Warm Up Problem

- What is the definition of a DFA? (Try it without looking!)
- Write a DFA over $\Sigma = \{a, b\}$ that accepts all words ending with $bba$. 
CS 241 Lecture 7
Non-Deterministic Finite Automata
With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot
Recall Regular Language

A **regular language** over an alphabet $\Sigma$ consists of one of the following:

1. The empty language and the language consisting of the empty word are regular.
2. All languages $\{a\}$ for all $a \in \Sigma$ are regular.
3. The union, concatenation or Kleene star of any two regular languages are regular.
A **DFA** is a 5-tuple \((\Sigma, Q, q_0, A, \delta)\):

- \(\Sigma\) is a finite non-empty set (alphabet).
- \(Q\) is a finite non-empty set of states.
- \(q_0 \in Q\) is a start state
- \(A \subseteq Q\) is a set of accepting states
- \(\delta : (Q \times \Sigma) \rightarrow Q\) is our [total] transition function (given a state and a symbol of our alphabet, what state should we go to?).
Extending $\delta$

We can extend the definition of $\delta : (Q \times \Sigma) \rightarrow Q$ to a function defined over $(Q \times \Sigma^*)$ via:

$$\delta^* : (Q \times \Sigma^*) \rightarrow Q$$

$$(q, \epsilon) \mapsto q$$

$$(q, aw) \mapsto \delta^* (\delta(q, a), w)$$

where $a \in \Sigma$ and $w \in \Sigma^*$ ($aw$ is concatenation). Basically, if processing a string, process a letter first then process the rest of a string. In this way...

**Definition**

A DFA given by $M = (\Sigma, Q, q_0, A, \delta)$ accepts a string $w$ if and only if $\delta^*(q_0, w) \in A$. 
Language of a DFA

With the previous slide we can make one more definition.

**Definition**

Denote the **language of a DFA** $M$ to be the set of all strings accepted by $M$, that is:

$$L(M) = \{w : M \text{ accepts } w\}$$
In a future course (CS 360/365), you will prove the following beautiful result:

**Theorem (Kleene)**

\[ L \text{ is regular if and only if } L = L(M) \text{ for some DFA } M. \] 
That is, the regular languages are precisely the languages accepted by DFAs.
Implementing a DFA

Algorithm 1 DFA algorithm

1:  \( s = q_0 \)
2:  \textbf{while} not EOF do
3:    \textbf{read} character \( ch \)
4:    \textbf{switch} \((s)\)
5:      \textbf{case} \( q_0 \): 
6:        \textbf{switch} \((ch)\)
7:          \textbf{case} \( ch = a_0 \):
8:            \( s = \text{new\_state\_a}_0 \)
9:          \textbf{case} \( ch = a_1 \):
10:             \( s = \text{new\_state\_a}_1 \)
11:          \ldots
12:          \textbf{case} \( ch = a_{|\Sigma|} \):
13:              \( s = \text{new\_state\_a}_{\sigma} \)
14:          \textbf{end switch}
15:      \textbf{case} \( q_1 \):
16:      \ldots
17:  \textbf{end switch}
18:  \textbf{end while}
Alternatively

You could also use a lookup table:

|        | \(q_0\) | \(q_1\) | \(\ldots\) | \(q_{|Q|}\) |
|--------|---------|---------|------------|-------------|
| \(a_0\) |         |         |            |             |
| \(a_1\) |         |         |            |             |
| \(\vdots\) |     |         |            |             |
| \(a_{|\Sigma|}\) |         |         |            |             |

where above, the blank table entries would be the next states.

Check out the provided assembler starter code in your assignment!
Extension to DFAs

We could also have DFAs where we attach actions to arcs.

