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 state.

<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

![Diagram of NFA]
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.

- **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 $x$. 

![DFA Diagram](image-url)
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
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...

REs $\subseteq$ CFGs

DFAs $\subseteq$ NFAs
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:**
  - $\epsilon$ 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 $\varepsilon$:

- Case $a$:
Base Case

• Case ɛ:

• Case a:
Base Case

- Case $\varepsilon$:

- Case $a$:
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 $A \cup B$:
Inductive Step

Case AB:
Inductive Step

Case AB:
Inductive Step

Case A*

\[ N_A \]
Inductive Step

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

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

\[
\begin{array}{c}
\text{REs} \subseteq \text{CFGs} \\
\cap \\
\text{DFAs} \subseteq \text{NFAs}
\end{array}
\]
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 *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
Example: NFA to DFA

NFA

DFA
Example: NFA to DFA

NFA

DFA

\[ a, b \]
Example: NFA to DFA

NFA

DFA
Example: NFA to DFA
Example: NFA to DFA
Example: NFA to DFA
Example: NFA to DFA
Example: NFA to DFA
The story so far...

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

\[ \text{DFAs} = \text{NFAs} \]
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...

- REs ⊆ CFGs
- DFAs ⊆ NFAs

Is this ⊆ really “=” or “⊄”? 
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 sketched below but you will not be tested on it
New Machinery: Generalized NFAs

• Like NFAs but allow
  – parallel edges (between the same pair of states)
  – regular expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ or $a$

• Machine can follow an edge labeled by $A$ by reading a string of input characters in the language of $A$
  – (if $A$ is $a$ or $\varepsilon$, this matches the original definition, but we now allow REs built with recursive steps.)
New Machinery: Generalized NFAs

• Like NFAs but allow
  – parallel edges
  – regular expressions as edge labels
    NFAs already have edges labeled $\varepsilon$ or $a$

• The label of a path is now the concatenation of the regular expressions on those edges, making it a regular expression

• Def: A string $x$ is accepted by a generalized NFA iff there is a path from start to final state labeled by a regular expression whose language contains $x$
Construction Idea

Add new start state and final state

Then delete the original states one by one, adding edges to keep the same language, until the graph looks like:

![Diagram](image-url)
Starting from an NFA

Then delete the original states one by one, adding edges to keep the same language, until the graph looks like:

Final graph has only one path to the accepting state, which is labeled by $A$, so it accepts iff $x$ is in the language of $A$

Thus, $A$ is a regular expression with the same language as the original NFA.
Only two simplification rules

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

If the machine would have used the edge labeled $A$ by consuming an input $x$ in the language of $A$, it can instead use the edge labeled $A\cup B$.

Furthermore, this new edge does not allow transitions for any strings other than those that matched $A$ or $B$. 
Only two simplification rules

- **Rule 2**: Eliminate non-start/accepting state $q_3$ by creating direct edges that skip $q_3$

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

Any path from $q_1$ to $q_2$ would have to match $AB^nC$ for some $n$ (the number of times the self loop was used), so the machine can use the new edge instead. New edge *only* allows strings that were allowed before.
Construction Overview

Add new start state and final state

While the box contains some state s:
for all states r, t with (r, s) and (s, t) in E:
create a direct edge (r, t) by Rule 2
delete s (no longer needed)
merge all parallel edges by Rule 1
Construction Overview

While the box contains some state $s$: for all states $r, t$ with $(r, s)$ and $(s, t)$ in $E$: create a direct edge $(r, t)$ by Rule 2 delete $s$ (no longer needed) merge all parallel edges by Rule 1

When the loop exits, the graph looks like this:

A is a regular expression with the same language as the original NFA.
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

![DFA Diagram]
Splicing out a state $t_1$

Create direct edges between neighbors of $t_1$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

$t_0 \rightarrow t_1 \rightarrow t_0 : 10^*2$
$t_0 \rightarrow t_1 \rightarrow t_2 : 10^*1$
$t_2 \rightarrow t_1 \rightarrow t_0 : 20^*2$
$t_2 \rightarrow t_1 \rightarrow t_2 : 20^*1$
Splicing out a state $t_1$

Delete $t_1$ now that it is redundant

$t_0 \rightarrow t_1 \rightarrow t_0 : 10^*2$
$t_0 \rightarrow t_1 \rightarrow t_2 : 10^*1$
$t_2 \rightarrow t_1 \rightarrow t_0 : 20^*2$
$t_2 \rightarrow t_1 \rightarrow t_2 : 20^*1$
Splicing out a state $t_1$

Create direct edges between neighbors of $t_2$ (so that we can delete it afterward)
Splicing out a state $t_1$

Regular expressions to add to edges

$R_1$: $0 \cup 10^*2$
$R_2$: $2 \cup 10^*1$
$R_3$: $1 \cup 20^*2$
$R_4$: $0 \cup 20^*1$
Splicing out state $t_2$ (and then $t_0$)

Delete $t_2$ now that it is redundant

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

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

Diagram:
- Start state $s$ transitions to $t_0$ on $\varepsilon$.
- $t_0$ loops back to itself on $\varepsilon$.
- $t_0$ transitions to final state $f$ on $\varepsilon$. 


Splicing out state $t_2$ (and then $t_0$)

Create direct $(s,f)$ edge so we can delete $t_0$

\[
\begin{align*}
R_1 & : 0 \cup 10^*2 \\
R_2 & : 2 \cup 10^*1 \\
R_3 & : 1 \cup 20^*2 \\
R_4 & : 0 \cup 20^*1 \\
R_5 & : R_1 \cup R_2 R_4^* R_3
\end{align*}
\]
Splicing out state $t_2$ (and then $t_0$)

Regular expressions to add to edges

$R_1$: $0 \cup 10^*2$
$R_2$: $2 \cup 10^*1$
$R_3$: $1 \cup 20^*2$
$R_4$: $0 \cup 20^*1$
$R_5$: $R_1 \cup R_2 R_4^* R_3$

$t_0 \xrightarrow{R_5} t_1 \xrightarrow{R_5} t_0$: $R_5^*$
Splicing out state \( t_2 \) (and then \( t_0 \))

Delete \( t_0 \) now that it is redundant

\[
R_1: \ 0 \cup 10^*2 \\
R_2: \ 2 \cup 10^*1 \\
R_3: \ 1 \cup 20^*2 \\
R_4: \ 0 \cup 20^*1 \\
R_5: \ R_1 \cup R_2R_4^*R_3 \\
R_6: \ R_5^* \\
\]

Graph: 
- \( s \) is the start state.
- \( f \) is the finish state.
- \( R_6 \) is the transition from \( s \) to \( f \).
Splicing out state $t_2$ (and then $t_0$)

Regular expressions to add to edges

\begin{align*}
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 \\
R_6 &: R_5^* \\
\end{align*}

Final regular expression: $R_6 =$ 

\[(0 \cup 10^*2 \cup (2 \cup 10^*1)(0 \cup 20^*1)^*(1 \cup 20^*2))^*\]
The story so far...

REs \subseteq \text{CFGs}

\equiv

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

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