

# Proving Kleene's theorem

## 1 Note: addition of sets

Given sets  $A$  and  $B$ , we obtain the set

$$A + B \stackrel{\text{def}}{=} \{(0, x) \mid x \in A\} \cup \{(1, y) \mid y \in B\}$$

For example, if  $A = \{3, 5, 7\}$  and  $B = \{7, 8, 10, 11\}$  then the sum has seven elements:

$$A + B = \{(0, 3), (0, 5), (0, 7), (1, 7), (1, 8), (1, 10), (1, 11)\}$$

By contrast, the union has only six elements:

$$A \cup B = \{3, 5, 7, 8, 10, 11\}$$

## 2 Kleene's theorem

Previously we saw:

**Theorem 1** (Kleene's Theorem) *For a language  $L \subseteq \Sigma^*$ , the following are equivalent.*

1.  $L$  can be described by a regex.
2. The matching problem for  $L$  can be solved by a DFA.

We are going to prove this theorem.

## 3 From regex to DFA

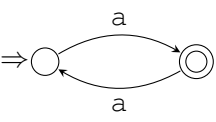
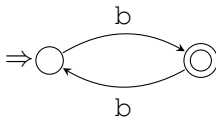
We shall see the forward direction: how to convert a regex into a DFA that recognizes the same language. We shall only study the proof of  $(1) \Rightarrow (2)$ , i.e. we'll see how to turn a regex into a DFA that recognizes the same language. As we saw, it suffices to construct an  $\varepsilon$ NFA, because then we remove the  $\varepsilon$ -transitions to obtain an NFA, and lastly we determinize to obtain a DFA.

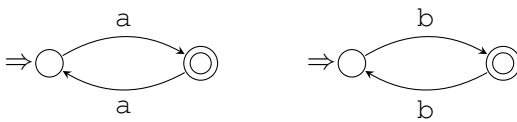
So we want to convert a regex into an  $\varepsilon$ NFA that recognizes the same language.

- The regex  $a$  is recognized by  $\Rightarrow \bigcirc \xrightarrow{a} \bigcirc$  and likewise if  $E$  is  $b$  or  $c$ .
- The regex  $\varepsilon$  is recognized by  $\Rightarrow \bigcirc$

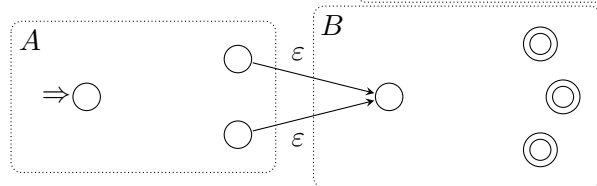
- If  $E_0$  is recognized by  $\Rightarrow \bigcirc$  and  $E_1$  by  $\Rightarrow \bigcirc$  then  $E_0|E_1$  is recognized by  $\Rightarrow \bigcirc$

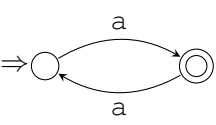
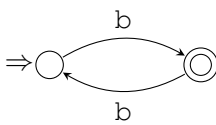
i.e. we take the sum of the two sets of states, or rename the states to ensure disjointness and take the union. Everything else is unchanged.

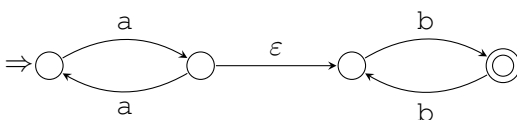
For example, knowing that  $a(aa)^*$  is recognized by  and  $b(bb)^*$  by 

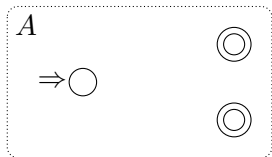
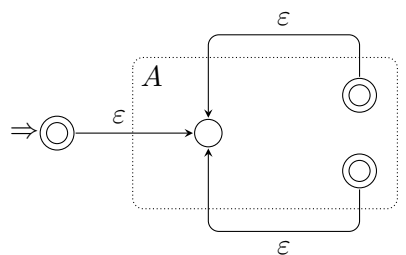
we deduce that  $a(aa)^*b(bb)^*$  is recognized by 

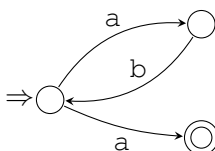
- If  $E_0$  is recognized by  and  $E_1$  by 

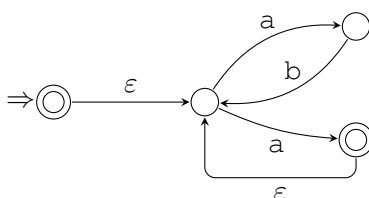
then  $E_0E_1$  is recognized by 

For example, knowing that  $a(aa)^*$  is recognized by  and  $b(bb)^*$  by 

we deduce that  $a(aa)^*b(bb)^*$  is recognized by 

- If  $E$  is recognized by  then  $E^*$  is recognized by 

For example,  $(ab)^*a$  is recognized by 

so  $((ab)^*a)^*$  is recognized by 

- Finally,  $\emptyset$  is recognized by the empty automaton (the partial DFA with no states).

We thus see that for every regex  $E$  there is a DFA that recognizes the same language. This is a course-of-values induction on the *length* of  $E$ . In other words, we prove that the statement is true for  $E$  assuming that it is true for all *shorter* expressions.

In fact, all we need to assume is that the property holds for *subexpressions*. For example, to prove the property for  $E_0E_1$ , we need only assume that it's true for the subexpressions  $E_0$  and  $E_1$ . This kind of argument often appears in computer science, and is called *structural* induction.

Let's follow the procedure described to convert the expression  $((ab|ac)^*(abb)^*)^*$  into an automaton. To save on work, we'll obtain  $\epsilon$ NFAs all the way through, and then, right at the end, we'll remove the  $\epsilon$ s and determinize.

- $a$  gives automaton

- $b$  gives automaton
- $ab$  gives automaton
- $c$  gives automaton
- $ac$  gives automaton
- $ab|ac$  gives automaton
- $(ab|ac)^*$  gives automaton
- $abb$  gives automaton
- $(abb)^*$  gives automaton
- $(ab|ac)^*(abb)^*$  gives automaton
- $((ab|ac)^*(abb)^*)^*$  gives automaton
- Removing  $\varepsilon$ s gives the NFA
- Determinization gives the DFA

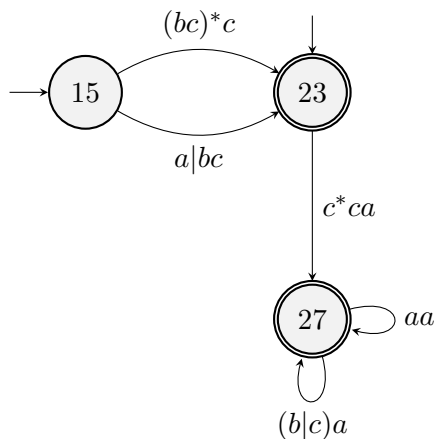
**Negative viewpoint** This argument can also be seen negatively. If there's a regex  $E$  that doesn't have an equivalent DFA, then there's a smaller regex  $E'$  that also doesn't, and therefore an even smaller one  $E''$ , and so on. But these are finite expressions, so this can't continue forever. Contradiction!

## 4 Generalized NFAs

Before explaining how to convert an automaton into a regex, let me first introduce the notion of a *generalized NFA*. This is a version of NFA where each arrow is labelled with a regex. There are finitely many states and arrows.

Given a word  $w$ , we start at an initial state and move from step to step. At each stage, we read in several characters at a time, forming a word  $x$ , and follow an arrow  $E$  that matches  $x$ . If we end on an accepting state, the word is accepted.

**Example 1** Here's a generalized NFA.



Are these words acceptable?

cccaaa	Y/N
aacca	Y/N
acaca	Y/N

Note that any  $\varepsilon$ NFA is also a generalized NFA.

## 5 From Generalized NFA to regex

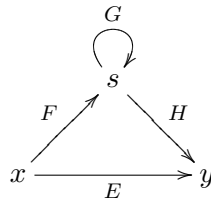
We convert a generalized NFA to a regex in several stages. Note that these operations do not change the language of the automaton, i.e. which words are acceptable.

1. Combine any two distinct arrows  $s \xrightarrow{E} t$  and  $s \xrightarrow{F} t$  into an arrow  $s \xrightarrow{E|F} t$ . Continue until there is at most one arrow between any two states. In particular, there is at most one loop on each state.
2. For the sake of simplicity, we would like an automaton with exactly one arrow between any two states. To achieve this, wherever there is no arrow  $x \rightarrow y$ , we insert a 0-labelled arrow. In particular, when there is no loop on  $x$ , we insert a 0-labelled loop. Since a 0-label cannot be followed, the language is unchanged.

3. Add in a Start state, which becomes the sole initial state, and an End state, which becomes the sole accepting state. The old states are called *intermediate*. Connect Start to each intermediate state  $s$  by an arrow labelled with  $\varepsilon$  if  $s$  was initial, and with 0 otherwise. Connect each intermediate state  $s$  to End by an arrow labelled by  $\varepsilon$  if  $s$  was accepting, and with 0 otherwise. Connect Start to End by an arrow labelled with 0.

Note that there is now exactly one arrow between every pair of states, *except* that no arrow goes into Start, and no arrow comes out of End.

4. Next we remove the intermediate states one by one. (The order of removal doesn't matter.) When we remove a state  $s$ , we remove all the arrows to it and from it, and also adjust the labels on all the other arrows. Specifically, if we had



where neither  $x$  nor  $y$  is equal to  $s$  (but possibly  $x = y$ ), then, after the removal of  $s$ , the new label on  $x \rightarrow y$  will be  $E|FG^*H$ .

5. When there are no intermediate states left, read off the label on Start  $\rightarrow$  End. This is a regex that's equivalent to the automaton we started with.