

# Module 3

## Regular languages and regular expressions

What can a computer do with no memory?

CS 360: Introduction to the Theory of Computing

Collin Roberts, University of Waterloo

3.1

### Topics of Module 3

- Regular Expressions
- Regular Languages
- Kleene's Theorem

3.2

## 1 Regular Expressions

### Regular Expressions

Recall our earlier rules for constructing new languages from existing languages:

1. *Union*  $L \cup M$  of languages  $L$  and  $M$
2. *Concatenation*  $LM$  of languages  $L$  and  $M$
3. *Closure*  $L^*$  of a language  $L$

3.3

### Regular Expressions

Let  $\Sigma$  be a finite alphabet. We construct the *regular expressions* over  $\Sigma$  (and describe the *language* which each regular expression represents, i.e. the set of words that fit into the mold defined by the regular expression) recursively, as follows.

#### Base

1.  $\emptyset$  is a regular expression, and  $L(\emptyset) = \emptyset$ .
2.  $\varepsilon$  is a regular expression, and  $L(\varepsilon) = \{\varepsilon\}$ .
3. If  $a \in \Sigma$  is any symbol, then  $a$  is a regular expression and  $L(a) = \{a\}$ .

#### Induction

1. If  $E$  and  $F$  are regular expressions, then  $E + F$  is a regular expression, and  $L(E + F) = L(E) \cup L(F)$ .
2. If  $E$  and  $F$  are regular expressions, then  $EF$  is a regular expression, and  $L(EF) = L(E)L(F)$ .
3. If  $E$  is a regular expression, then  $E^*$  is a regular expression, and  $L(E^*) = (L(E))^*$ .
4. If  $E$  is a regular expression, then  $(E)$  is a regular expression, and  $L((E)) = L(E)$ .

**Remark:** The regular expression for  $L = \{w\}$  is just  $w$  itself.

3.4

## Order of Precedence for Regular Expressions

As in algebra, there is an order of operations here:

1. *Parentheses* are used to override (or emphasize) the default order as needed.
2.  $*$
3. *Concatenation*
4.  $+$

Recommended Reading: Example 3.2 starting on p89 of the text (a regular expression over  $\Sigma = \{0, 1\}$ ).

3.5

## 2 Regular Languages

### Regular Languages

A language  $L$  is *regular* if it obeys the following recursive definition:

#### Base

1.  $L = \emptyset$  is regular.
2.  $L = \{\epsilon\}$  is regular.
3.  $L = \{a\}$  for some alphabet letter  $a \in \Sigma$  is regular.

#### Induction

1.  $L = L_1 \cup L_2$ , for regular languages  $L_1, L_2$
2.  $L = L_1 L_2$  for regular languages  $L_1, L_2$
3.  $L = L_1^*$ , for some regular language  $L_1$

No other languages are regular.

Remember that  $\emptyset \neq \{\epsilon\}$ !

3.6

### Beginnings of regular languages

First simple **Theorem:** All one-word languages  $L$  are regular.

**Proof:** Let  $L = \{w\}$ . The proof is by induction on  $|w|$ :

- Case 1:  $|w| = 0$ . Then  $L = \{\epsilon\}$ , which is regular by a rule.
- Case 2:  $|w| > 0$ . The induction hypothesis is that the Theorem is true for all one-word languages  $\{x\}$ , where  $|x| < |w|$ .
  - Let  $w = xa$ , where  $a$  is a single character.
  - Then  $L_1 = \{x\}$  is regular by the induction hypothesis.
  - And  $L_2 = \{a\}$  is regular by a rule.
  - And  $L = L_1 L_2$ , which is regular by a rule.
  - We are done.

3.7

### Finite languages are regular

**Theorem:** If  $L$  has a finite number of words, then  $L$  is regular.

**Proof:** By induction on  $|L|$ :

- Base case:  $|L| = 0$ . Then  $L = \emptyset$ , which is regular by a rule.
- Inductive case: Suppose all finite languages with  $k$  words are regular, and  $|L| = k + 1$ .
  - Then  $L = \{w_1, w_2, \dots, w_{k+1}\} = \{w_1, w_2, \dots, w_k\} \cup \{w_{k+1}\}$ .
  - By the induction hypothesis, the first language is regular.
  - By the previous Theorem, the second language is regular.
  - So  $L$  is the union of two regular languages, and thus  $L$  is regular.

