FORMAL LANGUAGES, AUTOMATA AND COMPUTATION

REGULAR LANGUAGES

NONDETERMINISTIC FINITE STATE AUTOMATA

Carnegie Mellon University in Qatar
Symbols, Alphabet, Strings, $\Sigma^*$, Languages, $2^{\Sigma^*}$

Deterministic Finite State Automata

- States, Labels, Start State, Final States, Transitions
- Extended State Transition Function
- DFAs accept regular languages
Since regular languages are sets, we can combine them with the usual set operations:

- Union
- Intersection
- Difference

**Theorem**

*If* $L_1$ *and* $L_2$ *are regular languages, so are* $L_1 \cup L_2$, $L_1 \cap L_2$, *and* $L_1 - L_2$.

**Proof Idea**

Construct cross-product DFAs
**CROSS-PRODUCT DFAs**

- A single DFA which simulates operation of two DFAs in parallel!
- Let the two DFAs be $M_1$ and $M_2$ accepting regular languages $L_1$ and $L_2$
  
  1. $M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$
  2. $M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$
- We want to construct DFAs $M = (Q, \Sigma, \delta, q_0, F)$ that recognize
  
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1 - L_2$
We need to construct $M = (Q, \Sigma, \delta, q_0, F)$

- $Q$ is pairs of states, one from $M_1$ and one from $M_2$
  - $Q = \{(q_1, q_2) | q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$
  - $Q = Q_1 \times Q_2$

- $q_0 = (q_0^1, q_0^2)$

- $\delta(((q_i^1, q_j^2), x) = (\delta_1(q_i^1, x), \delta_2(q_j^2, x))$

- Union: $F = \{(q_1, q_2) | q_1 \in F_1 \text{ or } q_2 \in F_2\}$

- Intersection: $F = \{(q_1, q_2) | q_1 \in F_1 \text{ and } q_2 \in F_2\}$

- Difference: $F = \{(q_1, q_2) | q_1 \in F_1 \text{ and } q_2 \notin F_2\}$
CROSS-PRODUCT DFA EXAMPLE

**Strings with even number of 1s**

\[ M_1 \]

\[ \begin{array}{c}
q_0^1 \\
0 \\
1 \\
q_1^1 \\
0 \\
1 \\
\end{array} \]

**Strings with odd number of 0s**

\[ M_2 \]

\[ \begin{array}{c}
q_0^2 \\
1 \\
0 \\
q_1^2 \\
1 \\
0 \\
\end{array} \]
DFA for $L_1 \cup L_2$

- DFA for $L_1 \cup L_2$ accepts when either $M_1$ or $M_2$ accepts.
DFA for $L_1 \cap L_2$ accepts when both $M_1$ and $M_2$ accept.
DFA for $L_1 - L_2$

- DFA for $L_1 - L_2$ accepts when $M_1$ accepts and $M_2$ rejects.

(Carnegie Mellon University in Qatar) Slides for 15-453 Lecture 3 Spring 2011
Another example: Find the cross-product DFA for

- DFA for binary numbers divisible by 3
- DFA for binary numbers divisible by 2
Other Regular Operations

- **Reverse:** $L^R = \{\omega = a_1 \ldots a_n | \omega^R = a_n \ldots a_1 \in L\}$
- **Concatenation:** $L_1 \cdot L_2 = \{\omega \nu | \omega \in L_1 \text{ and } \nu \in L_2\}$
- **Star Closure:** $L^* = \{\omega_1 \omega_2 \ldots \omega_k | k \geq 0 \text{ and } \omega_i \in L\}$
The reverse of a regular language is also a regular language.

- If a language can be recognized by a DFA that reads strings from right to left, then there is an “normal” DFA (one that reads from left to right) that accepts the same language.
- Counter-intuitive! DFAs have finite memory...
Assume $L$ is a regular language. Let $M$ be a DFA that recognizes $L$.

We will build a machine $M^R$ that accepts $L^R$.

If $M$ accepts $\omega$, then $\omega$ describes a directed path, in $M$, from the start state to a final state.

First attempt: Try to define $M^R$ as $M$ as follows:

- Reverse all transitions
- Turn the start state to a final state
- Turn the final states to start states!

But, as such, $M^R$ is not always a DFA.

- It could have many start states.
- Some states may have too many outgoing transitions or none at all!
What language does this DFA recognize?

