9.1 Non-Deterministic Finite Automata

In this section we will introduce the concept of a non-deterministic finite automaton (NDFA / NFA). An NDFA is the non-deterministic equivalent of a DFA. It is defined in much the same way as a DFA.

Def: A non-deterministic finite automata is an \( M = (S, A, \delta, s_0, F) \) where...

1. \( S \) is the finite set of states of control
2. \( A \) is the alphabet from which input symbols are chosen
3. \( \delta \) is the state transition function
   \[ \delta : S \times (A \cup \{ \epsilon \}) \to 2^S, \text{ where } 2^S \text{ denotes the set of all subsets of } S \]
4. \( s_0 \) is the initial state of the finite control
5. \( F \subseteq S \) is the set of final or accepting states

Note that the definition above only differs from the DFA definition with regards to the transition state function. In an NDFA, a state can accept a symbol from the alphabet and transition to any subset of the other states in the automaton. In other words, the NDFA can non-deterministically transition to one or more defined states on the same input symbol.

Let’s look at an example of an NDFA.

9.2 Example of an NDFA

Consider an NDFA \( M \) that accepts all strings that end in \( aba \). That is, \( L(M) = (a + b)^*aba \). We construct \( M \) as:

\[ M = (\{s_1, s_2, s_3, s_4\}, \{a, b\}, \delta, s_1, \{s_4\}) \]

where \( \delta \) is given by:

<table>
<thead>
<tr>
<th>State ( s )</th>
<th>Input ( \epsilon )</th>
<th>( a )</th>
<th>( b )</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>{( s_1, s_2 )}</td>
<td>{( s_1 )}</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( {s_3})</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( {s_4})</td>
<td>( \varnothing )</td>
<td>( {s_1})</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( s_4 )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( {s_2})</td>
</tr>
</tbody>
</table>

Below is a drawing of \( M \):
9.3 Transition Diagrams

Let $M = (S, A, \delta, s_0, F)$ be an NDFA. The transition diagram associated with $M$ is a directed graph $G = (S, E)$ with labelled edges. Think of this as the formal name of the drawing of $M$ above. The set of edges $E$ and their labels are defined as follows:

- If $\delta(s, a)$ contains $s'$ for some $a \in A \cup \{\epsilon\}$ then the edge $(s, s')$ is in $E$.
- The label of $(s, s')$ is the set of $b \in A \cup \{\epsilon\}$ such that $\delta(s, b)$ contains $s'$.

9.4 Theorems Relating DFAs, NDFAs, and Regular Languages

**Theorem 9.1.** Each language accepted by a non-deterministic finite automaton is a regular language.

**Theorem 9.2.** For every regular expression $\alpha$ there is a non-deterministic finite automata accepting the language denoted by the expression.

Corollary to 15.2: The set of all languages described by regular expressions is equivalent to the set of all languages described by NDFAs.

**Theorem 9.3.** If $L$ is a regular language then $L$ is accepted by a deterministic finite automaton.

Corollary to 15.3: The set of all languages described by NDFAs is equivalent to the set of all languages described by DFAs.

9.5 Constructing a DFA to Recognize a Suffix Pattern

We will now show an algorithm to construct a DFA for the language $A^*p$, where $A$ is an alphabet and $p$ is a pattern composed of symbols from $A$. In other words, the DFA will recognize any string over the alphabet $A$ that ends in $p$. The DFA we construct will make exactly one state transition per input symbol.
Algorithm 1: Construction of DFA for $A^*p$

Input: A pattern string $p = p_1p_2...p_L$ over $A$ where $p_{L+1} = \$,$ some new symbol that is not in $A$

Output: A DFA $M$ such that $L(M) = A^*p$

1. Use Failure Function Algorithm (see previous notes) to construct the failure function, $f$, for $p$.
2. Let $M = (S, A, \delta, 0, \{L\})$, where $S = \{0, 1, 2, ..., L\}$ and $\delta$ is constructed as follows:

   procedure CONSTRUCT $\delta$
     for $j \leftarrow 1, 2, \ldots, L$ do
       $\delta(j-1, p_j) = j$
     end for
     for each $a \in A$ such that $a \neq p_1$ do
       $\delta(a, 0) = 0$
     end for
     for $j \leftarrow 1, 2, \ldots, L$ do
       for each $a \in A$ such that $a \neq p_{j+1}$ do
         $\delta(j, a) = \delta(f(j), a)$
       end for
     end for
   end procedure

Theorem 9.4. The algorithm above constructs a DFA $M$ such that $(0, t_1t_2...t_k)^* \vdash (j, \epsilon)$ if and only if $p_1...p_j$ is a suffix of $t_1...t_k$, but for no $i > j$ is $p_1...p_i$ a suffix of $t_1...t_k$.

The theorem above can be proved by induction. This proof is left as an exercise to the reader. Instead we will present an example of a DFA that accepts the $A^*p$ where $A = \{a, b\}$ and $p = aabbaab$. That is, the DFA below accepts all strings given by the regular expression $(a+b)^*aabbaab$.

The only difference between $M$ and $M_p$ is that $M$ has precomputed the next state in case of a mismatch. Thus $M$ makes exactly one state transition on each input symbol.

9.6 Conclusion

As stated previously:

Theorem 9.5. In $O(|p| + |t|)$ time we can determine whether $p$ is a substring of $t$. 