1 Regular Expression to Minimized DFA (30 min)

Given the regular expression \((a \mid b)^* (c \mid d)^*\):

1. Create an equivalent nondeterministic finite automaton by our “mini-machine” method (aka Thompson’s method, but never mind).

2. Use the subset construction to convert the NDFA to a DFA.

3. Use the equivalence class argument to minimize the DFA.

Answer:
Possibly missing out a few pesky useless states in the NDFA, the process should look something like Fig. 1.

2 Regular Expressions (15 min)

1. Describe the strings produced by this RE in English (not “how”, but “what”).
   \(0^*(100^*)^* \mid 0^*(100^*)^*1\).

2. If we add the null character \(\epsilon\) to the alphabet, can we simplify the RegExp of the last question?

3. Write a Regular Expression for the language: “All strings of a’s and b’s such that either the number of a’s is evenly divisible by two or the string ends in exactly three b’s.”

Answer:
1. All strings of 0,1’s not having two consecutive 1’s.

2. \((0*100*)(\text{eps}|1)\) Should work, no?

3. \(b*(ab*a)*b^* \mid ((a \mid b)*a) \mid \text{eps})\) bbb FFQ: There was a question here about whether I meant “or” or “exclusive or”; of course I meant “—”, which means only one or another (not both) of the patterns gets run, but of course some strings emitted (matched) by one pattern are also legal for the other, as in \text{aaaabbb}.

3 Parsing (20 min)

Here’s a simple little CFG named BOT: Starting symbol is A.

\[
\begin{align*}
A & \rightarrow Ab \\
A & \rightarrow Bc \\
A & \rightarrow Bd \\
B & \rightarrow e
\end{align*}
\]

1. Draw a parse tree for the string \text{ecbb} using BOT.

2. Can you use BOT with an LL(1) parser? Why or why not?
3. If your answer to previous question was No, then rewrite BOT into a new grammar TOP that you can use in an LL(1) parser.

4. Draw a parse tree for the string ecbb using TOP.

5. What are the sets First(A) and Follow(A) in BOT?

Answer:

1. A
   \ /
   A  b
   \/
   A  b
   \/
   B  c
   |
   e

2. BOT has both left recursion and common prefixes, dooming it for tops-down use.

3. A → BKX
   X → bX | epsilon
   K → c | d
   B → e

4. A
   \ /
   B  K  X
   | | |
   e  c  b  X
   | | |
   b  X
   | |
   epsilon

5. First(A) is \{e\} and Follow(A) is \{b\}. 
| Chomsky Type | 0 | | | |
| Language Name | | Regular Expression | | |
| Acceptance Complexity | | Exponential | | |
| Formal Machine | | | Pushdown Automaton |

Table 1: FLAT Categories

4 **FLAT (10 min)**

Fill in Table 1 (Hint: the top row is not [0,1,2,3]). You may write your table into your bluebook or use this page and fold it in.

Answer:
<table>
<thead>
<tr>
<th>Chomsky Type</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Name</td>
<td>Recursively Enumerable</td>
<td>Regular Expression</td>
<td>Context Sensitive</td>
<td>Context Free</td>
</tr>
<tr>
<td>Acceptance Complexity</td>
<td>Uncomputable</td>
<td>linear</td>
<td>Exponential</td>
<td>polynomial</td>
</tr>
<tr>
<td>Formal Machine</td>
<td>Turing Machine</td>
<td>Finite Automaton</td>
<td>Linear Bounded Automaton</td>
<td>Pushdown Automaton</td>
</tr>
</tbody>
</table>

Table 2: FLAT Categories