- For example, consider a subset of the language of binary numbers without leading zeroes below.
- We’ll create a DFA where we also compute the decimal value of the number simultaneously. Could then print the value.
- Look at the DFA corresponding to $1(0 \mid 1)^*1$.
- In what follows, you should read $1/N \leftarrow 2N + 1$ as the leftmost 1 corresponds to a DFA transition, the $/$ has no meaning and the $N \leftarrow 2N + 1$ changes $N$ to be $2N + 1$.
Revisiting our Warm Up

What happens if we make out DFAs more complex? Let’s revisit our warmup example from today over the alphabet $\Sigma = \{a, b\}$:

$$L = \{w : w \text{ ends with } bba\}$$
Imagine

But what if we allowed more than one transition from a state?

Does such a thing make sense? Do we can any computability power from this?
Multiple Transitions

- When we allow for a state to have multiple branches given the same input, we say that the machine *chooses* which path to go on.
- This is called *non-determinism*.
- We then say that a machine accepts a word $w$ if and only if there exists *some* path that leads to an accepting state!
- We can then simplify the previous example to an NFA as defined on the next slide:
Simplified NFA

$L = \{w : w \text{ ends with } bba\}$

Machine “guesses” to stay in first state until $bba$ is seen. How does a machine do this?
Non-Deterministic Finite Automata

The above idea can be mathematically described as follows:

**Definition**

An **NFA** is a 5-tuple \((\Sigma, Q, q_0, A, \delta)\):

- \(\Sigma\) is a finite non-empty set (alphabet).
- \(Q\) is a finite non-empty set of states.
- \(q_0 \in Q\) is a start state
- \(A \subseteq Q\) is a set of accepting states
- \(\delta : (Q \times \Sigma) \to 2^Q\) is our [total] transition function. Note that \(2^Q\) denotes the *power set* of \(Q\), that is, the set of all subsets of \(Q\). This allows us to go to multiple states at once!
Language of a NFA

Similar to before, we have the following definition:

**Definition**

Let $M$ be an NFA. We say that $M$ **accepts** $w$ if and only if there exists *some* path through $M$ that leads to an accepting state.

Denote the **language of an NFA** $M$ to be the set of all strings accepted by $M$, that is:

$$L(M) = \{ w : M \text{ accepts } w \}$$
Extending $\delta$ For an NFA

Again we can extend the definition of $\delta : (Q \times \Sigma) \rightarrow 2^Q$ to a function $\delta^* : (2^Q \times \Sigma^*) \rightarrow 2^Q$ via:

$$\delta^* : (2^Q \times \Sigma^*) \rightarrow 2^Q$$

$$(S, \epsilon) \mapsto S$$

$$(S, aw) \mapsto \delta^* \left( \bigcup_{q \in S} \delta(q, a), w \right)$$

where $a \in \Sigma$. Analogously, we also have:

**Definition**

An NFA given by $M = (\Sigma, Q, q_0, A, \delta)$ accepts a string $w$ if and only if...
Extending $\delta$ For an NFA

Again we can extend the definition of $\delta : (Q \times \Sigma) \rightarrow 2^Q$ to a function $\delta^* : (2^Q \times \Sigma) \rightarrow 2^Q$ via:

\[
\delta^* : (2^Q \times \Sigma^*) \rightarrow 2^Q \\
(S, \varepsilon) \mapsto S \\
(S, aw) \mapsto \delta^* \left( \bigcup_{q \in S} \delta(q, a), w \right)
\]

where $a \in \Sigma$. Analogously, we also have:

Definition

An NFA given by $M = (\Sigma, Q, q_0, A, \delta)$ accepts a string $w$ if and only if $\delta^*({q_0}, w) \cap A \neq \emptyset$. 
Simulating an NFA

Algorithm 2 Algorithm to Simulate an NFA

1: $S = \{q_0\}$
2: while not EOF do
3:   $c = \text{read\_char}()$
4:   $S = \bigcup_{q \in S} \delta(q, c)$
5: end while
6: if $S \cap A \neq \emptyset$ then
7:   Accept
8: else
9:   Reject
10: end if
Practice Simulating $w = abbba$

Since $\{q_0, q_3\} \cap \{q_3\} \neq \emptyset$, accept.