Lecture 8 - Translating Regular Expressions to Automata

We are able to create the automata by starting with the end state, and then moving toward an empty state. We can start with the full word, and then with each new state we can remove the front letter of the word in order to create an automata.

Example 1:

* The alphabet is {A, B, C}
* Language consists of one word BABC
* RegEx: BABC

A diagram of a diagram

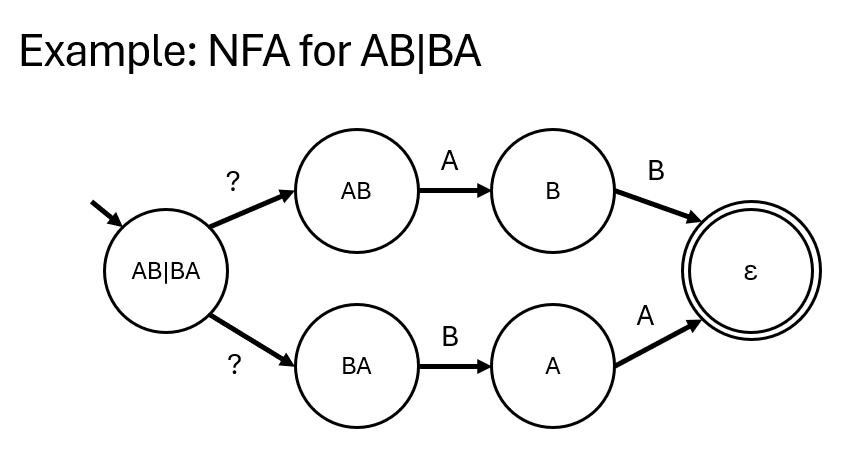
Description automatically generated

Review:

* When we have a sequence of things in our language
  + E.g. a sequence of letters in the word BABC
* We can construct different parts of the automata for each subsequence:
  + BABC
  + ABC
  + BC
  + C
  + Ε (empty)

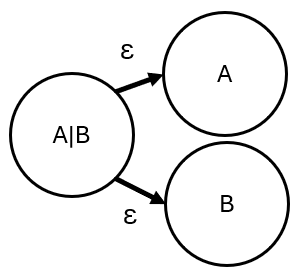
Example 2:

* The alphabet is {A, B}
* Language only has two words: AB and BA
* RegEx: AB|BA



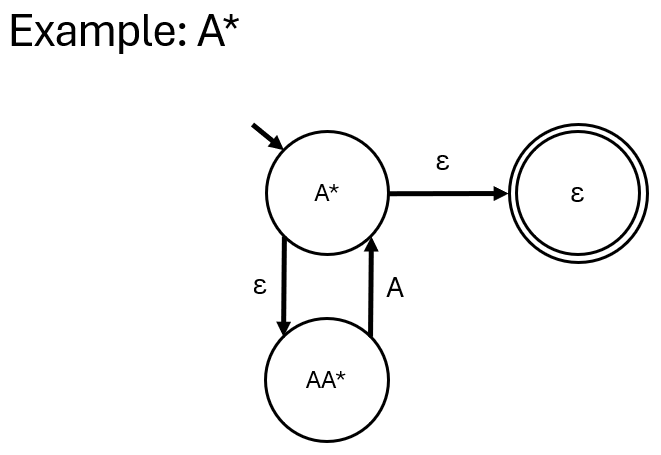
Review:

* If we have a choice in our regular expression
* We can create separate states representing those choices
* Use epsilon transitions to represent making those choices without reading the next character



Example 3:

* Alphabet is {A}
* Language consists of words that only have As in them, any number of As, including 0
* RegEx: A\*



Review:

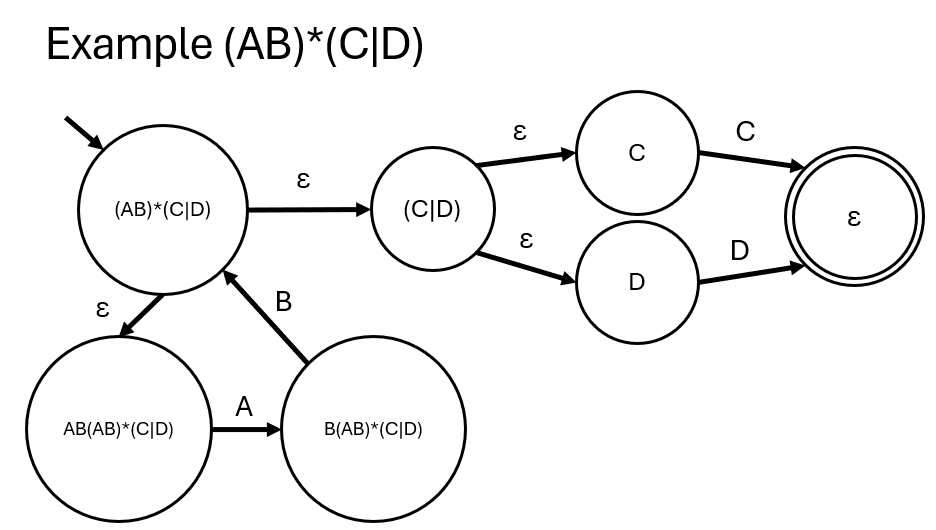
* When we have an iteration in our regular expression
* Create an epsilon transition to the state without that iteration
* And an epsilon transition to a state where one copy of the iteration content is popped out before the iteration

A diagram of a diagram

Description automatically generated

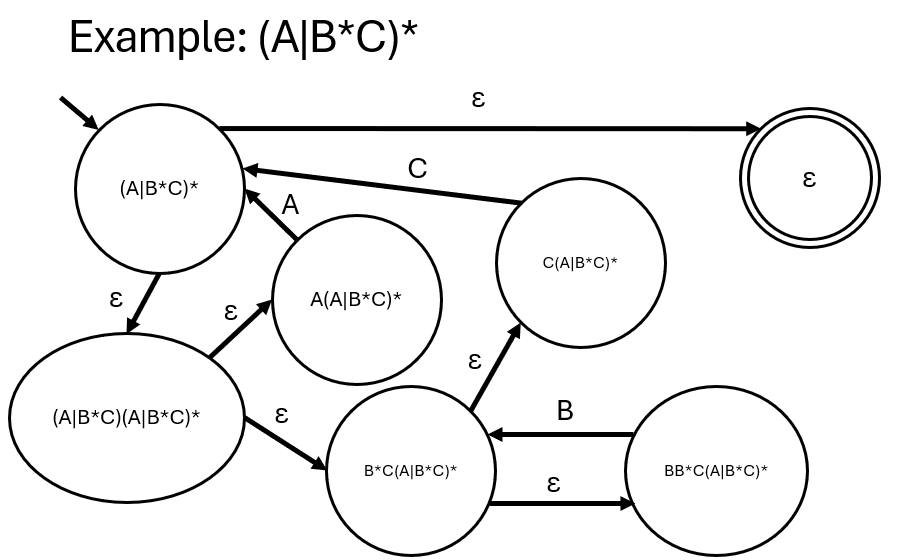
Example 4:

* Let’s combine everything we’ve looked at
* RegEx: (AB)\*(C|D)
* Iteration over AB, followed by a choice between C and D



Example 5:

* RegEx: (A|B\*C)\*



Summary:

* We can always turn a regular expression into an NFA.
* States represent regular expressions
* Epsilon transitions allow state transitions without reading any characters
* All regular expression operations are supported:
  + Sequence
  + Choice
  + Iteration