Today's learning goals

- Distinguish between an NFA and a DFA
- Define nondeterminism and why it may be helpful
- Design NFA recognizing a given language
- Precisely specify general constructions for NFA recognizing languages output by operations

Sipser Ch 1.1, 1.2
General proof structure/strategy

**Theorem:** For any $L$ over $\Sigma$, if $L$ is regular then [the result of some operation on $L$] is also regular.

**Proof:**
Given name variables for sets, machines assumed to exist.
**WTS** state goal and outline plan.
Construction using objects previously defined + new tools working towards goal. Give formal definition and explain.
Correctness prove that construction works.
**Conclusion** recap what you've proved.
Regular languages

Over any fixed alphabet,

- Each finite language is regular
- There are infinite regular languages
- If a set is regular, its complement is also regular
- If two sets are regular, their union is also regular
- If two sets are regular, their intersection is also regular
The regular operations

For A, B languages over same alphabet, define:

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

\[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

\[ A^* = \{ x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \} \]

How can we prove that the concatenation of two regular languages is a regular language?
Construction

\[ A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \]

How would this work for \( A = \{ a^n \mid n \geq 0 \} \), \( B = \{ b^m \mid m \geq 0 \} \)?
Solution: nondeterminism

- Allow transitions that don't "consume" any input

- Instead of exactly one outgoing transition from each state labelled by each symbol
  - Can have 0 transitions
  - Can have > 1 transitions
Construction

$A \circ B = \{ xy \mid x \in A \text{ and } y \in B \}$

How would this work for $A = \{ a^n \mid n \geq 0 \}$, $B = \{ b^m \mid m \geq 0 \}$?
New – easier – construction for union

• "Guess" one of finite list of criteria to meet

Input

M1

Accept if either (or both) accepts

M2
Caveat

• This nondeterminism trick doesn't help us with closure arguments.

Why?

• To prove that the class of regular languages is closed under OP, needs that OP(L) is recognized by a DFA
Caveat

• This nondeterminism trick doesn't help us with closure arguments **YET!**

Why?

• We'll prove that any language recognized by some NFA is also recognized by some (different) DFA.
Differences between NFA and DFA

- **DFA**: unique computation path for each input
- **NFA**: allow several (or zero) alternative computations on same input
  - $\delta(q,x)$ may specify *more than one* possible next states
  - $\epsilon$ transitions allow the machine to transition between states spontaneously, without consuming any input symbols
Formal definition of NFA

A nondeterministic finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where

1. \(Q\) is a finite set called the states
2. \(\Sigma\) is a finite set called the alphabet
3. \(\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)\) is the transition function
4. \(q_0 \in Q\) is the start state
5. \(F \subseteq Q\) is the set of accept states.

Which piece of the definition of NFA means there might be more than one possible next state from a given state, when reading symbol \(x\) from the alphabet?

A. Line 2, the size of \(\Sigma\)
B. Line 3, the domain of \(\delta\)
C. Line 3, the codomain of \(\delta\)
D. Line 5, that \(F\) is a set
E. I don't know.
Tracing NFA execution

- Is 0 accepted?
- Is 1 accepted?
- Is 0101 accepted?
- Is 110 accepted?
- Is the empty string accepted?
Acceptance in an NFA

An NFA \((Q, \Sigma, \delta, q_0, F)\) accepts a string \(w\) in \(\Sigma^*\) iff we can write \(w = y_1 y_2 \cdots y_m\) where each \(y_i \in \Sigma_\varepsilon\) and there is a sequence of states \(r_0, \ldots, r_m \in Q\) such that

1. \(r_0 = q_0\)
2. \(r_{i+1} \in \delta(r_i, y_{i+1})\) for each \(i = 0, \ldots, m - 1\)
3. \(r_m \in F\).
…structure i was using to display the output was a TreeCtrl, so I figured some kind of weird tree traversal system would work and i made it into this implicit state machine that used a stack to traverse up and down into different levels. In short, it was an absolute monster of a structure (which i am immensely proud of for continuing to work as new requirements were added every couple of hours :p). Needless to say, all that code is now gone. Tuesday I came in on a mission to reimplement the tree without any interruption in the continual updating of my project. This was made possible by mentor deciding to just disappear into thin air for the day. I designed an NFA (thanks be Shacham) that could cover every case and also allow the timeline to be completely extensible. Note, I needed an NFA because there are a few input strings that require up to 5 transitions (the tree gets really deep) and the states couldn't really be just jumped to because of the way the TreeCtrl works. After lunch, I implemented this NFA (as it turns out, there was a slight amount of copy and paste) and had it working and fully functional within an hour. On Wednesday, I arranged a short meeting with my mentor to show it off and discuss some requirements for a side project I was to begin working on. When my mentor showed up to the meeting, he brought with him a Qualstar for me! Qualstars are internal awards given to employees who exceed expectations and provide excellent work (or something like that) and I got one for completing my first project so quickly and saving engineer time. Needless to say, my mentor was not impressed with my NFA and gave me some more work to do...

I FINALLY USED SOMETHING I LEARNED AT UCSD WHICH I NEVER THOUGHT I WOULD USE!!!!! i used CSE 105 to design my NFA and knew that an NFA would be a good way to solve this problem only because of that class. Also, that Qualstar i got kinda feels like a real world A+, except it actually means something :p

Chris Miranda (CSE 197)
More differences between NFA and DFA

- **DFA**: unique computation path for each input
- **NFA**: allow several (or zero) alternative computations on same input
  - $\delta(q,x)$ may specify more than one possible next states
  - $\varepsilon$ transitions allow the machine to transition between states spontaneously, without consuming any input symbols

Types of components of formal definition

- **DFA** $\delta : Q \times \Sigma \rightarrow Q$
- **NFA** $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$
Similarities between DFA and NFA

• If L is a language recognized by a DFA, is there some NFA that recognizes it?

A. Yes
B. No
C. Depends on L
D. I don't know.
Next steps

• Defining NFA in closure constructions formally.
• Showing that NFA and DFA are equally expressive
Concatenation

"Guess" some stage of input at which switch modes

Given \( M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \), \( M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \) build

\[ N = (Q_1 \cup Q_2, \Sigma, \delta, q_1, F_2) \] with \( \delta \ldots \)
Concatenation

\[ \delta(q, x) = \]

if \( q \) is in \( Q_1 \), \( x \) is in \( \Sigma \)

if \( q \) is in \( Q_2 \), \( x \) is in \( \Sigma \)
\[ \delta(q, x) = \begin{cases} \delta_1(q, x) & \text{if } q \text{ is in } Q_1, \ x \text{ is in } \Sigma \\ \delta_2(q, x) & \text{if } q \text{ is in } Q_2, \ x \text{ is in } \Sigma \end{cases} \]
Concatenation

\[ \delta(q, x) = \begin{cases} \delta_1(q, x) & \text{if } q \text{ is in } Q_1, x \text{ is in } \Sigma \\ \delta_2(q, x) & \text{if } q \text{ is in } Q_2, x \text{ is in } \Sigma \\ \{q_2\} & \text{if } q \text{ is in } F_1, x = \varepsilon \\ \emptyset & \text{otherwise} \end{cases} \]

Correctness proof in the book (page 61)
Star operation

Given $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, build

$$N = (Q_1 \cup \{q_0\}, \Sigma, \delta, q_0, F_1 \cup \{q_0\})$$

and $\delta(q, x) = \ldots$

*Construction in the book (page 63)*
Next steps

• Showing that NFA and DFA are equally expressive.