highlights

Lemma: The language recognized by a DFA is the set of strings $x$ that label some path from its start state to one of its final states.

non-deterministic finite automaton (NFA)

• Graph with start state, final states, edges labeled by symbols (like DFA) but
  — Not required to have exactly 1 edge out of each state labeled by each symbol --- can have 0 or >1
  — Also can have edges labeled by empty string $\varepsilon$

• Definition: $x$ is in the language recognized by an NFA if and only if $x$ labels a path from the start state to some final state.

building an NFA

binary strings that have
- an even # of 1's
- or contain the substring 111 or 1000

NFAs and regular expressions

Theorem: For any set of strings (language) $A$ described by a regular expression, there is an NFA that recognizes $A$.

Proof idea: Structural induction based on the recursive definition of regular expressions...
build an NFA for \((01 \cup 1)^*0\)

\[(01 \cup 1)^*0\]

Every DFA is an NFA

- DFAs have requirements that NFAs don’t have

Can NFAs recognize more languages?

Theorem: For every NFA there is a DFA that recognizes exactly the same language.

Proof Idea:

- The DFA keeps track of ALL the states that the part of the input string read so far can reach in the NFA
- There will be one state in the DFA for each subset of states of the NFA that can be reached by some string

New start state for DFA

- The set of all states reachable from the start state of the NFA using only edges labeled ε

Conversion of NFAs to DFAs
For each state of the DFA corresponding to a set $S$ of states of the NFA and each symbol $s$:
- Add an edge labeled $s$ to state corresponding to $T$, the set of states of the NFA reached by starting from some state in $S$, then following one edge labeled by $s$, and then following some number of edges labeled by $\varepsilon$.
- $T$ will be $\emptyset$ if no edges from $S$ labeled $s$ exist.

Final states for the DFA:
- All states whose set contain some final state of the NFA.

Example: NFA to DFA
exponential blow-up in simulating nondeterminism

- In general the DFA might need a state for every subset of states of the NFA
  - Power set of the set of states of the NFA
  - $n$-state NFA yields DFA with at most $2^n$ states
  - We saw an example where roughly $2^n$ is necessary
    Is the $n^{th}$ char from the end a 1?

- The famous “$P=NP$?” question asks whether a similar blow-up is always necessary to get rid of nondeterminism for polynomial-time algorithms
DFAs = regular expressions

We have shown how to build an optimal DFA for every regular expression
- Build NFA
- Convert NFA to DFA using subset construction
- Minimize resulting DFA

**Theorem:** A language is recognized by a DFA if and only if it has a regular expression.

We show the other direction of the proof at the end of these lecture slides.

languages and machines!

DFAs recognize any finite language

Warmup: All finite languages are regular.

Exercise: Hard code it into the DFA.
Warmup 2: Surprising example here

languages and machines!

Generalized NFAs

- Like NFAs but allow
  - Parallel edges
  - Regular Expressions as edge labels
  - NFAs already have edges labeled ε or a
- An edge labeled by A can be followed by reading a string of input chars that is in the language represented by A
- A string x is accepted iff there is a path from start to final state labeled by a regular expression whose language contains x

languages and machines!

DFA = regular expressions

Theorem: A language is recognized by a DFA if and only if it has a regular expression

Proof: Last class: RegExp → NFA → DFA

Now: NFA → RegExp

(Enough to show this since every DFA is also an NFA.)

starting from an NFA

Add new start state and final state

Then eliminate original states one by one, keeping the same language, until it looks like:

Final regular expression will be A

L = \{ x ∈ \{0, 1\}^* : x has an equal number of substrings 01 and 10 \}.

L is infinite.

L is regular.

an interesting regular language

Theorem: A language is recognized by a DFA if and only if it has a regular expression

Proof: Last class: RegExp → NFA → DFA

Now: NFA → RegExp

(Enough to show this since every DFA is also an NFA.)

starting from an NFA

Add new start state and final state

Then eliminate original states one by one, keeping the same language, until it looks like:

Final regular expression will be A

languages and machines!
only two simplification rules

- **Rule 1:** For any two states $q_1$ and $q_2$ with parallel edges (possibly $q_1 = q_2$), replace

  $$q_1 \xrightarrow{A B} q_2$$

  by

  $$q_1 \xrightarrow{A \cup B} q_2$$

- **Rule 2:** Eliminate non-start/final state $q_3$ by replacing all

  $$q_1 \xrightarrow{A B C} q_2$$

  for every pair of states $q_1, q_2$ (even if $q_1 = q_2$)

converting an NFA to a regular expression

Consider the DFA for the mod 3 sum

- Accept strings from $\{0,1,2\}^*$ where the digits mod 3 sum of the
digits is 0

finite automaton without $t_1$

Label edges with regular expressions

$$
\begin{align*}
t_0 \rightarrow t_1 \rightarrow t_2 : & 10^*2 \\
t_0 \rightarrow t_1 \rightarrow t_2 : & 10^*1 \\
t_2 \rightarrow t_1 \rightarrow t_2 : & 20^*2 \\
t_2 \rightarrow t_1 \rightarrow t_2 : & 20^*1
\end{align*}
$$

Final regular expression:

$$(0 \cup 10^*2 \cup (2 \cup 10^*)((0 \cup 20^*)((1 \cup 20^*))^*)$$