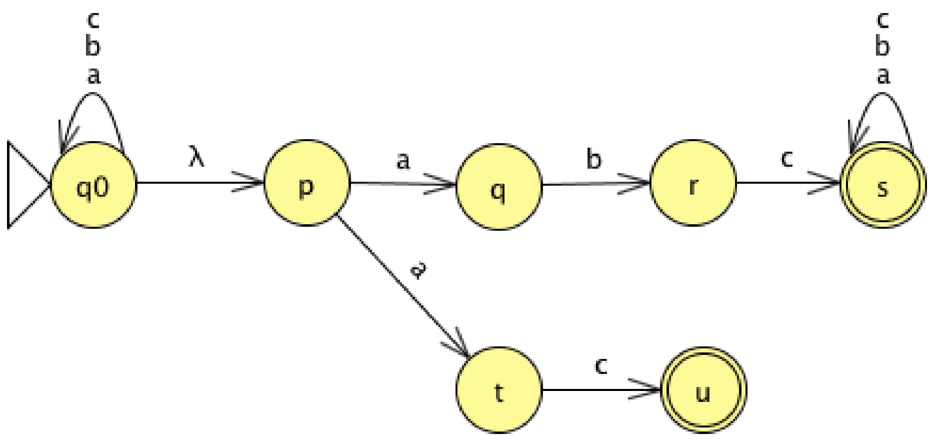
Table 1: State Transition Table

	a	b	c	λ
p	$\{p, q, t\}$	$\{p\}$	$\{p\}$	ϕ
q	ϕ	$\{r\}$		
r				
s	$\{s\}$	$\{s\}$	$\{s\}$	
t				
u	ϕ	ϕ		ϕ

Question 2. Complete the table above by filling out all blank cells in the table. Explain your answers.

Question 3.

1. Load the NFA in the file NFA-2.jflap, as shown below.
2. Verify that $Q = \{q0, p, q, r, s, t, u\}$.
3. Which state is the initial state?
4. What is F?
5. Verify that $\delta(p, a) = \{q, t\}$.
6. What is $\delta(q0, \lambda)$?
7. What is $\delta(r, b)$?
8. What is $\delta(t, \lambda)$?
9. What is $\delta(u, b)$?
10. What is $\delta(s, c)$?



Question 4. Do NFA-1.jflap and NFA-2.jflap accept the same set of words? Explain your answer in detail.

4 Summary

We saw above that an NFA is a generalization of a DFA. In a DFA there is a unique (exactly one) transition for every state and symbol pair. In an NFA, there can be zero or more transitions for a given state and a symbol. Furthermore, transitions with λ are also allowed. It follows that every DFA is also an NFA. Later, you'll see other types of finite machines that have nondeterministic generalizations, which are in general more powerful, i.e. they accept more sets of words than their deterministic counterparts. However, it turns out that for every NFA there is an equivalent DFA, one that accepts the same set of words as the NFA. In the next module, you will learn an algorithm for converting a given NFA into an equivalent DFA and you will use JFLAP to run this algorithm.

5 References

1. Introduction to the Theory of Computation (Third Edition), Michael Sipser. Cengage Learning. 2013.
2. JFLAP - An Interactive Formal Languages and Automata Package, Susan H. Rodger and Thomas W Finley. Jones and Bartlett Publishers. 2006