3.8

## Are all languages regular?

Every language is the union of a set of one-word languages.

Are all languages regular, using the proof model just shown?

- *No!*
- Finite unions are not infinite unions! We cannot do induction in that way!
- In fact,  $\{\epsilon, 01, 0011, 000111, 00001111, 0000011111, 000000111111, \dots\}$  is not regular, which we will see in the next module.

3.9

## Regular expressions

Regular expressions are another way of representing regular languages:

$$\begin{aligned} L &= \emptyset && \emptyset \\ L &= \{\epsilon\} && \epsilon \\ L &= \{a\} && a \\ L &= L_1^* && r_1^* \\ L &= L_1 L_2 && r_1 r_2 \\ L &= L_1 \cup L_2 && r_1 + r_2 \end{aligned}$$

- (Yes, just like in `grep`. But you might have already known that.)
- If  $R$  is a regular expression, then  $L(R)$  is the language of  $R$ , defined as we did earlier.

3.10

## Examples of regular languages

- If  $\Sigma = \{0, 1\}$ , then  $\Sigma^* = \{0, 1\}^*$ . Regular expression:  $(0 + 1)^*$
- Even-length sequences: Can be divided into 2-letter sub-words.  $(00 + 11 + 01 + 10)^*$  or  $((0 + 1)(0 + 1))^*$
- Sequences with length at most 3:  $(0 + 1)(0 + 1)(0 + 1) + (0 + 1)(0 + 1) + (0 + 1) + \epsilon$  or  $(0 + 1 + \epsilon)(0 + 1 + \epsilon)(0 + 1 + \epsilon)$  or  $\epsilon + 1 + 0 + 11 + 10 + 01 + 00 + 111 + \dots$
- Sequences with at most two zeros:  $1^*(0 + \epsilon)1^*(0 + \epsilon)1^*$
- And lots more.

3.11

## Basic rules about regular languages

Basic rules that you can often use (not exciting, but true...):

- $\emptyset e = e\emptyset = \emptyset$  (If  $w \in L(\emptyset e)$ ,  $w = w_1 w_2$  where  $w_1 \in L(\emptyset)$ . But nothing works for  $w_1$ .)
- $\emptyset^* = \{\epsilon\}$  (might be surprising)
- $\{\epsilon\}^* = \{\epsilon\}$
- $x + x = x$  (remember,  $+$  means union)
- $(x^*)^* = x^*$  (Taking closure twice equals taking closure once)
  - Side Note: an operation for which applying the operation twice equals applying the operation once is called *idempotent*.
  - Other natural examples of idempotents are projections in linear algebra.
- $x(y + z) = xy + xz$

Tons more of these, but we will not focus on them. The first two might be surprising, and are thus important.

3.12

# 3 Kleene's Theorem

## Kleene's Theorem

**Theorem:** Every regular language is the language of a DFA, and every DFA accepts a regular language.

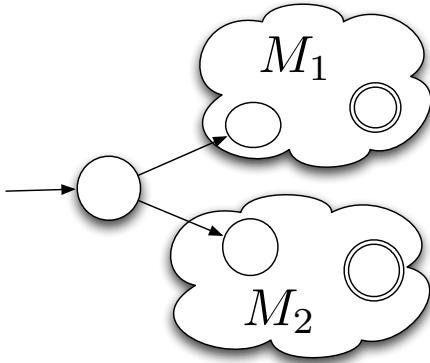
- Recall: We saw that DFAs are equally powerful to  $\epsilon$ -NFAs.
- We will show how to go from a regular language to an equivalent  $\epsilon$ -NFA.
- We will find a regular language which coincides with the language of a DFA.
- Together these constructions will prove the Theorem.

