Today's lecture: Nondeterministic Finite Automata (NFAs)

I. Nondetermistic Finite Automata **II.** Formal Definition of NFAs **III.** Applications of Nondeterminism

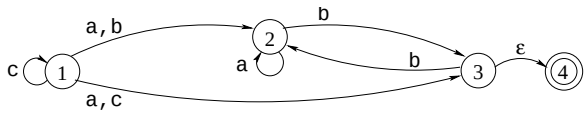
Schedule:

Today: NFAs
Lecture will cover through Example 1.38 (i.e. pages 53-54).
Homework 0 due. item**September 20:** Equivalence of DFAs and NFAs.
The rest of *Sipser* 1.2 (i.e. pages 54–63).

September 22: Regular Expressions – Read: *Sipser* 1.3.
Lecture will cover through Example 1.58 (i.e. pages 63-59). Homework 2 goes out (due Oct. 2).

September 25: Equivalence of DFAa and Regular Expressions.
The rest of *Sipser* 1.3 (i.e. pages 63–76).
Homework 1 due.

September 27 and beyond: see Sept. 6 notes.



1. A state can have multiple outgoing arcs for the same input symbol.
2. A state can have no outgoing arcs for some input symbol.
3. A state can have arcs that are taken without consuming any input symbol.

Figure 1: The pieces of a NFA

I. Nondetermistic Finite Automata

A. Transition graphs for NFAs:

1. A collection of states and arcs.

2. States drawn with a double circle are accepting.

3. States drawn with a single circle are non-accepting.

4. Arcs are labeled with the input symbol(s) for which they can be taken.

5. A state may have multiple outgoing arcs labeled with the same input symbol. If that symbol is read, the machine may move along *any* of those arcs.

6. A state may have no outgoing arcs labeled for some input symbol. If that symbol is read, the machine immediately rejects.

7. A state may have arcs labeled ϵ. When such an arc is taken, no input is read.

8. An NFA accepts a string if there is some set of choices for the nondeterministic transitions that lead to an accepting state after reading the complete string.

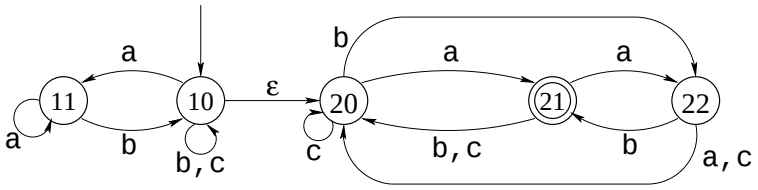


Figure 2: The NFA from the Sept. 15 notes

B. An example

1. Figure 2 shows the NFA that we derived in the Sept. 15 notes.

2. Accepting a string

a. A NFA accepts a string if there is at least one way to make the nondeterministic choices when reading that string such that the machine ends in an accepting state after reading the last symbol of the string.

b. To show that a NFA accepts a string, we just have to show a sequence of states that the machine is allowed to go through when reading the string such that the final state is accepting.

c. Consider reading the string: `abcabbbb`.

- The machine starts in state 10.
- The following sequence of states shows that `abcabbbb` is accepted:

$$10 \xrightarrow{a} 11 \xrightarrow{b} 10 \xrightarrow{c} 10 \xrightarrow{a} 11 \xrightarrow{b} 10 \xrightarrow{b} 10 \xrightarrow{\epsilon} 20 \xrightarrow{b} 22 \xrightarrow{b} 21$$

- State 21 is an accepting state. Therefore, the string is accepted.
- Note that there are other choices we could have made that would have led to a non-accepting state; for example:

$$10 \xrightarrow{a} 11 \xrightarrow{b} 10 \xrightarrow{c} 10 \xrightarrow{\epsilon} 20 \xrightarrow{a} 21 \xrightarrow{b} 20 \xrightarrow{b} 22 \xrightarrow{b} 21 \xrightarrow{b} 20$$

That's OK. The machine accepts if there is at least one way to make the choices so that the machine reaches an accepting state after reading the last symbol of the string.

3. Rejecting a string

a. A NFA rejects a string if there is not way to make the nondeterministic choices when reading that string such that the machine ends in an accepting state after reading the last symbol of the string.

b. To show that a NFA rejects a string, we have to show that *all* sequences of states that the machine is allowed to go through when reading the string lead to final states that are non-accepting.

c. Consider reading the string: `abcacbbb`.

- This string is rejected. Consider what state the machine is in after reading `abc`. If it's in state 10 or 11, then reading the `a` symbol of `acbbb` will move the machine to state 11, and the machine will reject when it reads the `c` (because state 11 has no outgoing edge for symbol `c`).
  On the other hand, if the machin is in state 20, 21 or 22 after reading `abc`, then it will be in state 20 after reading the `ac` symbols of `acbbb`. Then, the `bbb` string will lead it through the sequence of states below:

$$20 \xrightarrow{b} 22 \xrightarrow{b} 21 \xrightarrow{b} 20$$

  State 20 is not an accepting state, so the machine rejects the string.
- While the argument above shows that `abcacbbb` is rejected, it would be very tedius if we had to find a new way to analyze each string for each NFA to come up with a proof for acceptance or rejection. We would like a general method that will work with all NFAs and strings.

- Instead of considering individual trajectories, we can consider *all possible* at once by keeping track of the *set* of states that the NFA could possibly be in:

$$
\begin{array}{rrll}
\text{initial state:} & \{10\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{a}} & \\
& \{11, 21\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 20\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{c}} & \\
& \{10, 20\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{a}} & \\
& \{11, 21\} & \xrightarrow{\mathsf{c}}, & \text{(state 11 goes “Poof!”)} \\
& \{20\} & \xrightarrow{\mathsf{b}} & \\
& \{22\} & \xrightarrow{\mathsf{b}} & \\
& \{21\} & \xrightarrow{\mathsf{b}} & \\
& \{20\}, & & \text{reject } \text{☹}
\end{array}
$$

  This shows that the *only* state that our NFA can reach for this input string is state 20. State 20 is not an accepting state. Therefore, the string `abcacbbb` is rejected.

4. Accepting again.

a. We can use the method described above to show that a string is accepted.

b. Consider again reading the string: `abcabbbb`.

$$
\begin{array}{rrll}
\text{initial state:} & \{10\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{a}} & \\
& \{11, 21\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 20\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{c}} & \\
& \{10, 20\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{a}} & \\
& \{11, 21\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 20\} & \xrightarrow{\epsilon} & \\
& \{10, 20\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 22\} & \xrightarrow{\epsilon} & \\
& \{10, 20, 22\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 21, 22\} & \xrightarrow{\epsilon} & \\
& \{10, 20, 21, 22\} & \xrightarrow{\mathsf{b}} & \\
& \{10, 20, 21, 22\} & \xrightarrow{\epsilon} & \\
& \{10, 20, 21, 22\}, & & \text{the set of possible final states}
\end{array}
$$

The set of possible final states includes state 21, the accepting state. Therefore, the string is accepted.

II. Formal Definition of NFAs

A. The pieces of an NFA

1. A NFA is a 5-tuple: $(Q, \Sigma, \delta, q_0, F)$ where

Q is a finite set of states.

Σ is a finite set of symbols (the alphabet).

$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the state transition relation.

$q_0 \in Q$ is the initial state.

$F \subseteq Q$ is the set of accepting states.

2. Comparison with a DFA

a. Q, Σ, q_0, and F are the same.

b. δ is different.

DFA: $\delta : Q \times \Sigma \to Q$.

NFA: $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$

The DFA maps a (state, input-symbol) pair to the next state. For the NFA, recall that 2^Q is the power set of Q (i.e. the set of all subsets of Q, see *Sipser*, p. 6, Sipser writes $\mathcal{P}(Q)$ where I wrote 2^Q, either is acceptable). Thus, an NFA maps a (state, input-symbol) pair to the set of possible next states. The machine can move to any one of these sets.

The other difference is that the NFA includes mappings for (state, ϵ) pairs. These describe moves that the machine can make without consuming an input symbol.

