Warm-Up Problem

Write a DFA over Σ = {a, b} that...

- Accepts only words with an even number of *a*s
- Accepts only words with an odd number of *a*s and an even number of *b*s
- Accepts only words where the parity of the number of *a*s is equal to the parity of the number of *b*s
- What is the definition of a DFA? (Try it without looking!)
- Write a DFA over Σ = {a, b} that accepts all words ending with bba.

CS 241 Lecture 7

Non-Deterministic Finite Automata

With thanks to Brad Lushman, Troy Vasiga, Kevin Lanctot, and Carmen Bruni

Recall Regular Language

Definition

A **regular language** over an alphabet Σ 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 (pronounced klay-nee) of any two regular languages are regular.
4. Nothing else.

Recall: Deterministic Finite Automata

Definition

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

- Σ 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 δ

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

$$\begin{aligned} \delta^* : \quad & (Q \times \Sigma^*) \to Q \\ & (q, \varepsilon) \to q \\ & (q, aw) \to \delta^*(\delta(q, a), w) \end{aligned}$$

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

The **language of a DFA** M is the set of all strings accepted by M, that is:

$$L(M) = \{w : M \text{ accepts } w\}$$

A Beautiful Result

In a future course (CS 360/365), you will prove the following beautiful result:

Theorem (Kleene)

L is regular if and only if $L = L(M)$ for some DFA M. That is, the regular languages are precisely the languages accepted by DFAs.

Implementing a DFA

Algorithm 2 DFA Recognition Algorithm

w $= a_1 a_2 \ldots . a_n$
$s = q_0$
for i in 1 to n **do**
    s $= \delta(s, a_i)$
end for
if $s \in A$ **then**
    Accept
else
    Reject
end if

δ Itself

You could also use a lookup table:

	q_0	q_1	$\dots$	$q_{\|Q\|}$
a_0				
a_1				
.				
$a_{\|\Sigma\|}$				

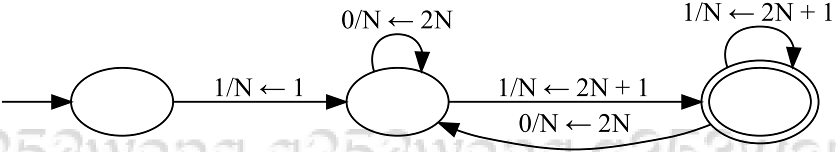
Above, the blank table entries would be the next states.

Check out the provided assembler starter code!

Extension to DFAs

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

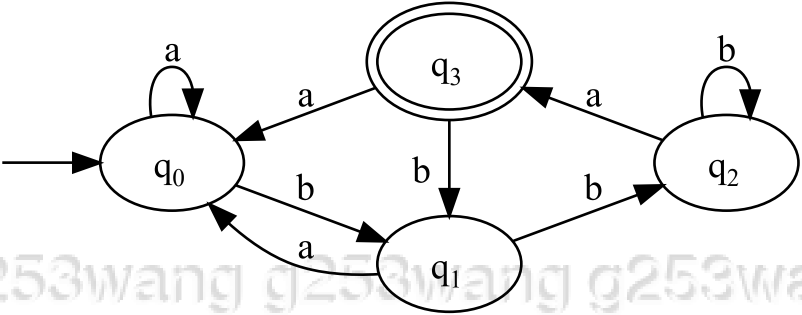
- For example, consider a subset of the language of binary numbers without leading zeroes described below.
- We'll create a DFA where we also compute the decimal value of the number simultaneously. Could then print the value.
- Look at the DFA corresponding to $1(0 \mid 1)^*1$.
- In what follows, you 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

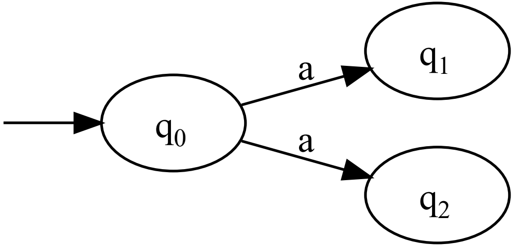
What happens if we make out DFAs more complex? Let's revisit our warmup example from today over the alphabet $\Sigma = \{a, b\}$:

- L = {w : w ends with bba}



Imagine

But what if we allowed more than one transition from a state with the same symbol?



Does such a thing make sense? Do we gain any computability power from this?

Multiple Transitions

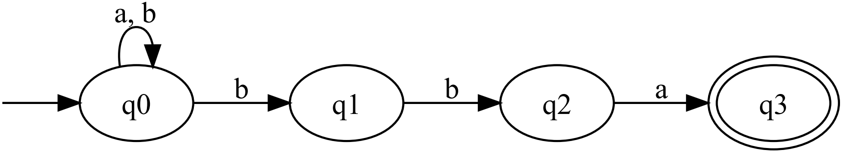
- When we allow for a state to have multiple branches given the same input, we say that the machine “chooses” which path to go on.
- To make the right choice, we would need an oracle that can predict the future, so to actually implement this, we would need to try every choice (yuck!)

Multiple Transitions

- 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.

Language of an 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.

The **language of an NFA** M is the set of all strings accepted by M, that is:

$$L(M) = \{w : M \text{ accepts } w\}$$

Non-Deterministic Finite Automata

The above idea can be mathematically described as follows:

Definition

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

- Σ 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
- $\delta : (Q \times \Sigma) \rightarrow 2^Q$ 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!

Extending δ For an NFA

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

$$\begin{aligned} \delta^* : (2^Q \times \Sigma^*) &\to 2^Q \\ (S, \epsilon) &\mapsto S \\ (S, aw) &\mapsto \delta^* \left(\bigcup_{q \in S} \delta(q, a), w \right) \end{aligned}$$

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

Definition

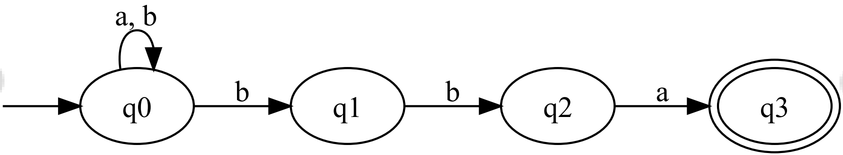
An NFA given by $M = (\Sigma, Q, q_0, A, \delta)$ **accepts a string** w if and only if $\delta^*(\{q_0\}, w) \cap A \neq \emptyset$

Simulating an NFA

Algorithm 3 NFA Recognition Algorithm

$w = a_1 a_2 \ldots a_n$
$S = \{q_0\}$
for i in 1 to n **do**
    $S = \bigcup_{q \in S} \delta(q, a_i)$
end for
if $S \cap A \neq \emptyset$ **then**
    Accept
else
    Reject
end if

Practice Simulating $w = abbba$



Processed	Remaining	S
ε	$abbba$	$\{q_0\}$
a	$bbba$	$\{q_0\}$
ab	bba	$\{q_0, q_1\}$
abb	ba	$\{q_0, q_1, q_2\}$
$abbb$	a	$\{q_0, q_1, q_2\}$
$abbba$	ε	$\{q_0, q_3\}$

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

NFA to DFA

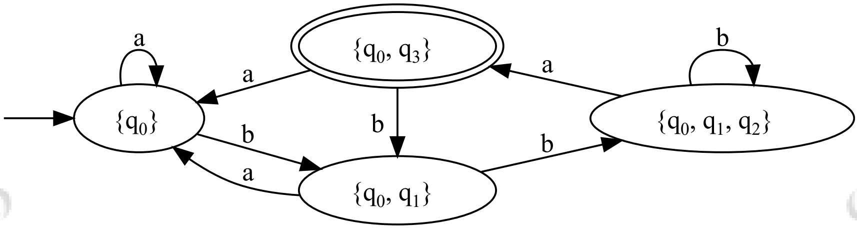
- NFAs are not more powerful than DFAs!
- Why not: Even the power-set of a set of states is still finite. So, we can represent *sets* of states in the NFA as *single* states in the DFA!

NFA to DFA

To convert an NFA to a DFA, one could write down all the 2^Q possible states and then connect them one by one based on δ and each letter in Σ. This however leads to a lot of extra states and a lot of unnecessary work. Instead,

- Start with the state S = {q0}
- From this state, go to the NFA and determine what happens on each a ∈ Σ for each q ∈ S. The set of resulting states should become its own state in your DFA.
- Repeat the previous step for each new state created until you have exhausted every possibility.
- Accepting states are any states that included an accepting state of the original NFA.

Previous NFA as a DFA



More Examples In Class

Let $\Sigma = \{a, b, c\}$. Write an NFA and the associated DFA for the following examples:

- $L = \{abc\} \cup \{w : w \text{ ends with } cc\}$
- $L = \{abc\} \cup \{w : w \text{ contains } cc\}$
- $L = \{w : w \text{ contains } cab\} \cup$
  $\{w : w \text{ contains an even number of } bs\}$
- $L = \{w : w \text{ contains exactly one } abb\} \cup$
  $\{w : w \text{ does not contain } ac\}$