building an automaton
Regular expression → NFA → DFA → Lexer
building a non-deterministic automaton

- Can use the features of non-deterministic automata, especially $\lambda$-transitions, to build up an automaton automatically from a regular expression:

<table>
<thead>
<tr>
<th>Expression</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td><img src="image" alt="Diagram a" /></td>
</tr>
<tr>
<td>$\lambda$</td>
<td><img src="image" alt="Diagram λ" /></td>
</tr>
<tr>
<td>AB</td>
<td><img src="image" alt="Diagram AB" /></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>A*</td>
<td><img src="image" alt="Diagram A*" /></td>
</tr>
</tbody>
</table>
• Build automaton for \((a|b)*(b|c)*\)
pros and cons of nfas
power of NFAs

• NFAs are exactly as powerful as regular expressions
  • If you have a regular expression, there exists an NFA that matches it
  • If you have an NFA, there exists a regular expression that defines the set of strings that the NFA matches
• Alternate definition of regular languages: exactly the set of languages that can be accepted by a (non-deterministic?) finite automaton
problems with non-determinism

• Could just build a non-deterministic automaton and call it a day
• But non-determinism has some drawbacks
  • Unpredictable running time: what if you make the wrong choice and have to backtrack?
  • (Causes actual bugs in real code!)
• Turns out that non-deterministic finite automata and deterministic finite automata are equally powerful
  • Can automatically generate a deterministic finite automaton from a non-deterministic one