Nondeterministic Finite Automata

Lecture 4
12 September 2019
Previously:

Defined how a strictly defined DFA can compute

Today:

Closure properties of regular languages
Simultaneously do and do not
Assignment 1
Office hours
Closure properties of regular languages
In addition to complement, the regular languages are closed under the \textit{regular operations}:

\textbf{Union (∪)}:

\[ A \cup B = \{w \mid w \in A \text{ or } w \in B\} \]

\textbf{Concatenation (⊙)}:

\[ A \circ B = AB = \{xy \mid x \in A, y \in B\} \]

\textbf{Kleene star (⋆)}:

\[ A^* = \{w \mid w = x_1x_2\ldots x_k, k \geq 0, x_i \in A\} \]
Proving closure under union (\( \cup \))
Show that if \( A \) and \( B \) are regular, then so is \( A \cup B \).

**Proof by construction:**

Given the automata

\[
M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \quad \text{recognizing} \ A
\]
\[
M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \quad \text{recognizing} \ B
\]

construct \( M = (Q, \Sigma, \delta, q_0, F) \) recognizing \( A \cup B \).

For simplicity, let the alphabets be the same.

...
You might think: Run the string through \( M_1 \), see whether \( M_1 \) accepts it, then run the string through \( M_2 \) and see whether \( M_2 \) accepts it.

But you can’t try something on the whole input string, and try another thing on the whole input string; you only get one pass!
**Approach:** Simulate $M_1$ and $M_2$ simultaneously as we read each input symbol.

Imagine putting two fingers on the diagrams of the automata for $M_1$ and $M_2$, and moving them around according to the input.

At the end, if either finger is on an accept state, then we accept.

Implement this strategy in $M$. 
Formalization

Keep track of a state in $M_1$ and a state in $M_2$ as a single state in $M$

Each state in $M$ corresponds to a pair of states, one in $M_1$ and one in $M_2$

Let $Q = Q_1 \times Q_2 = \{(q, r) \mid q \in Q_1, r \in Q_2\}$

How to define $\delta$?
Define $\delta$

When a new symbol comes in, go to wherever $q$ goes and wherever $r$ goes, individually

$$\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$$

Start state is $q_0 = (q_1, q_2)$

Accept set is $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Note $F_1 \times F_2$ would give the intersection, which isn’t what we want.

It’s clear by induction that the $k^{th}$ state of $M$ is just the $k^{th}$ state of $M_1$ and $k^{th}$ state of $M_2$. 
Proving closure under concatenation (\(\cdot\))
Recall: If $A = \{a, b\}$ and $B = \{b, c\}$, we can concatenate the languages to get $A \circ B = \{ab, ac, bb, bc\}$
Show that if $A$ and $B$ are regular, then so is $A \circ B$

**Proof by construction:**

Given the automata

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizing $A$

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizing $B$

construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A \circ B$.

...
How do we know where the first string ends and the second begins?

As with union, the solution involves keeping track of multiple possibilities.

But there isn’t a straightforward way to do this with a DFA; our model makes it too hard to keep track of the possibilities.

Instead, let’s consider a type of finite automaton that keeps track of multiple possibilities for us, much simplifying writing these proofs!
Nondeterministic finite automata
All of the computers we’ve seen so far are **deterministic finite automata** (DFAs)

A model of computation is **deterministic** if, at every point in the computation, there is exactly one move it can make.

A model of computation is **nondeterministic** if the machine can have multiple decisions it can make at one point.
Nondeterministic finite automata

Allow a finite automaton to have a choice of zero or more next states for each state-input pair.
NFAs are structurally similar to DFAs, but they represent a fundamental shift in how we’ll think about computation.

The present state does *not* determine the next state; there are multiple possible futures!

An NFA accepts if *any* series of choices leads to an accepting state.
A simple NFA

0, 1

start

\[ q_0 \rightarrow 1 \quad q_1 \rightarrow 1 \quad q_2 \]

\[ q_3 \]

0, 1

0, 1

0, 1

0, 1

0 1 0 1 1
A finite state machine (FSM) with the following states and transitions:

- **Start state**: $q_0$
- **Transitions**:
  - $q_0$ to $q_1$ on input $1$
  - $q_1$ to $q_2$ on input $1$
  - $q_3$ on transitions $0$, $1$ (self-loop)

The input sequence is $01011$. The machine transitions as follows:

- Start at $q_0$.
- Move to $q_1$ on $1$.
- Move to $q_2$ on $1$.
- Move back to $q_3$ on $0$, $1$.

The automaton accepts the input sequence.$\square$
A more complex NFA

If an NFA needs to make a transition when no transition exists, the automaton dies and that particular path does not accept.
The diagram shows a finite automaton with three states: start, $q_0$, $q_1$, and $q_2$. The transitions are labeled with input symbols 0 and 1. The transitions are as follows:

- From start to $q_0$: 0, 1
- From $q_0$ to $q_1$: 1
- From $q_1$ to $q_2$: 1
- From $q_2$ back to $q_0$: 0, 1

The input sequence shown at the bottom is 01011.
The diagram represents a finite automaton (FA) with the following states and transitions:

- **States**: $q_0$, $q_1$, $q_2$
- **Transitions**:
  - From $q_0$ on input $0$, $1$ move to $q_0$.
  - From $q_0$ on input $1$ move to $q_1$.
  - From $q_1$ on input $1$ move to $q_2$.
  - From $q_2$ on input $1$ move to $q_2$.

The automaton starts at state $q_0$ and accepts the string $01011$ as indicated by the highlighted string at the bottom of the image.
The diagram depicts a finite automaton with the following states:

- **Start state**: \( q_0 \)
- **States**: \( q_0, q_1, q_2 \)

Transitions:
- \( q_0 \) to \( q_1 \) on input \( 1 \)
- \( q_1 \) to \( q_2 \) on input \( 1 \)
- \( q_0 \) to \( q_0 \) on input \( 0, 1 \)

The automaton accepts the following input sequence:

\[
0 \ 1 \ 0 \ 1 \ 1
\]
Nowhere to go!
The diagram shows a deterministic finite automaton (DFA) with the following states and transitions:

- **Start State**: $q_0$
- **States**: $q_0$, $q_1$, $q_2$
- **Transitions**:
  - From $q_0$ to $q_1$ on input $1$
  - From $q_1$ to $q_2$ on input $1$
  - From $q_0$ to $q_0$ on input $0, 1$
- **Accepting State**: $q_2$

The input string $01011$ is processed through the DFA, starting from the initial state $q_0$. The automaton moves through the states as follows:

1. Start at $q_0$.
2. Input $0$: $q_0$ to $q_0$.
3. Input $1$: $q_0$ to $q_1$.
4. Input $0$: $q_1$ to $q_2$.
5. Input $1$: $q_2$.
6. Input $1$: $q_2$.

The automaton ends in the accepting state $q_2$.
The language of an NFA is
$L(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \}$

What’s the language of this NFA?

Assume $\Sigma = \{h, i\}$
The language of an NFA is
\[ L(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \} \]

What's the language of this NFA?

Assume \( \Sigma = \{0, 1\} \)
The language of an NFA is
\[ L(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \} \]

What’s the language of this NFA?

Assume \( \Sigma = \{ 0, 1 \} \)
For both DFAs and NFAs, you must read a symbol in order for the machine to make a move.

We can define a variant model, NFA-ε, that can move without consuming an input symbol – an \( \varepsilon \)-transition.
Example

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\varepsilon & 0 & \varepsilon & 1 \\
\varepsilon & 0 & \varepsilon & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
q & 1 & r & \varepsilon \\
1 & 0 & 0 & \varepsilon \\
\end{array}
\]

\[
\begin{array}{cccc}
r & \varepsilon & s & 1 \\
\varepsilon & 1 & 1 & \varepsilon \\
\end{array}
\]

\[
\begin{array}{cccc}
s & \varepsilon & 0 & 1 \\
\varepsilon & 1 & 0 & 1 \\
\end{array}
\]
An NFA-ε is not required to follow ε-transitions; they’re just another choice of path for the computation.

Since this is very convenient, when we talk about an NFA, we usually want it to be an NFA-ε.
Nondeterministic machines are a serious departure from physical computers.

There are two helpful ways to build an intuition for them:

- Perfect positive guessing
- Massive parallelism
Perfect positive guessing
Perfect positive guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: \[ a \ b \ a \ b \ a \ b \ a \]
Perfect positive guessing

\[
\begin{align*}
q_0 & \xrightarrow{a,b} q_0 \\
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]
Perfect positive guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Start state: \( q_0 \)

Input symbols: \( a, b \)

Sequence: \( a b a b a a \)
Perfect positive guessing

\[
\begin{align*}
q_0 &\rightarrow a \rightarrow q_1 \\
q_1 &\rightarrow b \rightarrow q_2 \\
q_2 &\rightarrow a \rightarrow q_3 \\
\end{align*}
\]
Perfect positive guessing

\[ \text{start} \quad q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Transition:
- \( q_0 \) to \( q_1 \): a, b
- \( q_1 \) to \( q_2 \): b
- \( q_2 \) to \( q_3 \): a