- All strings that contain a substring of 2 or more 0s followed by a 1.
What happens with input 100?

- There are multiple transitions from a state labeled with the same symbol.
- State transitions are not deterministic any more: the next state is not uniquely determined by the current state and the current input. → Nondeterminism
We will say that this machine accepts a string if there is some path that reaches an accept state from a start state.
HOW DOES NONDETERMINISM WORK?

- When a nondeterministic finite state automaton (NFA) reads an input symbol and there are multiple transitions labeled with that symbol
  - It splits into multiple copies of itself, and
  - follows all possibilities in parallel.
Deterministic vs Nondeterministic Computation

Deterministic Computation

Non-Deterministic Computation

accept or reject

reject

accept
HOW DOES NONDETERMINISM WORK?

- When a nondeterministic finite state automaton (NFA) reads an input symbol and there are multiple transitions with labeled with that symbol
  - It splits into multiple copies of itself, and
  - follows all possibilities in parallel.

- Each copy of the machine takes one of the possible ways to proceed and continues as before.

- If there are subsequent choices, the machine splits again.
  - We have an unending supply of these machines that we can boot at any point to any state!
A state need not have a transition with every symbol in $\Sigma$

- No transition with the next input symbol? $\Rightarrow$ that copy of the machine dies, along with the branch of computation associated with it.
- If any copy of the machine is in a final state at the end of the input, the NFA accepts the input string.

NFAs can have transitions labeled with $\epsilon$ – the empty string.
If a transition with $\epsilon$ label is encountered, something similar happens:

- The machine does **not** read the next input symbol.
- It splits into multiple copies, one following each $\epsilon$ transition, and one staying at the current state.

What the NFA arrives at $p$ (say after having read input $a$, it splits into 3 copies
NFA Example

- Accepts all strings over \( \Sigma = \{a, b, c\} \) with at least one of the symbols occurring an odd number of times.

- For example, the machine copy taking the upper \( \epsilon \) transition guesses that there are an odd number of \( a \)'s and then tries to verify it.
Nondeterminism

- So nondeterminism can also be viewed as
  - guessing the future, and
  - then verifying it as the rest of the input is read in.
- If the machine’s guess is not verifiable, it dies!
Accepts all strings over $\Sigma = \{0, 1\}$ where the 3rd symbol from the end is a 1.

- How do you know that a symbol is the 3rd symbol from the end?
- The start state guesses every 1 is the 3rd from the end, and then the rest tries to verify that it is or it is not.
- The machine dies if you reach the final state and you get one more symbol.
A Nondeterministic Finite State Acceptor (NFA) is defined as the 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states
- $\Sigma$ is a finite set of symbols – the alphabet
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$, is the next-state function
  - $2^Q = \{P | P \subseteq Q\}$
- $q_0 \in Q$ is the (label of the) start state
- $F \subseteq Q$ is the set of final (accepting) states

$\delta$ maps states and inputs (including $\epsilon$) to a set of possible next states

Similarly $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$

- $\delta^*(q, \epsilon) = \{q\}$
- $\delta^*(q, \omega \cdot a) = \{p | \exists r \in \delta^*(q, \omega) \text{ such that } p \in \delta(r, a)\}$
  - $a$ could be $\epsilon$
An NFA accepts a string $\omega = x_1 x_2 \cdots x_n$ if a sequence of states $r_0 r_1 r_2 \cdots r_n$, $r_i \in Q$ exist such that

1. $r_0 = q_0$ (Start in the initial state)
2. $r_i \in \delta(r_{i-1}, x_i)$ for $i = 1, 2, \ldots n$ (Move from state to state – nondeterministically: $r_i$ is one of the allowable next states)
3. $r_n \in F$ (End up in a final state)
We know that DFAs accept regular languages.

Are NFAs **strictly more powerful** than DFAs?

Are there languages that some NFA will accept but no DFA can accept?

It turns out that NFAs and DFAs accept the same set of languages.

- $Q$ is finite $\implies |2^Q| = |\{P | P \subseteq Q\}| = 2^{|Q|} \text{ is also finite.}$
THEOREM

Every NFA has an equivalent DFA.

PROOF IDEA

- Convert the NFA to an equivalent DFA that accepts the same language.
- If the NFA has $k$ states, then there are $2^k$ possible subsets (still finite)
- The states of the DFA are labeled with subsets of the states of the NFA
- Thus the DFA can have up to $2^k$ states.
THEOREM

Every NFA has an equivalent DFA.

