

Kleene's theorem

Theorem 1 (Kleene's Theorem)

1. For a language $L \subseteq \Sigma^*$, the following are equivalent.
 - a. L can be described by a regex.
 - b. The matching problem for L can be solved by a DFA.

We are going to prove this theorem.

From regex to DFA

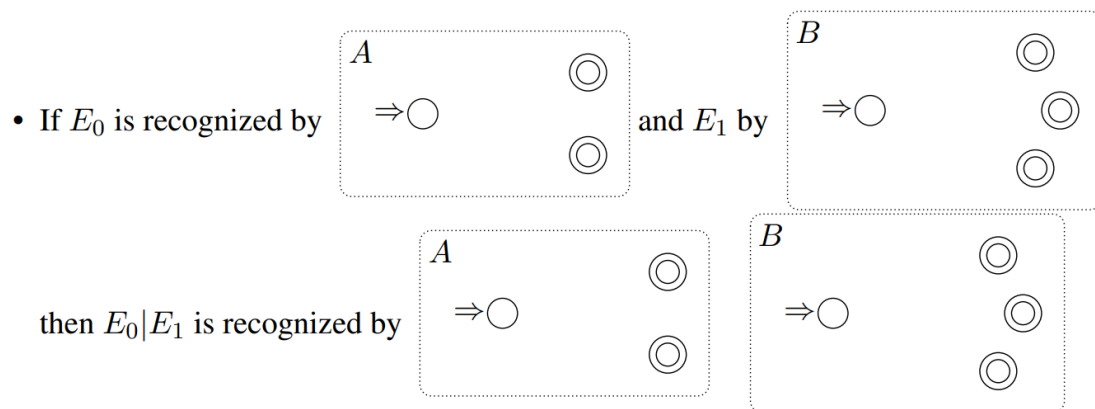
We shall see the forward direction:

How to convert a regex into a DFA that recognizes the same language.

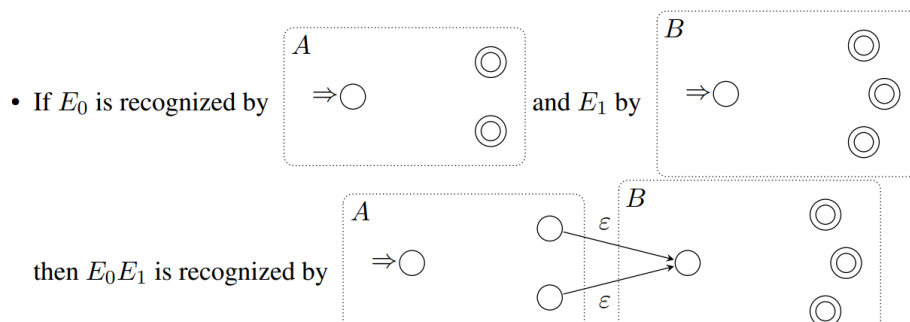
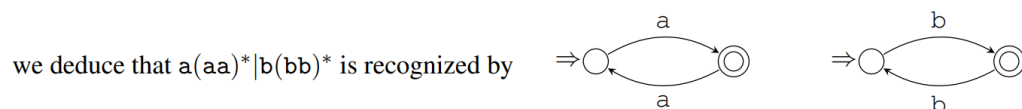
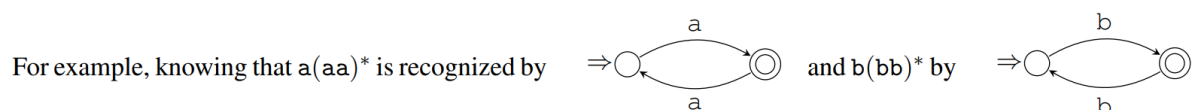
- As we saw, it suffices to construct an ε NFA
 - We remove the ε -transitions to obtain an NFA
 - We determinize to obtain a DFA.

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

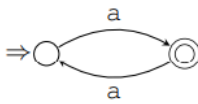
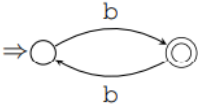
- The regex a is recognized by $\Rightarrow \text{circle} \xrightarrow{a} \text{double circle}$ and likewise if E is b or c .
- The regex ε is recognized by $\Rightarrow \text{double circle}$

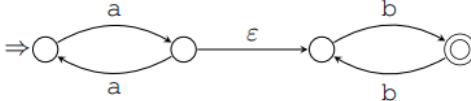


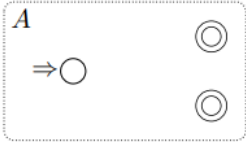
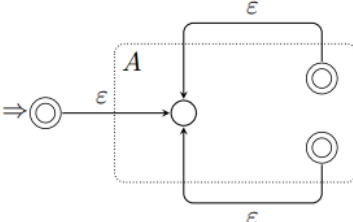
(It's essential for the sets of states to be disjoint. If necessary, renumber states to achieve this.)

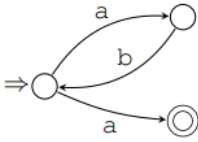


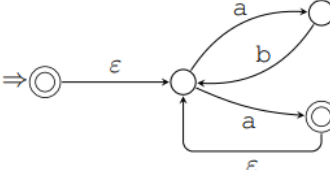
ε symbols represents a movement of state without a consumption of turn
(Again the sets of states need to be disjoint.)

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).

With E^* representing 0 or more instances of E

Proof by induction:

- 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*
 - This is the entire strong induction rule
- 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/ *strong* induction

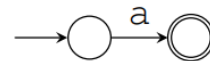
Proof by contradiction:

- 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!
-

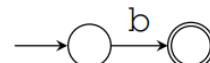
Let's follow the procedure described to convert the expression $((ab|ac)^*(abb)^*)^*$ into an automaton.

1. To save on work, we'll obtain ϵ NFAs all the way through
2. Then, right at the end, we'll remove the ϵ s and determinize.
3. To save on work we will not explicitly write name of the states for most of this process (we may consider the name of a state to coincide with its coordinates on the page).

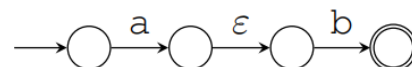
- a gives automaton



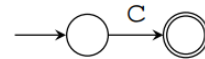
- b gives automaton



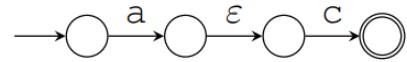
- ab gives automaton



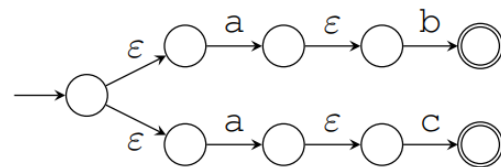
- c gives automaton



- ac gives automaton

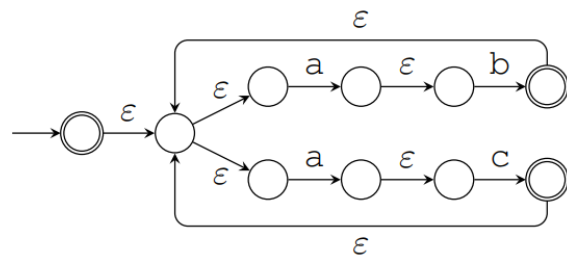


- $ab|ac$ gives automaton (with a minor simplification)

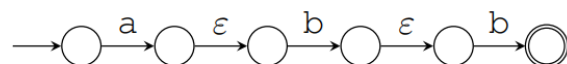


Epsilon used throughout because we are combining automata

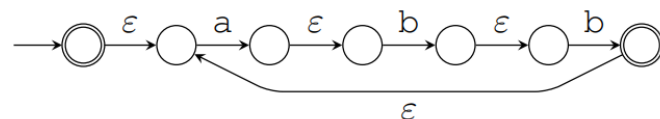
- $(ab|ac)^*$ gives automaton



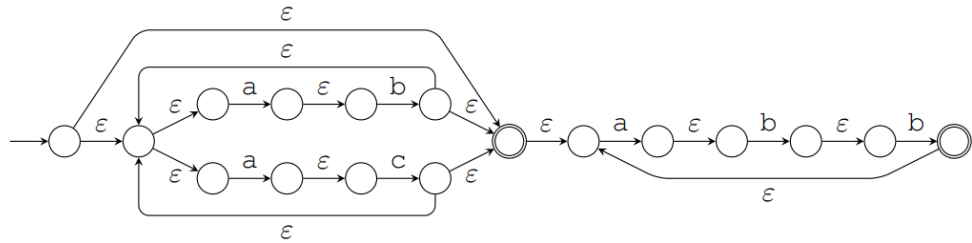
- abb gives automaton



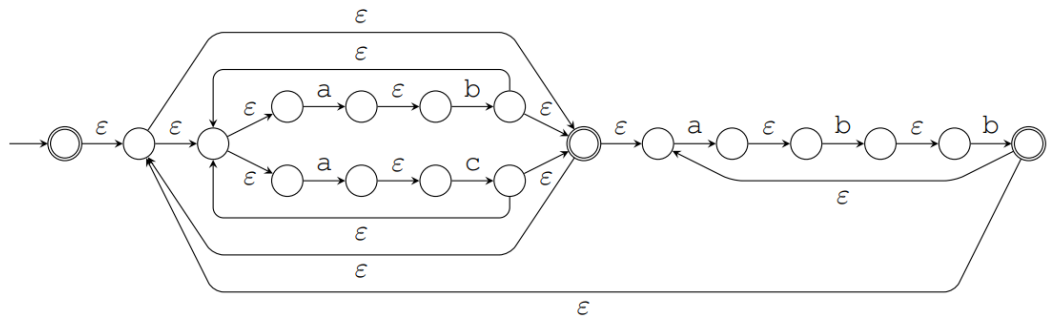
- $(abb)^*$ gives automaton



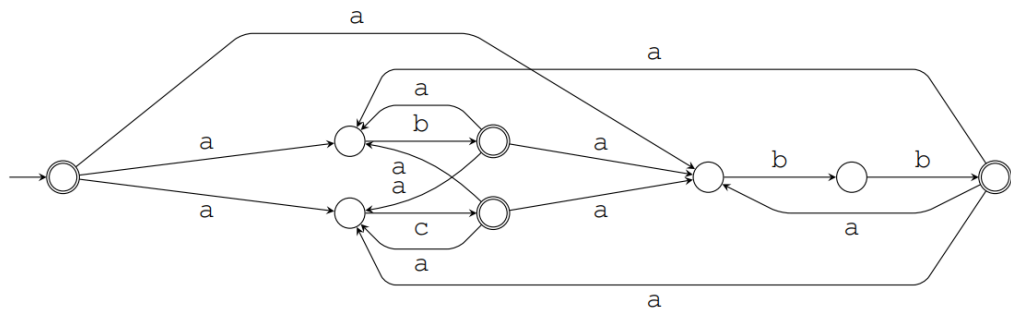
- $(ab|ac)^*(abb)^*$ gives automaton



- $((ab|ac)^*(abb)^*)^*$ gives automaton

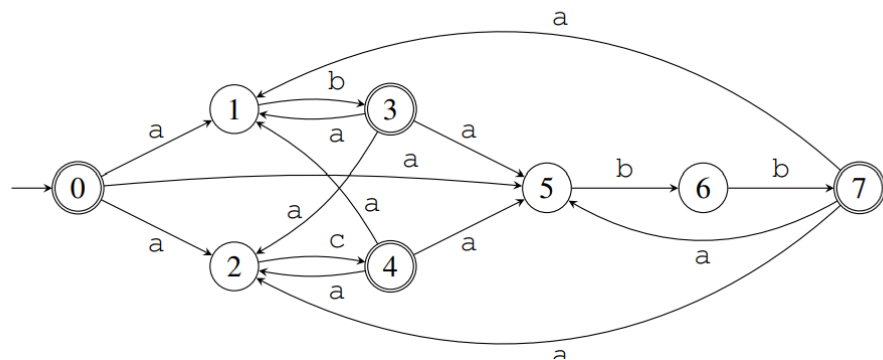


- Removing ϵ s and all states that become unreachable gives the NFA

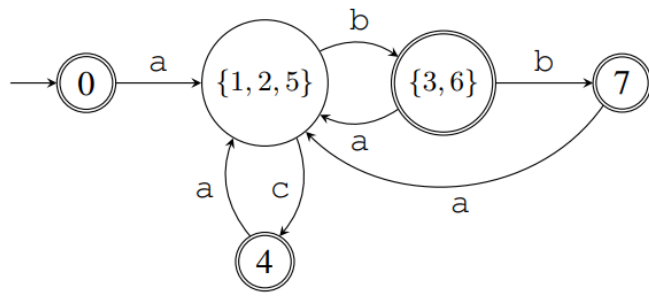


Must remove ϵ s in order to convert it to a DFA

Before we determinize, let us tidy the automaton, and name the states



- Determinization gives the partial DFA



Generalized NFAs (not examinable):

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

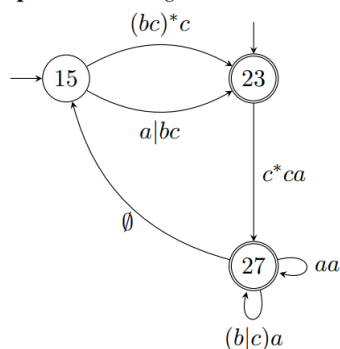
accepted.

3 Generalized NFAs (not examinable)

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?

YES

NO

cccaaa

aacca

acaca

- Also note that any \emptyset -labelled arrow can be removed without changing the language.
- Note that any ϵ NFA is also a generalized NFA.

From Generalized NFA to regex (not examinable)

- 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.

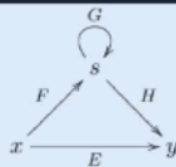
Combine any two distinct arrows $s \xrightarrow{F} t$ and $s \xrightarrow{E} t$ into an arrow $s \xrightarrow{E|F} t$. Continue until there is at most one arrow between any two states. In particular, there is at most one loop on each state.

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 an \emptyset -labelled arrow. In particular, when there is no loop on x , we insert an \emptyset -labelled loop.

3. Add in two new states: a Start state, which becomes the sole initial state, and an End state, which becomes the sole accepting state. Connect Start to each old state s by an arrow labelled with ε if s was initial, and with \emptyset otherwise. Connect each old state s to End by an arrow labelled by ε if s was accepting, and with \emptyset otherwise. Connect Start to End by an \emptyset -labelled arrow.

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 old 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 $E|FG^*H$. It's worth noting that \emptyset^* is equivalent to ε .

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