Input sequence: a b a b a a
Perfect positive guessing

\[
\begin{align*}
q_0 &\xrightarrow{a} q_1 \\
q_1 &\xrightarrow{b} q_2 \\
q_2 &\xrightarrow{a} q_3 \\
\end{align*}
\]
Perfect positive guessing

\[
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\]

\[\text{start} \quad a, b\]
Perfect positive guessing

\[ a, b \]

\[ \text{start} \]

\[ q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \]

\[ a \ b \ a \ b \ a \ b \ a \]
Perfect positive guessing

$\text{start} \xrightarrow{a, b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$

Illustration by Gemma Correll
NFAs are our “Liquid Luck” potion
Perfect positive guessing

We can think of nondeterministic machines as having *magic superpowers* that enable them to guess the correct choice of moves to make.

If there is at least one choice leading to an accepting state for the input, the machine will guess it.

If there are no choices, the machine guesses any one of the wrong answers.

There’s no physical analog for this style of computation.
Massive parallelism

![Diagram](attachment://diagram.png)

The diagram illustrates a transition system with states `q0`, `q1`, `q2`, and `q3`, labeled with inputs `a` and `b`. The diagram starts with `q0` and can transition to `q1` on `a`, then to `q2` on `b`, and finally to `q3` on `a`. The transition `q3` is a loop, indicating an infinite sequence of `a`.
Massive parallelism
Massive parallelism

- start
- $q_0 \xrightarrow{a,b} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_3$
- $a b a b a a$
Massive parallelism

```
q0 a b q1
  |   |   |
  a   b   a  q2
   |   |
   a   q3
```

Input: a b a b a a
Massive parallelism
Massive parallelism
Massive parallelism

```
a b a b a a
```
Massive parallelism

![Diagram showing a workflow with states q0, q1, q2, and q3, with transitions labeled a and b. The states are connected with arrows, and the input sequence is ababa.](image)
Massive parallelism

\[q_0, a, b, q_1, b, q_2, a, q_3\]
Massive parallelism
Massive parallelism

![Diagram](image-url)
Massive parallelism

\[ \text{start} \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

- \( q_0 \) is the start state.
- Transitions: \( a, b \) from \( q_0 \) to \( q_1 \), \( a \) from \( q_1 \) to \( q_2 \), \( b \) from \( q_2 \) to \( q_3 \).

Input sequence: \( ababaab \)
Massive parallelism
Massive parallelism

a, b

start

$q_0$ a $q_1$ b $q_2$ a $q_3$

a b a b a
Massive parallelism

\[
\begin{align*}
q_0 &\xrightarrow{a} q_1 \\
q_1 &\xrightarrow{b} q_2 \\
q_2 &\xrightarrow{a} q_3
\end{align*}
\]
Massive parallelism

\[ q_0, a, b, q_1, a, b, q_2, a, q_3 \]

Start state: \( q_0 \)

Inputs: a, b
Massive parallelism

We’re in at least one accepting state, so there’s some path that gets us to an accepting state.

Illustration by Gemma Correll
The future was and is massive parallelism.

Connection machine CM-1 schematic art by Tamiko Thiel, 1983
Massive parallelism

An NFA can also be thought of as a DFA that can be in many states at once.

Each symbol read causes a transition on every active state into each potential state that could be visited.

Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
Perfect guessing is a helpful way to think about how to design a machine to recognize a language.

Massive parallelism is a great way to test machines, and it has nice theoretical implications.
Nondeterministic machines may not be feasible, but nondeterminism is a convenient way to think about computation. This leads to the question, “Can any problem that can be solved by a nondeterministic machine be solved by a deterministic one?”

The answer will vary from automaton to automaton. We’ll explore this in more depth as we keep going.
Formal definition of a nondeterministic finite automaton (NFA)

\[ N = (Q, \Sigma, \delta, q_0, F) \text{ like a DFA, but:} \]

\[ \delta(q, a) \text{ is a set of states, rather than a single state} \]

Extension to \( \hat{\delta} \):

**Basis:** \( \hat{\delta}(q, \varepsilon) = \{q\} \)

**Induction:** Let

\[ \hat{\delta}(q, w) = \{p_1, p_2, \ldots, p_k\} \]

\[ \delta(p_i, a) = S_i \text{ for } i = 1, 2, \ldots, k \]

Then \( \hat{\delta}(q, wa) = S_1 \cup S_2 \cup \cdots \cup S_k \)

**Language of an NFA:** An NFA accepts \( w \) if *any* path from the start state to an accepting state is labeled \( w \). Formally:

\[ L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\} \]
Designing NFAs
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.,

\[\ldots11\]
\[\ldots21112\]
\[\ldots312123\]

*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.
Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser