Recap

A language is a **regular language** iff some finite automaton recognizes it.

What languages aren't regular?
- Languages that require you to remember more than a finite number of possibilities.
- Finite automata can't count.
- Finite automata can't (generally) remember exactly what's been read.

$L = \{ww\}$ isn't regular
$L = \{0^n1^n\}$ isn't regular.
Imagine a string from here to the moon. You’re trying to recognize it, but you only have a fixed number of states, e.g., 100.

The regular languages are closed under the regular operations.

- We proved this for union by keeping track of the possible states of the machines for the languages we were taking the union of.
- But proving it for concatenation look impossible with DFAs. How could we know when we’d seen all of the string from language 1 and were now looking at the string from language 2?

We can recognize languages where we only need to keep track of a fixed set of possibilities, e.g.,

- All even-length strings
- Any number of $a$s followed by any number of $b$s
- Alternating $a$s and $b$s
- Strings ending with er, or, or ist

Proving closure under concatenation is easy… if we have nondeterminism.
Two roads diverged in a wood, and I –
both of them, at the same time, like a boss
I took the one less traveled by,
And that has made all the difference.
Robert Frost

_Determinism:_ Computation proceeds according to the design of the transition function. No choices, no randomness, no magic.

_Nondeterminism:_ Zero or more options to continue the computation. Accept if any of them would succeed.

Relating DFAs and NFAs

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!**
Equivalence of NFAs and DFAs

NFAs and DFAs recognize the same class of languages.

A bit surprising: NFAs seem more powerful.

This is useful: It’s often easier to specify an NFA for a language, and then convert to DFA.

Two machines are equivalent if they recognize the same language.

Theorem

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Intuition

NFAs and DFAs can only have finitely many states.

An NFA can be in any combination of its states, but there are only a finite number of possible combinations.

So, we could build a DFA where each state of the DFA corresponds to a set of states in the NFA.

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.
Construct \( Q \)'s, the set of subsets of \( Q \):

\[
Q = \{ q_0, q_1, q_2 \}
\]

\[
Q' = \{ \emptyset, \{ q_0 \}, \{ q_1 \}, \{ q_2 \}, \{ q_0, q_1 \}, \{ q_0, q_2 \}, \{ q_1, q_2 \}, \{ q_0, q_1, q_2 \} \}
\]

Convert the following NFA to a DFA

\[
\begin{aligned}
\text{start} & \quad \text{a, b} \\
q_0 & \quad b \quad q_1 \quad b \quad q_2
\end{aligned}
\]

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

\[
\begin{array}{|c|c|c|}
\hline
a & b & q_0' = \{ q_0 \} \\
\hline
\{ q_0 \} & \{ q_0 \} & \{ q_0, q_1 \} \\
\{ q_1 \} & \emptyset & \{ q_2 \} \\
\{ q_2 \} & \emptyset & \emptyset \\
\{ q_0, q_1 \} & \{ q_0 \} & \{ q_0, q_1, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0 \} & \{ q_0, q_1 \} \\
\{ q_1, q_2 \} & \emptyset & \{ q_2 \} \\
\{ q_0, q_1, q_2 \} & \{ q_0 \} & \{ q_0, q_1, q_2 \} \\
\emptyset & \emptyset & \emptyset \\
\hline
\end{array}
\]

\[
\begin{aligned}
\text{start} & \quad a, b \\
\{ q_0 \} & \quad a \quad b \\
\{ q_1 \} & \quad a \quad b \\
\{ q_0, q_1 \} & \quad a \quad b \\
\{ q_0, q_2 \} & \quad a \quad b \\
\{ q_1, q_2 \} & \quad a \quad b \\
\emptyset & \quad a, b \\
\end{aligned}
\]

Useless – unreachable – states
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**

Start state $q_0 = \{1\}$

- $\delta(1, a) = \{2, 3\}$
- $\delta(1, b) = \{4\}$
- $\delta(2,3), a = \delta((2), a) \cup \delta((3), a)$
  - $= \emptyset \cup (7,9)$
  - $= (7,9)$
- $\delta(2,3), b = \delta((2), b) \cup \delta((3), b)$
  - $= (5,6) \cup (8)$
  - $= (5,6,8)$
- $\delta(4), a = \emptyset$
- $\delta(4), b = \{10\}$

Final states $F' = \{(5,6,8), \{10\}\}$

The DFA

- $\delta(1), a = (2,3)$
- $\delta(1), b = \{4\}$
- $\delta(2,3), a = (7,9)$
- $\delta(2,3), b = (5,6,8)$
- $\delta(4), a = \emptyset$
- $\delta(4), b = \{10\}$
- $\delta(7,9), a = \emptyset$
- $\delta(7,9), b = \emptyset$
- $\delta(5,6,8), a = \emptyset$
- $\delta(5,6,8), b = \emptyset$
- $\delta(10), a = \emptyset$
- $\delta(10), b = \emptyset$
- $\delta(\emptyset, a) = \emptyset$
- $\delta(\emptyset, b) = \emptyset$

Start state $q_0 = \{1\}$

Final states $F' = \{(5,6,8), \{10\}\}$
Problem  Try converting this NFA to a DFA:

\[
\begin{array}{c}
\text{start} \\
q_0 \quad a, b \\
q_1 \quad a, a, b \\
q_2 \quad a, b \\
q_3 \quad a, a \\
\end{array}
\]

In the worst case, the number of states of the DFA constructed to simulate an NFA can be exponentially larger than the number of states of the NFA.

Proof

Let \( N = (Q, \Sigma, \delta, q_0, F) \) be an NFA recognizing some language \( A \).

We construct a DFA \( M = (Q', \Sigma, \delta', q_0', F') \) that recognizes \( A \):

\[
Q' = P(Q)
\]

If the NFA has \( k \) states, the DFA will have \( 2^k \) states.

\( M \) starts in the state corresponding to the collection containing just the start state of \( N \):

\[
q_0' = \{q_0\}
\]

The machine \( M \) accepts if one of the possible states that \( N \) could be in at this point is an accept state:

\[
F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}
\]

For \( R \in Q' \) and \( a \in \Sigma \),

\[
\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)
\]

Every state \( R \) of \( M \) is a set of states of \( N \). When \( M \) is in state \( R \) and reads a symbol \( a \), it tracks where \( a \) would go in \( N \) from each state in \( R \).
First there was the DFA.
For every state and every alphabet symbol, there is exactly one move that the machine can make:
\[ \delta: Q \times \Sigma \rightarrow Q \]
\( \delta \) is a total function, i.e., it is defined for all \( q \in Q \) and \( a \in \Sigma \)

Then there was the NFA.
For every state and every alphabet symbol, there are zero or more moves that the machine can make:
\[ \delta: Q \times \Sigma \rightarrow \mathcal{P}(Q) \]
\( \delta(q, a) \subseteq Q \)

And now, the newest member of the finite automaton family...
**NFA-ε:** *nondeterministic finite automaton with ε-transitions*

“Now with even less determinism!”

For both DFAs and NFAs, you must read a symbol in order for the machine to make a move.
An NFA-ε can move without consuming an input symbol – an *ε-transition.*
Note: NFAs are not required to follow $\varepsilon$-transitions; they’re just another choice of path for the computation.

### Designing an NFA-$\varepsilon$

Embrace the nondeterminism.

A good approach is guess-and-check:
- Is there some information you’d like to have?
- Have the machine nondeterministically guess that information, then have it deterministically check that the guess was right, i.e., filter out the bad guesses.

### Guess-and-check

$L = \{w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w\}$
Guess-and-check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]

NFA?

Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser
Jeffrey Ullman