3.13

### How to accept $L_1 \cup L_2$ ?

What about for  $L_1 \cup L_2$ , if both are accepted by NFAs  $M_1$  and  $M_2$ ?

- Can we start out by going to both NFAs,  $M_1$  and  $M_2$ , and just accept in either?
- Again this requires transitions that do not use letters from the input.
- The set of accepting states in the constructed machine is the union of the sets of accept states of  $M_1$  and  $M_2$ .



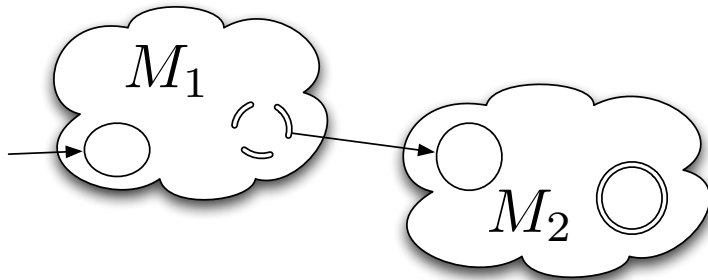
3.14

### How to accept $L_1L_2$ ?

Suppose  $L_1$  is accepted by NFA  $M_1$ , and  $L_2$  is accepted by NFA  $M_2$ .

Find: A new  $\epsilon$ -NFA  $M$  accepting  $L_1L_2$  ( $L_1$  concatenated to  $L_2$ ).

- We would kind of like:

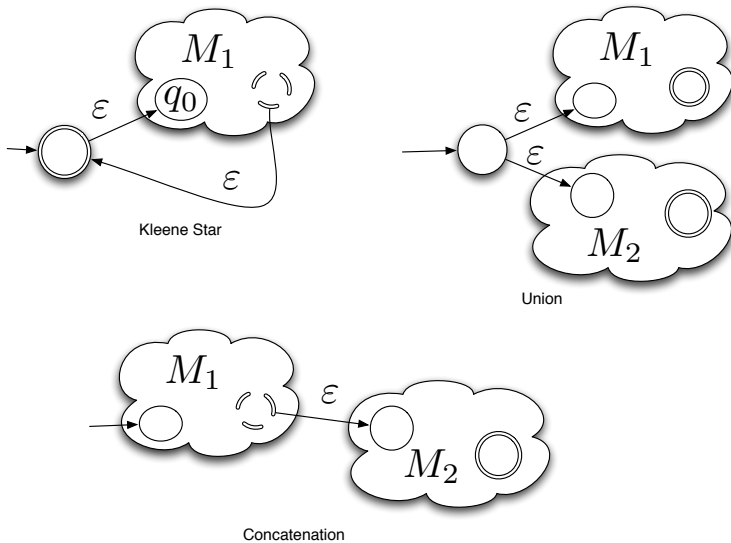


- ... where the combined machine  $M$  only accepts in  $M_2$ 's accept states
- and where the new machine  $M$  jumps from accept states of  $M_1$  to start state of  $M_2$  without reading input characters.

3.15

### Easy with an $\epsilon$ -NFA

In both cases, and for the closure operator, we can do this with  $\epsilon$ -transitions:



### Proof of the Theorem

We want to prove:

- For every regular language  $L$ , there is an  $\epsilon$ -NFA  $M$ , where  $L(M) = L$ .
- Recall:  $L$  is a regular language if  $L$  is any of:

Base

1.  $\emptyset$
2.  $\{\epsilon\}$
3.  $\{a\}$ , for some alphabet character  $a \in \Sigma$

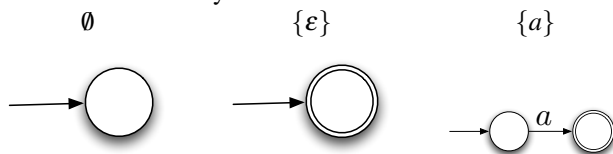
Induction

1.  $L_1 \cup L_2$  for regular languages  $L_1$  and  $L_2$ .
2.  $L_1 L_2$  for regular languages  $L_1$  and  $L_2$
3.  $L_1^*$  for regular language  $L_1$

We will show an  $\epsilon$ -NFA for the base cases, then prove the existence of the other three cases by structural induction.

