Problem Set 1
Fall 23
Due: Monday, September 4th.

1. Yea or Nay

State whether the following statements are true or false, and justify your answer.

(a) Just-in-time (JIT) compilation typically yields slower execution than static compilation.

(b) Jim creates a new language, Jaython, which changes Java’s syntax to be Python-like (supporting elif, indentation-based blocks, etc.). To build a Jaython compiler, Jim only needs to modify the lexer/parser of a Java compiler.

Solution:

(a) True. JIT compilation happens at run time and needs extra time.

(b) True. The modified lexer/parser will handle the modified syntax and generate the same AST as a standard Java compiler does. Therefore the other components of the Java compiler can be reused.

2. Regular Expression

For strings containing the letters $a$ and $b$ give a regular expression that captures all strings $w$ such that

(a) The same letter does not repeat in consecutive.

(b) Starts and ends with the same letter.

Example,

(a) Expressions accepted: $a$, $aba$

(b) Expressions not accepted: $ab$, $aa$

Solution:

The regular expression that captures the above language is:

$$a(ba)^*|b(ab)^*$$
The first (or second) clause corresponds to the case that \( a \) (or \( b \)) appears at the start of string \( w \). The subexpression \( ab \) or \( ba \) represents any substring that does not contain any repeating \( a \) or \( b \) in consecutive; and the wrapping Kleene closure \( (\ldots)^* \) allows this pattern to repeat arbitrary number of times. Note that \( \lambda \) belongs to any Kleene closure, and also belongs to this language, as zero is also even.

3. NFA

Give a non-deterministic finite automaton that captures the regular expression from above. Show the automaton in graphical form.

Solution:
We will build this NFA up piece by piece. First, note that the NFA for \( a \) is:

\[
\begin{array}{c}
\text{Initial State} \\
\rightarrow \quad a \\
\text{Final State}
\end{array}
\]

and the NFA for \( ba \) is:

\[
\begin{array}{c}
\text{Initial State} \\
\rightarrow \quad b \\
\rightarrow \quad a \\
\text{Final State}
\end{array}
\]

As per the construction rule for Kleene closure, we can then build an NFA for \( a(ba)^* \) as follows:

\[
\begin{array}{c}
\text{Initial State} \\
\rightarrow \quad a \\
\rightarrow \quad \lambda \\
\rightarrow \quad b \\
\rightarrow \quad a \\
\rightarrow \quad \lambda \\
\text{Final State}
\end{array}
\]

Similarly, the NFA for \( b(ab)^* \) is:

\[
\begin{array}{c}
\text{Initial State} \\
\rightarrow \quad b \\
\rightarrow \quad \lambda \\
\rightarrow \quad a \\
\rightarrow \quad b \\
\rightarrow \quad \lambda \\
\text{Final State}
\end{array}
\]

Now we can put the two parts together to build the overall NFA:
Note that we have assigned a label to each state for the convenience of the next step.

4. DFA

Using the construction described in class, give a deterministic version of the automaton.
You only need to show the transition table.

Solution:

Remember that the way to perform DFA construction is to simulate every possible “configuration” of states that can occur after seeing a particular letter (and taking all possible λ transitions). So when we start out, our “pointers” immediately take all possible lambda transitions, giving us an initial state of \{A, B, G\}. We can now build the table from here:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>final?</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABG</td>
<td>CD</td>
<td>HI</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>CD</td>
<td>φ</td>
<td>E</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>HI</td>
<td>J</td>
<td>φ</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>FCD</td>
<td>φ</td>
<td>no</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>φ</td>
<td>KHI</td>
<td>no</td>
<td>4</td>
</tr>
<tr>
<td>FCD</td>
<td>φ</td>
<td>E</td>
<td>yes</td>
<td>5</td>
</tr>
<tr>
<td>KHI</td>
<td>J</td>
<td>φ</td>
<td>yes</td>
<td>6</td>
</tr>
</tbody>
</table>

5. Minimization (for in-person courses only)

Give a minimized version of the finite automaton, using the algorithm we used in class.
You only need to show the state transition diagram.
Solution:

When each state gets assigned a number, the NFA looks like this:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>final?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>φ</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>φ</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>φ</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>φ</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>φ</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>φ</td>
<td>yes</td>
</tr>
</tbody>
</table>

The minimization starts with two groups of states, depending on whether the state is accepting:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>final?</th>
</tr>
</thead>
<tbody>
<tr>
<td>034</td>
<td>?</td>
<td>?</td>
<td>no</td>
</tr>
<tr>
<td>1256</td>
<td>?</td>
<td>?</td>
<td>yes</td>
</tr>
</tbody>
</table>

Depending on the transitions on $a$ and $b$ the first state can be refined into the following groups:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>final?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>26</td>
<td>no</td>
</tr>
<tr>
<td>15</td>
<td>φ</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>φ</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>φ</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>φ</td>
<td>26</td>
<td>no</td>
</tr>
</tbody>
</table>

Now all of the four states are distinct and the automaton looks like this:

6. Regular Expression to DFA

Build a minimized, deterministic automaton for the following regular expression:

$ab^*(cb^*)^* \mid cb^*(ab^*)^*$

Solution:

The minimized, deterministic automaton has five states: