# Regex Theory: Converting Regex to DFA

Eckel, TJHSST AI2, Spring 2022

## Background & Explanation

Regular expressions arose as a concise way to describe the behaviors of finite automata. If you define a regular expression grammar that contains these grammatical elements:

- ab to specify two tokens in sequence (token a followed by token b)
- a|b to specify two different options (token a OR token b)
- (ab) to group a regular expression into a single token
- a* to specify any number, including zero, of a token (token a any number of times)
- a+ to specify any number, excluding zero, of a token (token a at least once)
- a? to specify an optional token (token a once or not at all)

…then each regular expression corresponds to a specific DFA which can then be executed in a single left to right pass on any input string.

This is not to say that converting regular expressions to DFAs is a trivial process! It proceeds in several stages. First, the regex is converted to a **nondeterministic finite automaton with epsilon moves**. (Nondeterministic means that from each state there may be multiple options of where to move next – the user, not the machine itself, determines which path to follow. Epsilon moves are moves that transition from one state to another without consuming an input character.) This **NFA w/ ε** is next converted to an **NFA without epsilon moves**, or simply an **NFA**. Finally, the **NFA** is converted to a **DFA**.

This process is quite complex; you should not take on this assignment unless you've already taken notes on this topic in class or unless you're prepared to do further research yourself. In class, we will do several examples together.

## Regex > NFA w/ ε

Each of the above pieces of grammar corresponds to a specific set of connections on an **NFA w/ ε**. We will cover these in a lecture in class. Refer to your notes.

At the end of this process, you should have an NFA with some finite number of states and each connection should be labelled with a single character from the language or ε.

## NFA w/ ε > NFA

For each state, find its **epsilon enclosure** (the list of all states that it can reach via only epsilon moves). Inherit finality from any of those nodes. Then, for each letter in the language, find all the states that can be reached from any state in the enclosure via a connection with that letter. Draw a connection directly to all of those nodes from the current node with that letter. In other words:

**I am a state. Consider all the states I can reach with only ε moves.**

- **If any of those states is a final state, then I am now also a final state.**
- **Any non-ε connections that any of those states have to another state I also now have to the same state.**

Delete any epsilon moves.

At the end of this process, you should have an NFA with the same states as before. All connections should be labelled with a single character from the language; no epsilons. You may have several connections with the same label coming from the same node.

## NFA > DFA

For each node that has multiple connections with the same language character, form a new node that combines the characteristics of all of the endpoints of those connections and point this node at that one instead. In other words:

**I am a state. I have multiple moves for a particular input character *c*. Those moves result in a list of states *s*.**

- **Make a new state named after all the states in *s*.**
- **If any state in *s* is final, so is the new state.**
- **Any connection that any state in *s* has is also a connection the new state has.**

At the end of this process, you should have a DFA. It should have all the same states as before, but it should also contain new states constructed as specified above. Some states may be unreachable.

## Finally: Clean Up the DFA

It is likely that this DFA contains extra stuff. You'll need to first **remove any nodes that aren't actually reachable from the start node**. Then, you will want to remove redundant nodes.

To remove redundant nodes:

- Consider only final states grouped together.
- Find any final states that contain exactly the same connections to other states.
- Merge those states together.
- Repeat the process considering only the non-final states grouped together.

## Your Task

This is not a coding assignment. For each of the regular expressions below, on your own paper, complete the process described above. For each regex, I should see a diagram of each of the following:

- Your final NFA w/ ε after completing the first step.
- Your final NFA after completing the second step.
- Your final DFA after completing the third step.
- Your cleaned DFA after removing unreachable & consolidating redundant nodes.

To be clear, that means **for each problem below I should see four separate finite automata**. Feel free to do scratch work on scratch paper, but the final paper that you turn in should contain **those four and only those four drawings** for each problem.

In each drawing, the states should be labelled where 0 is the start state and each state is numbered (or labelled with a list during the NFA > DFA process). A state with a single circle is not a final node; a double circle indicates a final node. Each connection should be labelled with a character from the language (or ε).

## Problems

This is already complicated enough; for the sake of simplicity, every problem below is on the language "ab".

1) abbb|aba+
2) (a*|b*)abb
3) ab?a*
4) a|ab|aa|abb|aba|b*
5) (ab+)*a?b+a*

## Specification

Turn this in to me in class on paper.

This assignment is **complete** if:

- Your name and class period are on the front page.
- For each problem, you have four diagrams – and ONLY four diagrams – as specified above.