Languages

Simon Niklaus

a language is a set of strings

- can be finite
- can be infinite
- can be empty

- but what is a string?

a string is a sequence of symbols

- always finite
- can be empty

- but what are symbols?

an alphabet defines a set of symbols

- mostly finite
- mostly nonempty
- written as Σ or Γ

- can we have an example?
  - $\Sigma = \{a, b, c\}$

a string is therefore defined over an alphabet of symbols

- can we have an example?
  - $accbcca$
- the empty string is special case
  - it is written as ε

there are several operations defined for string

- length
  - $|accbcaa| = 7$
  - $|\varepsilon| = 0$
- concatenation
  - concatenating a and b yields ab
- reverse
  - $accb^R = bcca$
- many more

and we can now have a look at some languages

- $\{a, b, c, ab, ac, ba\}$
- $\{a\}$
- $\{\ \} = \emptyset \neq \{\varepsilon\}$

what are some possibilities to define a language?

- enumeration – what we already did
- regular expressions – we learn about that tomorrow
- automatas – we learn about that today
- grammars – you will learn about that later on in your graduate program – hopefully from me, because i love grammars
- set notation - $\{x \mid x \text{ begins with an } a\}$
- many more

a note on languages in Schaum's outline

- he talks about operations on languages
- you actually need to know more about a language, in order to be able to perform operations on
- negating a language is therefore valid for some languages, while it is invalid for others
- this is why i will not talk about operations on languages yet

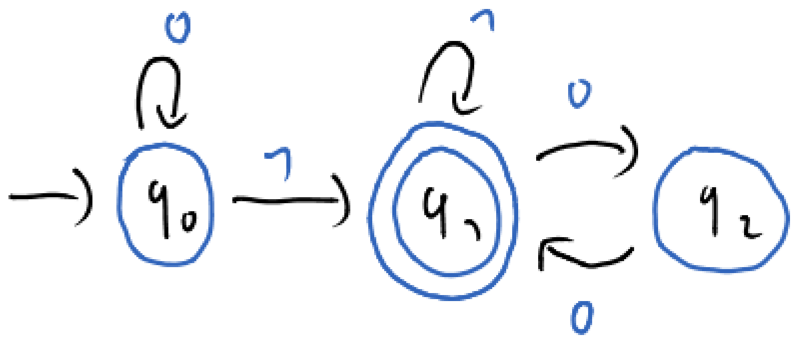
Regular Languages

Simon Niklaus

let us talk about finite state machines

- we have to talk about finite state machines, before we can define regular languages
- finite state machine and finite automata are by the way synonyms
- finite automata are like small computers, with limited memory – they are usually qu

an example without furhter explanation



- such an automaton can be used, in order to
  - generate / accept strings
  - generate / recognize languages

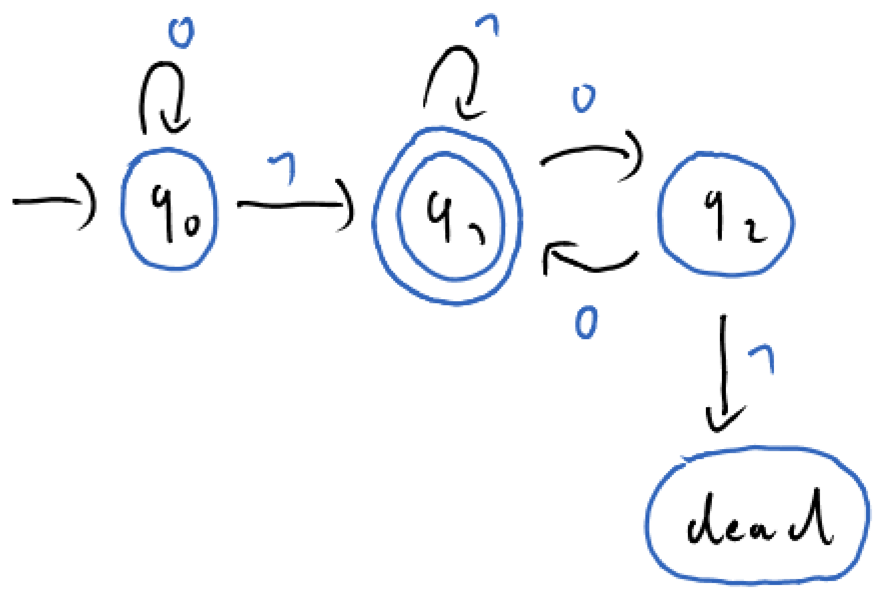
the formal description of a finite automata as a 5 tupel

- $M = (Q, \Sigma, \delta, q_0, F)$

- Q – a set of states – finite
- Σ – the alphabet / a set of symbols – finite with $\varepsilon \notin \Sigma$
- δ – a transition function – $Q \times \Sigma \rightarrow Q$
- q_0 – a starting state – with $q_0 \in Q$
- F – a set of accepting / final states – with $F \subseteq Q$

but wait, the example is not complete?

- computer scientist are lazy, so we handle the transition function quite loosely
- missing transitions are therefore defined, to just end in a dead state – so they will never end in an accpeting state



notation - $L(M)$

- defines the language, that is being recognized by M
- it therefore stands for the set of strings, that is being accepted by M

let us formalize the way finite state machines work

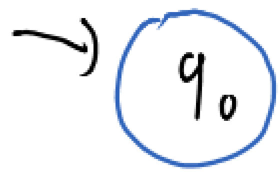
- let M be a finite state machine
- let $w = w_1 \dots w_n$ be a string where $w_i \in \Sigma$
- M accepts w, if there is a sequence of states $r_0 \dots r_n$ where $r_i \in Q$, such that
  - $r_0 = q_0$
  - $\delta(r_i, w_{i+1}) = r_{i+1}$
  - $r_n \in F$

definition – regular language

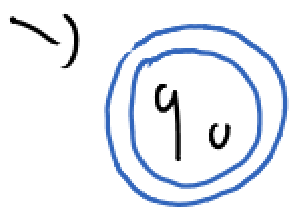
- a language is a regular language, iff some finite state machine recognizes it
- note the term iff
- both directions are therefore valid

some examples of corner cases

- { } - the empty language



- {ε} - the language containing the empty string



now it is time for you to exersice

- $\Sigma = \{a, b\}$

- $L = \{w \mid w = aba\}$
- $L = \{w \mid w = aba$ or $w = aaa\}$
- $L = \{w \mid w$ does contain aba in it$\}$
- $L = \{w \mid w$ does not contain $aabb$ in it$\}$
- $L = \{w \mid w$ contains an odd number of a's and an odd

okay, the last one is not a regular language

- the reason is, that finite state machines do not have a memory
- but how can we actually prove that a language is not regular?
- with the pumping lemma for regular languages, but you are going to learn about that later on in your graduate program

finite state machines can describe real problems

- design for example a finite state machine, that reads a binary number from the msb to the lsb and decides, whether the number is divisble by 3

- msb = most significant bit
- lsb = least significant bit

- $\Sigma = \{0,1\}$
- $L = \{0,11,110,1001, \dots\}$

finite automatas are closed under certain operations

- union – $L_1 \cup L_2$
- concatenation – $L_1 \circ L_2$
- kleene / star – L^*

but what does closed actually mean?

- if L / L_1 / L_2 are regular languages, their union / con-catenation / kleene will also be a regular language
- we are going to prove that on the next set of slides

can we have an example of these operations?

- $\Sigma = \{a, b, c, d\}$
- $A = \{aa, b\}$
- $B = \{cc, d\}$

- $A \cup B = \{aa, b, cc, d\}$
- $A \circ B = \{aacc, aad, bcc, bd\}$
- $A^* = \{\varepsilon, aa, b, aab, baa, bb, aaaa, \dots\}$

Nondeterminism

Simon Niklaus

our finite state machines so far were deterministic

- given a current state and the next symbol from the input, we knew exaclty what to do
- there were no choices or any form of randomness
- this is how computers actually should be
- let us introduce nondeterminism, what behaves slightly different

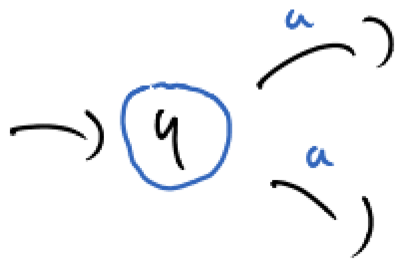
terminology

- DFA – deterministic finite state automaton
- NFA – nondeterministic finite state automaton

- we are going to talk about NFAs in this part of the lecture, as they are nondeterministic – contrary to what we have had so far

what are we going to allow from now on?