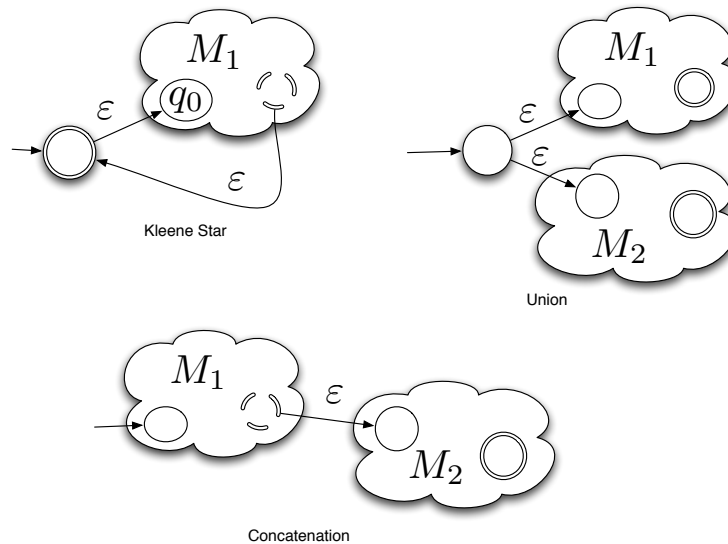
### Base cases

- The base cases are easy:



- Now, we must show the machines for the recursive definitions of regular languages.
- Assume  $L_1$  and  $L_2$  are regular languages, and that they are accepted by  $\epsilon$ -NFAs  $M_1$  and  $M_2$ , respectively.
- We must give  $\epsilon$ -NFAs for  $L_1 L_2$ ,  $L_1^*$  and  $L_1 \cup L_2$ .

## Inductive cases



But, we already showed those, right?

## We are not actually done yet

- It might seem like that is all we have to do.
- But must verify we have made the right construction, with a proof.
- One informal proof that this works, for the Kleene \* operator:
  - Consider the Kleene \* construction  $L_1^*$  for some regular language  $L_1$ .
  - Suppose  $x$  accepted by the  $\epsilon$ -NFA,  $M$ , which we have constructed as above for  $L_1^*$  (i.e. suppose  $x \in L(M)$ ).
  - Then there is a path in  $M$  for  $x$  ending in the start state.
  - This path can then be broken down into sub-paths, each of which begins and ends in the start state.
  - Each sub-path from the start state back to the start state corresponds exactly to a word from  $L_1$  ( $M_1$  accepts exactly  $L_1$ , and the only non-trivial path to the accept state of  $M$  is via an  $\epsilon$ -transition from an accept state of  $M_1$ ).
  - Therefore  $x$  is a concatenation of zero or more words in  $L_1$ , in other words  $x \in L_1^*$ . (Note that  $x$  can be  $\epsilon$ .)
  - This shows that  $L(M) \subseteq L_1^*$ .

## The other direction

Now, suppose that  $x \in L_1^*$ .

- Then  $x = x_1 x_2 x_3 \cdots x_k$ , with all  $x_i \in L_1$ .
- So there is a path from start state in the new machine to an accept state for each of the  $x_i$ .
- Join the paths together, following each with the  $\epsilon$ -transition back to the start state to get a path for  $x$  from the start state back to itself.
- The start state is an accept state, so our new machine accepts  $x$ .
- So  $x \in L(M)$ .
- This shows that  $L_1^* \subseteq L(M)$ .

Now we are done.

- We have shown that  $L(M) \subseteq L_1^*$  and  $L_1^* \subseteq L(M)$ .
- Therefore we have  $L(M) = L_1^*$ , as claimed.

The other arguments are left as exercises.

3.19

3.20

3.21

## Where are we going, again?

To prove:

- The class of regular languages is the class of languages accepted by finite automata.

We have already seen:

- From a regular language, we can produce an  $\epsilon$ -NFA which recognizes the given language.
- Languages accepted by  $\epsilon$ -NFAs = Languages accepted by DFAs (Module 2)
- Languages accepted by NFAs = Languages accepted by DFAs (also Module 2)

So given a regular language, we can produce a DFA which accepts the regular language.

Now we need to go the other way, i.e. given a DFA, produce a regular language which is the language of the DFA.

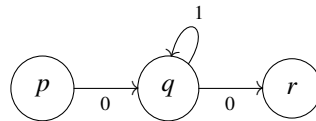
3.22

## Find a regular expression for a DFA

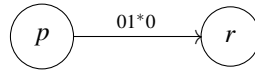
Given a DFA  $D$ , we must find a regular expression for its language.

One idea:

- Paths through a state can be replaced by the regular expression that represents going from the previous state to the next one.



- Remove  $q$ , and represent this as:



3.23

## State removal

More general:

- What about accept states and the start state?
- Make a new start state,  $s$ , with an  $\epsilon$ -transition to the old start state
- Make a new accept state,  $f$ , with an  $\epsilon$ -transition from all old accept states.

(Why? To avoid incoming/outgoing edges in start/final states.)

3.24

## Removing states

Edges in our “generalized FA” labelled with regular expressions.

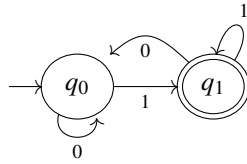
- For each state  $q$  that is not  $s$  or  $f$ :
  - For all state pairs  $(p, r)$  that “flank”  $q$ : ( $p$  could be the same as  $r$ , but  $q$  cannot equal  $p$  or  $r$ )
    - \* Let  $e_1$  = label of edge from  $p$  to  $q$
    - \* Let  $e_2$  = label of edge from  $q$  to  $r$
    - \* Let  $e_3$  = label of loop from  $q$  to itself (or  $\emptyset$  if there is not one)
    - \* Let  $e_4$  = label of edge from  $p$  to  $r$  (or  $\emptyset$  if there is not one).
    - \* Make an edge from  $p$  to  $r$  with label  $e_4 + e_1 e_3^* e_2$ . Remember:  $\emptyset^* = \epsilon$  for our purposes here .
  - Then remove the node  $q$ .

At the end of the process, we will have a single edge, from  $s$  to  $f$ .

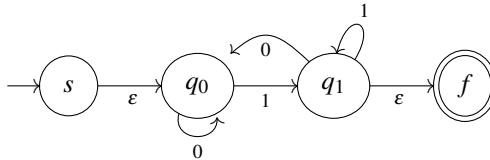
3.25

### And the label is the expression

- At the end of state elimination, the edge from  $s$  to  $f$  is labelled with a regular expression for the DFA's language.
- Let's do an example.



- First, make states  $s$  and  $f$ :

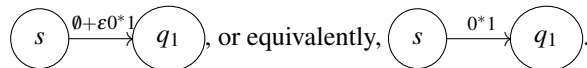


3.26

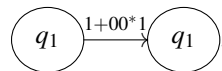
### State elimination example

Now, eliminate state  $q_0$ .

- Consider all possible paths through  $q_0$ .
- The possible choices of flanking pairs for  $q_0$  in this example are
  - $(s, q_1)$ : For this pair, we have  $e_1 = \epsilon, e_2 = 1, e_3 = 0, e_4 = \emptyset$ , which means our new diagram must include



- $(q_1, q_1)$ : For this pair, we have  $e_1 = 0, e_2 = 1, e_3 = 0, e_4 = 1$ , which means our new diagram must include

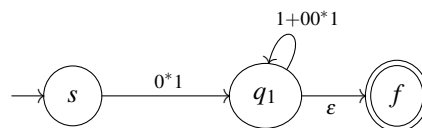


3.27

### State elimination example

Now, finish eliminating state  $q_0$ .

- We add an edge from  $s$  to  $q_1$ , and change the label of the self-loop on  $q_1$ :

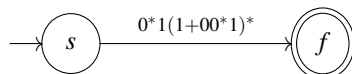


3.28

### End of example

Now, eliminate state  $q_1$  in the same way.

- Paths from  $s$  to  $f$  via  $q_1$  require a word in  $0^*1$ , followed by any number of words in  $(1 + 00^*1)$ , followed by  $\epsilon$ .
- This gives this new FA:



And the language of the original FA is  $L(0^*1(1 + 00^*1)^*)$ .

*Note:* this may not be the simplest possible form for this regular expression!  
(What is? How long an expression can this generate?)

3.29



### This is not a proof!

No, it is not. We can give a formal proof for state removal, but we will not.

We will give a formal proof, but with a different way of showing that  $L(M)$  is regular, where  $M$  is an arbitrary DFA.

Recall:

- $L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in F\} = \bigcup_{r \in F} \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = r\}$
- Characterize all strings that take us from some state  $q$  to  $r$ : Define  $L(q, r) = \{x \in \Sigma^* \mid \hat{\delta}(q, x) = r\}$ .
- With this definition,  $L(M) = \bigcup_{r \in F} L(q_0, r)$ .
- This is the set of words that take us from  $q_0$  to any accept state  $r$ .

3.30

### Proving this language is regular

- $L(M) = \bigcup_{r \in F} L(q_0, r)$ .
- This is a finite union, so if all  $L(q_0, r)$  are regular, then so is  $L(M)$ . (Prove this, by induction on  $|F|$ !)

What we now want:

- **Theorem:** for any two states  $q$  and  $r$  in a DFA,  $L(q, r)$  is regular.
- We will prove this using structural induction.
- How do we get from  $q$  to  $r$ ?
- Do we use an intermediate state  $p$ ?
- If so, we go from  $q$  to  $p$ , maybe from  $p$  to itself, then from  $p$  to  $r$ .
- One term of the regular expression for  $L(q, r)$  might be  $L(q, p)L(p, p)^*L(p, r)$ .

But structural induction needs a *base case*!

[Remember: structural induction goes from “simple” structures to “complex”]

3.31

### New definitions

How can we make the base case simple enough?

- Restrict the number of states that can come between  $q$  and  $r$ , and grow this set of states.
- Number the states of the DFA  $M$   $1, \dots, n$ .
- Let  $L(q, r, k) = \{ \text{all words in } L(q, r) \text{ where all } \textit{intermediate} \text{ states between } q \text{ and } r \text{ are from } 1, \dots, k \}$ . (Remember,  $M$  is a DFA, so each word has only one state path.)

3.32

### More about the languages $L(q, r, k)$

- If the path from  $q$  to  $r$  for word  $x$  uses state  $k$ , then  $x$  is not in  $L(q, r, k-1)$ , or  $L(q, r, k-2)$ , or any other  $L(q, r, i)$  for  $i < k$ .
- Formally:  $L(q, r, k) = \{x \in \Sigma^* \mid \hat{\delta}(q, x) = r \text{ and } \hat{\delta}(q, w) \leq k \text{ for all proper prefixes } w \text{ of } x \text{ where } w \neq \varepsilon \text{ and } w \neq x\}$ .
- (Terminology:  $w$  is a *proper* prefix of  $x$  if  $w$  is a prefix of  $x$  and not equal to  $x$ .)
- Now it is enough to show  $L(q, r, n)$  is regular, since  $L(q, r) = L(q, r, n)$ .
- Then it is enough show that  $L(q, r, k)$  is regular for all  $k$ .
- The proof is by induction on  $k$ .

3.33

### Goal, beginning of proof

Goal:  $L(q, r, k)$  is regular for all  $k$ : proof by induction on  $k$ .

Base case:  $k = 0$ :

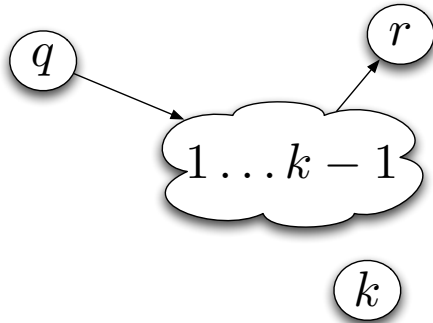
- If  $x \in L(q, r, 0)$ , then  $\hat{\delta}(q, w) \leq 0$  for all proper prefixes  $w$  of  $x$ .
- But that means there are no proper prefixes of the word  $x$ .
- Therefore,  $|x| = 0$  or  $1$ .
- So all words in  $L(q, r, 0)$  are of length 0 or 1.
- As our alphabet is finite, this implies  $L(q, r, 0)$  is finite (and thus regular by an earlier Theorem).

3.34

### Inductive step

Induction step:  $k = n \geq 1$ :

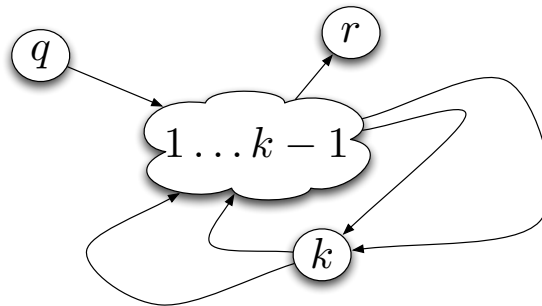
- Inductive hypothesis:  $L(q, r, k - 1)$  is regular for all  $q$  and  $r$ .
- We must show that  $L(q, r, k)$  is regular.
  - Let  $x$  be a word in  $L(q, r, k)$ .
  - If the path from  $q$  to  $r$  for  $x$  never touches the state  $k$ , then  $x \in L(q, r, k - 1)$ .



3.35

### Inductive step, continued

- Otherwise:  $k$  is on the path from  $q$  to  $r$  for  $x$ .
- Then there is a first and a last time we are in state  $k$ . Between the first and last time, we are only



in states  $1, \dots, k$ , regardless of whether  $q > k$  or  $r > k$ .

3.36

### Inductive step, finished

So we can divide  $x$  into:

- The part from  $q$  to  $k$  the first time,
- The first loop (if any) from  $k$  to  $k$ ,
- The next loop from  $k$  to  $k$ ,
- ...
- The last loop from  $k$  to  $k$ ,
- and the part from  $k$  to  $r$ .

Words divided in this way are exactly the words in  $L(q, r, k)$  that include state  $k$  on their state path.

This set of words is  $L(q, k, k - 1)(L(k, k, k - 1))^*L(k, r, k - 1)$ .

So the language  $L(q, r, k)$  is the union of two languages:

- $L(q, r, k - 1)$ , which we know is regular by the induction hypothesis, and
- $L(q, k, k - 1)(L(k, k, k - 1))^*L(k, r, k - 1)$ , which is the concatenation of three languages (each of which is regular by the induction hypothesis), and thus is also regular.

Hence  $L(q, r, k)$  is regular.

3.37

## Wrapping it up

We are done, but that may not be obvious yet.

- We proved that  $L(q, r, k)$  is regular for all  $k$ .
- We noted that  $L(q, r) = L(q, r, n)$ , so  $L(q, r)$  is always regular for any  $q, r \in Q$ .
- Therefore  $L(q_0, r)$  is regular for any  $r \in F$ .
- Thus  $\bigcup_{r \in F} L(q_0, r)$  is regular (as it is the union of a finite number of regular languages).
- But  $L(M) = \bigcup_{r \in F} L(q_0, r)$  is therefore regular.

So  $L(M)$  is regular! (Again, how long is the expression for  $L(q_0, r)$ ?)

3.38

## Quite an achievement

This is the end of the proof of Kleene's Theorem:

- Given a DFA, we have shown that its language is regular.
- Given a regular language, we can produce an  $\varepsilon$ -NFA which recognizes it.
- NFAs and  $\varepsilon$ -NFAs have the same computing power as DFAs.

Next module: the boundaries of regular languages, and closure rules for them.

3.39