B. Acceptance

1. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w \in \Sigma^*$ be a string. N accepts w iff there is a sequence of state transitions using w as input that leads to an accepting state after w has been consumed.

2. A mathematical definition for machines without ϵ transitions.

a. We extend δ to apply to sets of states and strings:

$$\begin{aligned} \delta(B, \epsilon) &= B \\ \delta(B, x \cdot c) &= \bigcup_{q \in \delta(B,x)} \delta(q, c) \end{aligned}$$

In English, $\delta(B, w)$ computes all states reachable starting from some state in B, making moves that are allowed when reading w. If $w = \epsilon$, then $\delta(B, w) = B$ (because we're ignoring ϵ moves for now). If $w = x \cdot c$, we first find all states that are reachable by starting from a state in B an reading x – thats the $q \in \delta(B, x)$ part of the defintion. For each such state, q, we find where the machine can go when reading c – that's $\delta(q, c)$ – and we take the union of all of those sets. That union is the set of all states that can be reached starting from a state in B and reading w.

b. N accepts w iff $(\delta(\{q_0\}, w) \cap F) \neq \emptyset$.

In English, that says we find all states that can be reached by starting in state q_0 and reading w. If any of these states are in F, then it is possible for N to reach an accepting state when reading w; thus, N accepts w.

3. Handling ϵ transitions.

a. For each state, q, we define a set $\phi(q)$ of states that can be reached by zero or more ϵ moves starting from q. State p is in $\phi(q)$ iff

- $p = q$, or
- $\exists r \in \phi(q).\ p \in \delta(r, \epsilon)$.

b. We now extend $\phi(q)$ to apply to sets:

$$\phi(B) = \bigcup_{q \in B} \phi(q)$$

c. We now extend $\delta(B, w)$ to include ϵ moves:

$$\begin{aligned} \delta(B, \epsilon) &= \phi(B) \\ \delta(B, x \cdot c) &= \phi\left(\bigcup_{q \in \delta(B,x)} \delta(q, c) \right) \end{aligned}$$

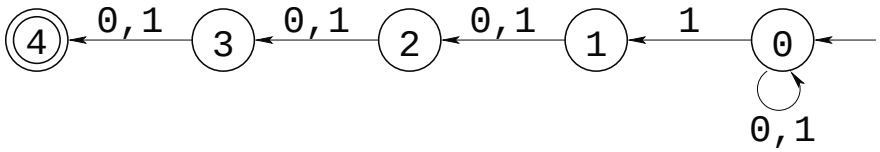


Figure 3: Another NFA example

In English, this says that for each input symbol, we calculate δ as before. Then, for each state that we reach, we find all states that are reachable from it by taking ϵ moves.

d. N accepts w iff $(\delta(\{q_0\}, w) \cap F) \neq \emptyset$.
The same as before in II.B.2.b, we've just extended the definition of δ to handle ϵ moves.

C. An Example – see figure 3

1. This NFA accepts 011010:

$$\begin{array}{lrll} \text{initial state:} & \{0\} & \xrightarrow{0} & \\ & \{0\} & \xrightarrow{1} & \\ & \{0,1\} & \xrightarrow{1} & \\ & \{0,1,2\} & \xrightarrow{0} & \\ & \{0,2,3\} & \xrightarrow{1} & \\ & \{0,1,3,4\} & \xrightarrow{0} & \\ & \{0,2,4\}, & & (\{0,2,4\} \cap \{4\} = \{4\}) \Rightarrow \text{accept } \text{☺} \end{array}$$

2. This NFA rejects 10111:

$$\begin{array}{lrll} \text{initial state:} & \{0\} & \xrightarrow{1} & \\ & \{0,1\} & \xrightarrow{0} & \\ & \{0,2\} & \xrightarrow{1} & \\ & \{0,1,3\} & \xrightarrow{1} & \\ & \{0,1,2,4\} & \xrightarrow{1} & \\ & \{0,1,2,3\}, & & (\{0,1,2,3\} \cap \{4\} = \emptyset) \Rightarrow \text{reject } \text{☹} \end{array}$$

III. Applications of Nondeterminism

A. Modeling the environment of a program

B. Avoiding over-specification