- multiple edges that go out of a state, that have the same label



- epsilon edges

let us formalize our 5 tupel again for NFAs

- $M = (Q, \Sigma, \delta, q_0, F)$

- Q – a set of states – finite
- Σ – the alphabet / a set of symbols – finite with $\varepsilon \notin \Sigma$
- δ – a transition function – $Q \times \Sigma_\varepsilon \to P(Q)$
- q_0 – a starting state – with $q_0 \in Q$
- F – a set of accepting / final states – with $F \subseteq Q$

- note, that the only difference to DFAs lies within the transition function

how about the acceptance of strings with NFAs?

- let M be a NFA
- let $w = w_1 \ldots w_n$ be a string where $w_i \in \Sigma$
- M accepts w, if there is a sequence of states $r_0 \ldots r_n$ where $r_i \in Q$, such that
  - $r_0 = q_0$
  - $r_{i+1} \in \delta(r_i, w_{i+1})$
  - $r_n \in F$
- in other words – the NFA accepts, if there is at least one valid path that ends in a final state

one first exercise to get cofortable with NFAs

- $\Sigma = \{a, b\}$
- $L = \{w \mid w \text{ ends with two } b\text{'s}\}$
- note, that is usually easier to come up with an NFA, than it is to come up with a DFA – NFAs are also usually more compact

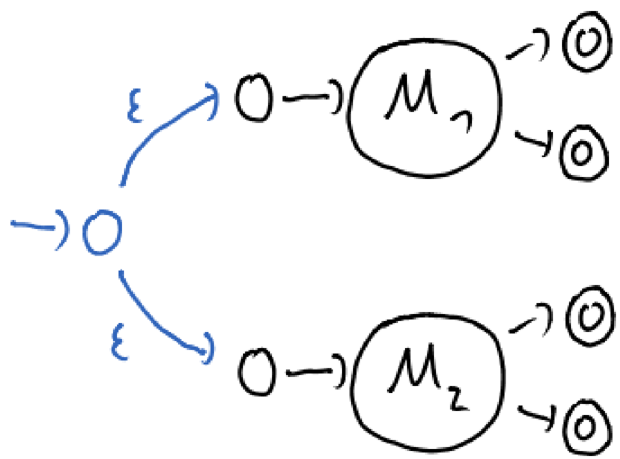
but are NFAs more powerful then DFAs?

- to clarify it a little further – is it possible to define languages beyond that are more advanced than regular languages?
- nope, sorry
- you are going to see a proof by construction later in your graduate program, that converts any NFA into an equivalent DFA
- DFA ↔ NFA

so we were talking about the term closed earlier

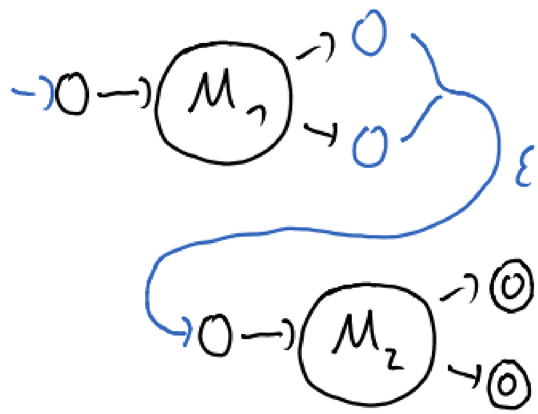
- since we now know, that DFAs and NFAs are equivalent, we can use the nondeterminism to prove the earlier statement

- union

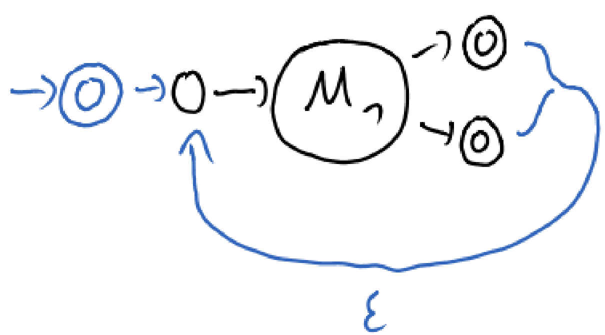


so we were talking about the term closed earlier

- concatenation



- kleene / star



would anyone like to have more exercises?

- $L = \{w \mid w = 0\}$ – solve this with exaclty three states
- $L = \{w \mid w$ contains an even number of a's or exactly