Nondeterministic Finite Automata

Last class:

- Introduced finite automata
- Defined deterministic finite automata and their operation
- Began proving closure of finite operations – but found it too hard for concatenation & Kleene star!

Today:

- Introduce nondeterministic finite automata.
- Show they have the same power as DFAs.
- Assignment 1 released after class.

**Felix felicis**

We proved that the regular languages were closed under the union operation. Could we make that proof easier?

- What if you knew in advance whether the input word was in \( A_1 \) or in \( A_2 \)?
- What if you could just guess correctly every time?

This will also allow us to easily prove closure for concatenation.

**Determinism**

All of the FAs we have seen so far are deterministic finite automata (DFAs)

- Only one possible move from each state to another for a given input symbol.
- A move in each for every input symbol.

![Diagram](image.png)
Nondeterministic finite automata

Allow (deterministic) FA to have a choice of zero or more next states for each state–input pair.

Note: two “1” arrows from \(q_0\); can take either.
The present state does not determine the next state; there are multiple possible futures!

How does the NFA work?

Multiple alternative computations on the input.

When there’s more than one possible way to proceed, take all of them!

Imagine a parallel computer following each of the paths independently: Create a new thread at every point of nondeterminism it comes to.

What happens when parallel branches differ in their output?

One choice might end up at \(q_3\), and another may end up not at \(q_3\).

If any computational branch leads to an accept state, the machine accepts the input

Acceptance overrules rejection.

Only reject if every possible way to proceed for the input leads to rejection.

Example

Input: 010110

Begin at start state \(q_0\).

Read 0: Follow the loop back to \(q_0\).

Read 1: There are 2 arrows labeled 1 starting at \(q_0\); split into 2 paths to represent the 2 places machine could be: \(q_0\) and \(q_1\).

Read 0: Now each path proceeds independently; they represent different threads of computation.

The path at \(q_1\) goes back to \(q_0\).

There’s no place for the path at \(q_1\) to go (no arrow with 0), so remove that path.

Only the path at \(q_0\) is left.
Example

Input: 010110

Read 1: Branch into $q_0$, $q_1$
Read 0: Follow 0 arrows from $q_0$, $q_1$, to get to $q_0$, $q_1$, $q_2$
Read o: Follow o arrows from $q_0$, $q_1$, to get to $q_0$, $q_3$

Each path represents a different thread of the computation

The NFA accepts this string because at least one path ended up at an accepting state ($q_3$), i.e., $010110 \in L(B)$

By contrast, 01011 is not in $L(B)$ because paths end at $q_o$, $q_1$, $q_2$, all possibilities are reject states.

Problem

Design an NFA to accept strings over alphabet \{1, 2, 3\} such that the last symbol appears previously, without any intervening higher symbol, e.g.,

\[
\begin{align*}
\ldots & 11 \\
\ldots & 21112 \\
\ldots & 312123
\end{align*}
\]

Trick: Use the start state to mean “I guess I haven’t seen the symbol that matches the ending symbol yet”.

Three other states represent a guess that the matching symbol has been seen, and remembers that symbol.

Proving closure

Think about how to prove the closure properties from earlier.

Showed that if $A_1$ and $A_2$ are regular, so is $A_1 \cup A_2$

Given a machine $M_1$ recognizing $A_1$, and a machine $M_2$ recognizing $A_2$, we built a machine $M$ that recognizes $A_1 \cup A_2$ by simulating $A_1$ and $A_2$ in parallel
Prove closure under concatenation

If $A_1$ and $A_2$ are regular, then so is $A_1A_2$

Start off the same way:
- Suppose $M_1$ recognizes $A_1$ and $M_2$ recognizes $A_2$
- Construct $M$ recognizing $A_1A_2$
- What does $M$ need to do?

Given a string $w$, $M$ needs to determine if it is possible to cut $w$ into 2 pieces, the first of which is in $A_1$ and the second of which is in $A_2$

$$
\begin{array}{c|c|c}
 w & \in A_1 & \in A_2 \\
\end{array}
$$

There are many possible ways of cutting!

Why not feed $w$ into $M_1$ until we get to an accept state, and then transition control to $M_2$ by going to the start state of $M_2$?

Just because you found an initial piece of $W$ in $A_1$ does not necessarily mean you found the right place to cut $W$!

It is possible that the remainder is not in $A_2$, and you wrongly reject the string

Maybe we should wait until a later time to switch to $A_2$

Non-determinism provides an elegant solution to this problem

Formal NFA

$N = (Q, \Sigma, \delta, q_0, F)$ like a DFA, but:

$\delta(q, a)$ is a set of states, rather than a single state

Extension to $\delta$
- Basis: $\delta(q, \varepsilon) = \{q\}$
- Induction: Let:
  * $\delta(q, w) = \{p_1, p_2, \ldots, p_k\}$
  * $\delta(p_i, a) = S_j$ for $i = 1, 2, \ldots, k$
  * Then $\delta(q, wa) = S_j \cup S_{2j} \cup \ldots \cup S_{kj}$

Language of an NFA

An NFA accepts $w$ if any path from the start state to an accepting state is labeled $w$. Formally:

$L(N) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset \}$

Example

Here is a DFA that accepts a language $L$ consisting of all strings over $\Sigma(a,b)$ that begin with either $aa$ or $bb$
Suppose we want to make an automaton to recognize \( \text{Rev}(L) \), the language of all strings that end with \( aa \) or \( bb \).

**Easy solution:** Reverse all transitions and interchange the start and accept states:

**But this isn't a DFA!**

![Diagram](image)

More than one start state...

