Lecture 25: NFAs and their relation to REs & DFAs
Recall: DFAs

• States
• Transitions on input symbols
• Start state and final states
• The “language recognized” by the machine is the set of strings that reach a final state from the start

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₀</td>
<td>s₀</td>
<td>s₁</td>
</tr>
<tr>
<td>s₁</td>
<td>s₀</td>
<td>s₂</td>
</tr>
<tr>
<td>s₂</td>
<td>s₀</td>
<td>s₃</td>
</tr>
<tr>
<td>s₃</td>
<td>s₃</td>
<td>s₃</td>
</tr>
</tbody>
</table>
Recall: DFAs

• Each machine designed for strings over some fixed alphabet $\Sigma$.

• Must have a transition defined from each state for every symbol in $\Sigma$.

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Last Time: Nondeterministic Finite Automata (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 some valid execution of the machine gets to an accept state
Three ways of thinking about NFAs

- **Perfect guesser:** The NFA has input $x$ and whenever there is a choice of what to do it magically guesses a good one (if one exists)

- **Outside observer:** Is there a path labeled by $x$ from the start state to some accepting state?

- **Parallel exploration:** The NFA computation runs all possible computations on $x$ step-by-step at the same time in parallel
Path Labels

**Def:** The label of path $v_0, v_1, ..., v_n$ is the concatenation of the labels of the edges $(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n)$.

**Example:** The label of path $s_0, s_1, s_2, s_0, s_0$ is 1100.
Deterministic Finite Automata (DFA)

• **Def:** $x$ is in the language recognized by an DFA if and only if $x$ labels a path from the start state to some final state

![DFA Diagram]

• Path $v_0, v_1, ..., v_n$ with $v_0 = s_0$ and label $x$ describes a correct simulation of the DFA on input $x$
  - i-th step must match the i-th character of $x$
Nondeterministic Finite Automata (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 some path from the start state to an accepting state
Three ways of thinking about NFAs

- Perfect guesser: The NFA has input $x$ and whenever there is a choice of what to do it magically guesses a good one (if one exists)

- Outside observer: Is there a path labeled by $x$ from the start state to some accepting state?

- Parallel exploration: The NFA computation runs all possible computations on $x$ step-by-step at the same time in parallel
Compare with the smallest DFA
Parallel Exploration view of an NFA

Input string 0101100

0101100
Summary of NFAs

• Generalization of DFAs
  – drop two restrictions of DFAs
  – every DFA is an NFA

• Seem to be more powerful
  – designing is easier than with DFAs

• Seem related to regular expressions
The story so far...

\[
\begin{align*}
\text{REs} & \subseteq \text{CFGs} \\
\text{DFAs} & \subseteq \text{NFAs}
\end{align*}
\]
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...
Regular Expressions over $\Sigma$

- **Basis:**
  - $\varepsilon$ is a regular expression
  - $a$ is a regular expression for any $a \in \Sigma$

- **Recursive step:**
  - If $A$ and $B$ are regular expressions then so are:
    - $A \cup B$
    - $AB$
    - $A^*$
Base Case

- Case ε:

- Case a:
Base Case

- Case $\varepsilon$

- Case $a$
Base Case

- Case $\varepsilon$:

- Case $a$: 

\[ a \]
Inductive Hypothesis

• Suppose that for some regular expressions $A$ and $B$ there exist NFAs $N_A$ and $N_B$ such that $N_A$ recognizes the language given by $A$ and $N_B$ recognizes the language given by $B$
Inductive Step

Case $A \cup B$:
Inductive Step

Case $A \cup B$:

\[ N_A \]

\[ N_B \]
Inductive Step

Case AB:

\[ N_A \quad N_B \]
Inductive Step

Case AB:
Inductive Step

Case A*
Inductive Step

Case A*
Build an NFA for \((01 \cup 1)^* 0\)
Solution

\((01 \cup 1)^*0\)
The story so far...

![Diagram showing the relationship between REs, DFAs, CFGs, and NFAs]

- REs ⊆ CFGs
- DFAs ⊆ NFAs
NFAs and DFAs

Every DFA is an NFA
  – DFAs have requirements that NFAs don’t have

Can NFAs recognize more languages?
NFAs and DFAs

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
Three ways of thinking about NFAs

• Outside observer: Is there a path labeled by \( x \) from the start state to some final state?

• Perfect guesser: The NFA has input \( x \) and whenever there is a choice of what to do it magically guesses a good one (if one exists)

• Parallel exploration: The NFA computation runs all possible computations on \( x \) step-by-step at the same time in parallel
Parallel Exploration view of an NFA

Input string 0101100
Conversion of NFAs to a DFAs

• Construction Idea:
  – The DFA keeps track of ALL states reachable in the NFA along a path labeled by the input so far
    (Note: not all paths; all last states on those paths.)

  – There will be one state in the DFA for each \textit{subset} of states of the NFA that can be reached by some string
Conversion of NFAs to a DFAs

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 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
Conversion of NFAs to a DFAs

Final states for the DFA

– All states whose set contain some final state of the NFA

NFA

DFA
Example: NFA to DFA
Example: NFA to DFA

NFA

DFA
Example: NFA to DFA
Example: NFA to DFA

NFA

DFA
Example: NFA to DFA
Example: NFA to DFA
Example: NFA to DFA
Example: NFA to DFA

NFA

DFA
The story so far...

\[
\begin{align*}
\text{REs} \subseteq \text{CFGs} \\
\text{DFAs} = \text{NFAs}
\end{align*}
\]
Regular expressions $\subseteq$ NFAs $\equiv$ DFAs

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

Thus, we could now implement a RegExp library

– most RegExp libraries actually simulate the NFA
– (even better: one can combine the two approaches: apply DFA minimization lazily while simulating the NFA)
The story so far...

\[ \text{REs} \subseteq \text{CFGs} \]

\[ \text{DFAs} \cong \text{NFAs} \]

Is this \( \subseteq \) really “=” or “\( \not\subseteq \)”?
Regular expressions ≡ NFAs ≡ DFAs

**Theorem:** For any NFA, there is a regular expression that accepts the same language.

**Corollary:** A language is recognized by a DFA (or NFA) if and only if it has a regular expression.

You need to know these facts

- the construction for the Theorem is included in the slides after this, but you will not be tested on it.
The story so far...

\[ \text{REs} \subseteq \text{CFGs} \]

\[ \text{DFAs} = \text{NFAs} \]
The story so far...

Next time: Is this $\subseteq$ really “=” or “$\subsetneq$”?