9.1 Non-Deterministic Finite Automata

In this section we will introduce the concept of an 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 $M = (S, A, δ, s_o, F)$ where...

1. $S$ is a finite set of **states** of control
2. $A$ is the alphabet from which input symbols are chosen
3. $δ$ is the **state transition function**
   
   $δ : S × (A ∪ \{\epsilon\}) → 2^S$, where $2^S$ denotes the set of all subsets of $S$
4. $s_o$ is the **initial state** of the finite control
5. $F ⊆ S$ 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:

$M = (\{s_1, s_2, s_3, s_4\}, \{a, b\}, δ, s_1, \{s_4\})$ where $δ$ is given by:

<table>
<thead>
<tr>
<th>State</th>
<th>Input →</th>
<th>a</th>
<th>b</th>
<th>ϵ</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td></td>
<td>{s₁,s₂}</td>
<td>{s₁}</td>
<td>∅</td>
</tr>
<tr>
<td>s₂</td>
<td></td>
<td>∅</td>
<td>{s₃}</td>
<td>∅</td>
</tr>
<tr>
<td>s₃</td>
<td></td>
<td>{s₄}</td>
<td>∅</td>
<td>{s₁}</td>
</tr>
<tr>
<td>s₄</td>
<td></td>
<td>∅</td>
<td>∅</td>
<td>{s₂}</td>
</tr>
</tbody>
</table>


Below is a drawing of $M$:

![Diagram 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.

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

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

**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 The Failure Function

Continuing from last class, suppose after having read $t_1t_2...t_k$ (the first $k$ characters of the text $t$) we find that $M_p$ (the pattern matching machine we are trying to construct) is in State $J$. This implies that the last $j$ symbols of $t_1t_2...t_k$ are $p_1p_2...p_j$ and the last $m$ symbols of $t_1t_2...t_k$ are not a prefix of $p = p_1...p_l$ for $m > j$. But what about the case where $t_{k+1} \neq p_{j+1}$? That is, when the next symbol of the text is not the next symbol in the pattern. In this case $M_p$ enters the highest number state $i$ such that $p_1p_2...p_i$ is a suffix
of \( t_1t_2...t_{k+1} \). To help determine \( i \), the machine \( M_p \) has associated with it an integer valued function \( f \). This function \( f \) is called the failure function for the pattern \( p \). We define \( f \) such that \( f(j) \) is the largest integer \( s \) less than \( j \) for which \( p_1...p_s \) is a suffix of \( p_1p_2...p_j \). That is, \( f(j) \) is the largest \( s < j \) such that \( p_1p_2...p_s = p_{j-s+1}p_{j-s+2}...p_j \). In words, \( s \) is the number of positions we can “fall back” to keep looking for the pattern \( p \), based on the fact that the suffixes of some prefixes of \( p \) are prefixes of \( p \) itself. If there is no such \( s \geq 1 \), then \( f(j) = 0 \). The next lecture will cover how to compute the failure function and use it to run the wondrous Knuth-Morris-Pratt algorithm.

Here is an example of the failure function for the pattern \( p = \text{aabbaab} \). The function, \( f \) maps an integer index \( i \) to another integer such that \( 0 \leq f(i) < i \). The inputs and outputs of \( f \) are represented in the table below:

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_i )</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

For example, \( f(7) = 3 \) since \( \text{aab} \) is the longest proper prefix of \( \text{aabbaab} \) that is also a suffix of \( \text{aabbaab} \).

### 9.6 Failure Function Algorithm

We will now present an algorithm to compute the failure function for a pattern \( p \). To see how the failure function is used by \( M_p \), let us define the function \( f^m(j) \) as follows:

i) \( f^1(j) = f(j) \), and

ii) \( f^m(j) = f(f^{m-1}(j)) \), for \( m > 1 \)

That is, \( f^m(j) \) is just \( f \) applied \( m \) times to \( j \). In our example above, \( f^2(6) = 1 \).

Supposed once again that \( M_p \) is in state \( j \), having read \( t_1t_2...t_k \) and \( t_{k+1} \neq p_{j+1} \). At this point \( M_p \) applies the failure function repeatedly to \( j \) until it finds the smallest value of \( m \) for which either:

**Case 1:** \( f^m(j) = u \) and \( t_{k+1} = p_{k+1} \), or

**Case 2:** \( f^m(j) = 0 \) and \( t_{k+1} \neq p_1 \)

That is, \( M_p \) backs up through states \( f^1(j) \), \( f^2(j) \),..., and so on until either Case 1 or Case 2 holds for \( f^m(j) \) but not for \( f^{m-1}(j) \). In Case 1 \( M_p \) enters State \( u+1 \) and in Case 2 \( M_p \) enters State 0. In either case, the input pointer is advanced to position \( t_{k+2} \). In Case 1 it is easy to verify that if \( p_1p_2...p_j \) was the longest prefix of \( p \) that is a suffix of \( t_1t_2...t_k \) then \( p_1p_2...p_{f^m(j)+1} \) is the longest prefix of \( p \) that is a suffix of \( t_1t_2...t_k \).

In Case 2, no prefix of \( p \) is a suffix of \( t_1t_2...t_k \).

\( M_p \) then proceeds processing input symbol \( t_{k+2} \). \( M_p \) continues operating in this fashion either until it enters the final state \( l \) in which case we know that the last input symbols constitute an instance of the patterns \( p = p_1p_2...p_l \), or until \( M_p \) has processed the last input symbol of \( t \) without entering State \( l \), in which case we know that pattern \( p \) is not found in the input text \( t \).
9.7 An Example of the Failure Function

Now we’ll present an example of the failure function, as represented by a pattern matching machine $M_p$. Let $p = \text{aabbaab}$ and $t = \text{abaabaabbaab}$. $M_p$ is as follows, where the dashed arrows represent the failure function:

For example, initially $M_p$ is in State 0. On reading the first symbol of $t$, $M_p$ enters state 1. Since there is no transition from state 1 on the second input symbol of $t$ (ie. b), $M_p$ enters state 0. That is, $M_p$ goes back to the state given by the output of the failure function from State 1. Now since the first symbol of $p$ is not $t_2$, Case 2 from above prevails and $M_p$ remains in state 0. From here $M_p$ continues consuming characters in the input string and following the corresponding arrows. If the machine ever reaches State 7, the pattern $p$ has been found in the text $t$. The next set of notes will go into how we can calculate the failure function and apply it the task of finding any pattern $p$ in any body of text.