More than one transition labeled with the same symbol...

This is an NFA!

**NFAs and DFAs**

Because there is a degree of choice available in an NFA, is it more powerful than a DFA?

That is, can NFAs recognize languages a DFA cannot?

**NO!**

**Theorem**

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

**Equivalence of NFAs and DFAs**

*NFAs and DFAs recognize the same class of languages.*

A bit surprising: NFAs seem more powerful.

Useful: easier to specify an NFA for a language, convert to DFA

Two machines are equivalent if they recognize the same language
Proof idea

If a language is recognized by an NFA, show the existence of a DFA that recognizes it.

Proof by construction: Given an NFA, construct a DFA that recognizes the same language.

Intuitively, we can simulate the NFA by keeping track of all the states you can get to on a given input.

Construct $Q'$, the set of subsets of $Q$:

$Q = \{q_0, q_1, q_2\}$

$Q' = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

Proof

1. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing some language $A$.
2. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes $A$.
3. $Q' = \mathcal{P}(Q)$, the set of subsets of $Q$.
4. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$.
   - If $R$ is a state of $M$, it is also a set of states of $N$ (because of 3).
   - When $M$ is in state $R$ and reads a symbol $a$, it tracks where $a$ would go in $N$ from each state in $R$.
   - Because each state may go to a set of states, we take the union of all these sets. This can be written as: $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$
5. $q_0' = \{q_0\}$
   - $M$ starts in the state corresponding to the collection containing just the start state of $N$.
6. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$.
   - The machine $M$ accepts if one of the possible states that $N$ could be in at this point is an accept state.

Convert the following NFA to a DFA:

$Q' = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

For $R \in Q'$ and $a \in \Sigma$,
let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$

- $\delta'(\{q_0\}, a) = \{q_0\}$
- $\delta'(\{q_1\}, a) = \{q_1\}$
- $\delta'(\{q_2\}, a) = \{q_2\}$
- $\delta'(\{q_0, q_1\}, a) = \{q_0, q_1\}$
- $\delta'(\{q_0, q_2\}, a) = \{q_0, q_2\}$
- $\delta'(\{q_1, q_2\}, a) = \{q_1, q_2\}$
- $\delta'(\{q_0, q_1, q_2\}, a) = \{q_0, q_1, q_2\}$

$F' = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$
Lazy strategy

You don’t have to construct all the possible state sets at the outset.

Instead, construct state sets as they appear in the computation,

i.e., start with the start state set, construct the set for transitions on each input symbol, then construct the set for transitions from those sets, etc.
Example

The DFA

Problem

Recap
Then, the NFA

Nondeterminism

When machine is in a given state and reads a symbol, the machine will have a choice of where to move to next.

There may be states where, after reading a given symbol, the machine has nowhere to go.

Applying the transition function will give, not one state, but zero or more states.

Transition function
– $\delta$ is a function from $Q \times \Sigma$ to $2^Q$
– $\delta(q, a) =$ subset of $Q$ (possibly empty)

And now...

introducing...

the newest member of the FA family...

the nondeterministic finite automaton with $\epsilon$ transitions (NFA-\$\epsilon\$)

NFA with $\epsilon$-transitions

For both DFAs and NFAs, you must read a symbol in order for the machine to make a move.

In NFA with $\epsilon$ transitions, the machine can make a move without reading a symbol off the input tape.

Such a move is called a $\epsilon$-transition.

NFA with $\epsilon$-transitions

Allow $\epsilon$ to be a label on arcs

Nothing else changes: acceptance of $w$ is still the existence of a path from the start state to an accepting state with label $w$

But $\epsilon$ can appear on arcs, and means the empty string (i.e., no visible contribution to $w$)

When an arc labeled $\epsilon$ is traversed, no input is consumed
**Example**

Input: 001

0 \epsilon 0 1 \epsilon = 001

---

**Formal definition of NFA-ε**

A Nondeterministic Finite Automaton with \( \epsilon \)-transitions is a 5-tuple \((Q, \Sigma, q_0, \delta, F)\) where

- \(Q\) is a finite set of states
- \(\Sigma\) is a finite alphabet of symbols
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of accepting states
- \(\delta\) is a function from \(Q \times (\Sigma \cup \{\epsilon\})\) to \(\mathcal{P}(Q)\) (i.e., the transition function)

---

**DFAs and NFA-εs**

\(\epsilon\)-transitions are a convenience, but they do not increase the power of FAs.

*For any NFA-ε, there is an equivalent DFA, i.e., one that accepts the same language.*

The construction is similar to the NFA-to-DFA construction.

---

**Creating a DFA from an NFA-ε**

*or, eliminating \(\epsilon\)-transitions*

1. Compute the \(\epsilon\)-closure for each state, i.e., the set of states reachable from that state using only \(\epsilon\)-transitions.
2. The start state of the DFA is now \(\epsilon\)-CLOSE\((q_0)\).
3. Define \(\delta\) for each \(a \in \Sigma\) and each the \(\epsilon\)-CLOSED set \(S\):
   - If a state \(p \in S\) can reach state \(q\) on input \(a\) (not \(\epsilon\!\)),
     - then add a transition on input \(a\) from \(S\) to \(\epsilon\)-CLOSE\((q)\).
4. The set of final states for the DFA now includes those sets that contain at least one accepting state of the NFA-ε.
NFAs, NFAs with ε-transitions, and DFAs describe same class of languages. Thus to show a language is a regular language, you can just build a NFA that recognizes it, rather than a DFA.

Many times it is more convenient to build a NFA rather than a DFA, especially if you want to keep track of multiple possibilities.