CONSTRUCTION

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We construct $M = (Q', \Sigma, \delta', q'_0, F')$.

1. $Q' = 2^Q$, the power set of $Q$
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{ q \in Q \mid q \in \epsilon(\delta(r, a)) \text{ for some } r \in R \}$
   - For $R \in Q$, the $\epsilon$-closure of $R$, is defined as $\epsilon(R) = \{ q \mid q \text{ is reachable from some } r \in R \text{ by traveling along zero or more } \epsilon \text{ transitions} \}$

3. $q'_0 = \epsilon(\{ q_0 \})$
4. $F' = \{ R \in Q' \mid R \cap F \neq \phi \}$: at least one of the states in $R$ is a final state of $N$
NFA Example

- Note that $q_0$ has an $\epsilon$-transition.
- Some states (e.g., $q_1$) do not have a transition for some of the symbols in $\Sigma$. Machine dies if it sees input 1 when it is in state $q_1$.
- $\epsilon(\{q_0\}) = \{q_0, q_1\}$
Given $N = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\})$, construct $M = (Q', \Sigma, \delta', q'_0, F')$.

$\epsilon(\{1\}) = \{1, 3\}$

<table>
<thead>
<tr>
<th>$\delta'$</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>$\phi$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>${1}$</td>
<td>$\phi$</td>
<td>${2}$</td>
</tr>
<tr>
<td>${2}$</td>
<td>${2, 3}$</td>
<td>${3}$</td>
</tr>
<tr>
<td>${3}$</td>
<td>${1, 3}$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>${1, 2}$</td>
<td>${2, 3}$</td>
<td>${2, 3}$</td>
</tr>
<tr>
<td>${1, 3}$</td>
<td>${1, 3}$</td>
<td>${2}$</td>
</tr>
<tr>
<td>${2, 3}$</td>
<td>${1, 2, 3}$</td>
<td>${3}$</td>
</tr>
<tr>
<td>${1, 2, 3}$</td>
<td>${1, 2, 3}$</td>
<td>${2, 3}$</td>
</tr>
</tbody>
</table>
NFA to DFA Conversion Example

Given \( N = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\}) \), construct 
\( M = (Q', \Sigma, \delta', q'_0, F') \).

\( \epsilon(\{1\}) = \{1, 3\} \)

\( \{1, 3\} \) is the start state of \( M \)
NFA to DFA Conversion Example

Given \( N = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\}) \), construct \( M = (Q', \Sigma, \delta', q'_0, F') \).

- States \( \{1\} \) and \( \{1, 2\} \) do not appear as the next state in any transition! They can be removed.
- States with labels \( \{1, 3\} \) and \( \{1, 2, 3\} \) are the final states of \( M \).
- We can now relabel the states as we wish!

<table>
<thead>
<tr>
<th></th>
<th>( \delta' )</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_5 )</td>
<td>( \phi )</td>
<td>( \phi )</td>
<td>( \phi )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( {2} )</td>
<td>( {2, 3} )</td>
<td>( {3} )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( {3} )</td>
<td>( {1, 3} )</td>
<td>( \phi )</td>
</tr>
<tr>
<td>( q_0 )</td>
<td>( {1, 3} )</td>
<td>( {1, 3} )</td>
<td>( {2} )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( {2, 3} )</td>
<td>( {1, 2, 3} )</td>
<td>( {3} )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( {1, 2, 3} )</td>
<td>( {1, 2, 3} )</td>
<td>( {2, 3} )</td>
</tr>
</tbody>
</table>
NFA to DFA Conversion Example

Given $N = (\{1, 2, 3\}, \{a, b\}, \delta, 1, \{1\})$, construct $M = (Q', \Sigma, \delta', q'_0, F')$.

- States $\{1\}$ and $\{1, 2\}$ do not appear as the next state in any transition! They can be removed.
- States with labels $\{1, 3\}$ and $\{1, 2, 3\}$ are the final states of $M$.
- We can now relabel the states as we wish!

<table>
<thead>
<tr>
<th>$\delta'$</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_5$</td>
<td>$q_5$</td>
<td>$q_5$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_0$</td>
<td>$q_5$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_4$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$q_4$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>