Fall 2015

Lecture 24: DFAs, NFAs, and regular expressions
• FSMs with output at states
• State minimization
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.
nondeterministic 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
**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?
Every DFA is an NFA
– DFAs have requirements that NFAs don’t have

Can NFAs recognize more languages? No!

**Theorem:** For every NFA there is a DFA that recognizes exactly the same language.
Conversion of NFAs to DFAs

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 $\varepsilon$

Conversion of NFAs to a DFAs
For each state of the DFA corresponding to a set $S$ of states of the NFA and each letter $a$

- Add an edge labeled $a$ 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 $a$, and
  then following some number of edges labeled by $\varepsilon$

- $T$ will be $\emptyset$ if no edges from $S$ labeled $a$ exist
Final states for the DFA

- All states whose set contain some final state of the NFA
example: NFA to DFA
example: NFA to DFA
example: NFA to DFA
example: NFA to DFA
example: NFA to DFA

NFA

DFA
example: NFA to DFA
example: NFA to DFA
example: NFA to DFA
exponential blow-up in simulating mondeterminism

• 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
1 in third position from end

with NFA size $n$. 

```
A -- 1 -- B --> 0,1 -- C --> 0,1 -- D
```

```
\begin{itemize}
  \item A, D
  \item A, B
  \item A, B, D
  \item A, B, C
  \item A, C, D
  \item A, B, C, D
\end{itemize}
```
1 in third position from end
1 in third position from end
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!
languages and machines!

Warmup: All finite languages are regular.

{001, 10, 12}
DFAs recognize any finite language

Exercise: Hard code it into the NFA.

\[ L = \{ w_1, w_2, \ldots, w_k \} \]

\[ w_1 \cup w_2 \cup \cdots \cup w_k \]
languages and machines!

Warmup 2: Surprising example here

0* DFA NFA Regex
Main Event: Prove there is a context-free language that isn’t regular.
DFAs $\equiv$ regular expressions

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

**Proof:** We already saw: RegExp $\rightarrow$ NFA $\rightarrow$ DFA

**Now:** NFA $\rightarrow$ RegExp

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

• Like NFAs but allow
  – Parallel edges
  – Regular Expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ 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 $x$
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
only two simplification rules

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

  \[ q_1 \xrightarrow{A} q_2 \quad \text{by} \quad q_1 \xrightarrow{A \cup B} q_2 \]

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

  \[ q_1 \xrightarrow{A} q_3 \xrightarrow{B} q_2 \quad \text{by} \quad q_1 \xrightarrow{A \cup B} q_2 \]

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

\[ \text{DFA Diagram} \]
Label edges with regular expressions

\[
\begin{align*}
&t_0 \rightarrow t_1 \rightarrow t_0 : 10*2 \\
&t_0 \rightarrow t_1 \rightarrow t_2 : 10*1 \\
&t_2 \rightarrow t_1 \rightarrow t_0 : 20*2 \\
&t_2 \rightarrow t_1 \rightarrow t_2 : 20*1
\end{align*}
\]
finite automaton without $t_1$

$R_1$: $0 \cup 10^*2$
$R_2$: $2 \cup 10^*1$
$R_3$: $1 \cup 20^*2$
$R_4$: $0 \cup 20^*1$

$R_5$: $R_1 \cup R_2 R_4^* R_3$

Final regular expression:

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