Turing Machines

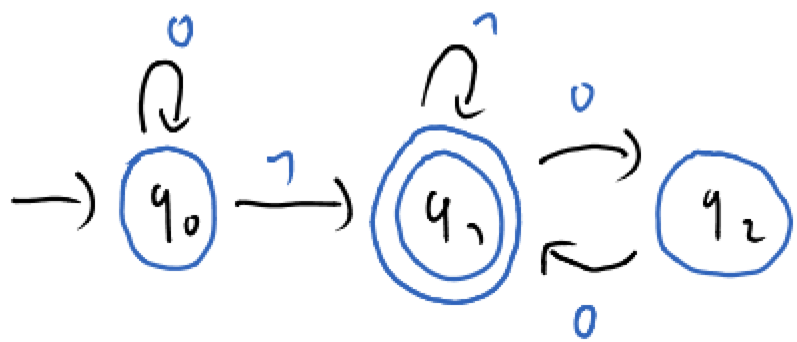
Simon Niklaus

recap – alphabets, strings and languages

- an alphabet defines a set of symbols
  - $\Sigma = \{a, b, c\}$
- a string is a sequence of symbols
  - $accbcca$
- a language is a set of strings
  - $\{a, b, c, ab, ac, ba\}$

recap – finite automata

- what is a finite state machine again?



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

recap – DFAs

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

- 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 $F \subseteq Q$

recap – 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

recap – exercises

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

- $L = \{w \mid w = aba\}$
- $L = \{w \mid w = aba$ 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

recap – exercises

- the last exercise is not a regular language – to prove this, the pumping lemma for regular languages can be applied
- it is sufficient for us though, to realize that finite automata do not have memory and the given language would require some sort of memory

chomsky hierarchy

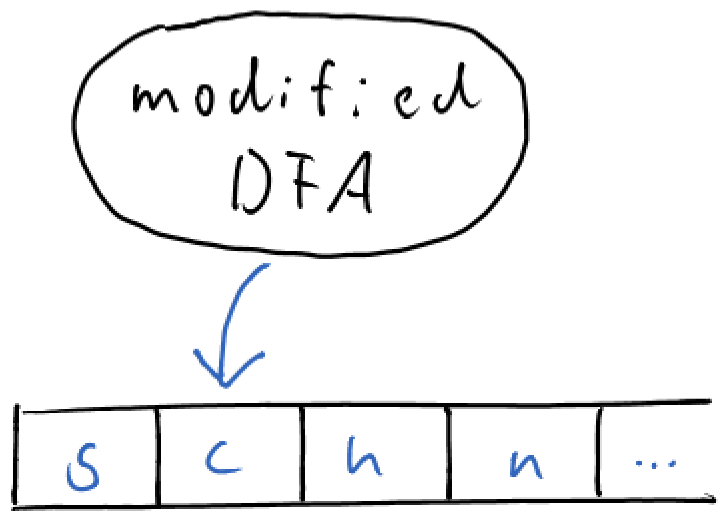
- type 3 – regular
  - finite state automata
  - no memory / only a history, finite
  - $\{w \mid w$ is of the form $a^n b$ with $n \geq 0\}$
- type 2 – context-free
  - nondeterministic pushdown automata
  - stack, infinite
  - $\{w \mid w$ is of the form $a^n b^n$ with $n \geq 0\}$

chomsky hierarchy

- type 1 – context-sensitive
  - linear bounded nondeterministic turing machines
  - tape, linear bounded
  - $\{w \mid w$ is of the form $a^n b^n c^n$ with $n \geq 0\}$
- type 0 – recursively enumerable
  - turing machines
  - tape, infinite
  - $\{w \mid w \text{ is a prime number}\}$

note: $\forall q \exists p \forall x \forall y [((q < p) \land (x > 1) \land (y > 1)) \rightarrow (x * y \neq p)]$ where $q, p, x, y \in \mathbb{N}$

an outline to start with



- there are variations within the definition from textbook to textkbook – these variations are all equivalent

characteristics of the finite automaton

- we are going to start with deterministic turing machines – the controlling finite automaton is therefore chosen to be deterministic
- there is only one accept state and furthermore an additional reject state – the latter one is similar to a dead state
  - both of these states take effect immediately
  - this property will be useful with nondeterminism
- the head of the turing machine initially points at the leftmost cell

characteristics ofthe tape

- it is infinite, even though it has a lower bound – every unused cell contains the blank symbol ⊔
- we can use each transition of the controlling finite automaton to read and write to the tape – the head can furthermore be moved to the left or to the right
- when working with finite state machines, we implicitly had an input string that we have read successively
  - turing machines do not have an input stream
  - we therefore place the input string on the tape initially

uses of turing machines

- we used finite automata in order to recognize languages
  - turing machines can be used for the same task
  - they are capable to recognize a larger set of langauges though
- we can however access and interpret the content of the tape after the turing machine has finished
  - a turing machine can therefore be used to do actual computations
  - the output is simply placed on the tape, from which it can be retrieved afterwards

the formal description of a turing machine as a 7 tupel

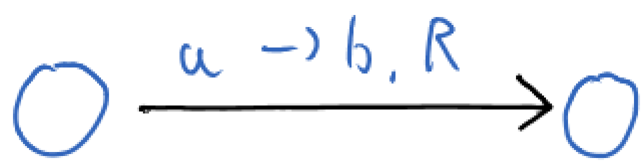
- $M = \left(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\right)$

- Q – a set of states – finite
- Σ – the input alphabet – finite with $\sqcup \notin \Sigma$ and $\varepsilon \notin \Sigma$
- Γ – the tape alphabet – finite with $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$
- δ – a transition function – $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathcal{L}, \mathcal{R}\}$
- q_0 – a start state – with $q_0 \in Q$
- q_{accept} – a accept state – with $q_{acc} \in Q$ and $q_{acc} \neq q_{rej}$
- q_{reject} – a reject state – with $q_{rej} \in Q$ and $q_{rej} \neq q_{acc}$

note: the tranisition function is deterministic

the new transition function

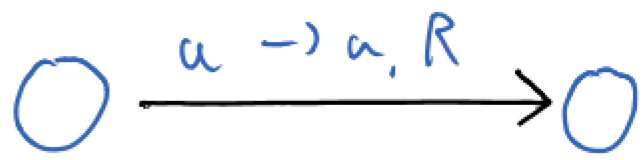
- given a current state of the controlling finite automaton, we read a symbol from the current cell
- we then write a symbol to the same cell and move either left or right afterwards



- what if we are on the left side of the tape and move the head left again – the head will stay at the leftmost cell

the new transition function

- since the definition forces us to always write a symbol to the tape, how do we leave a cell unchanged – we basically write the same symbol again



- since this is a quite common problem, we use a shorthand for that and simply omit the symbol on the right hand side of the transition

note: not wanting to move the head would be useless

possible outcomes of a computation

- the turing machine can either
  - halt and accept
  - halt and reject
  - loop
- note that it is impossible to loop with a finite state machine on a given input string, because strings are finite in length
- you might ask why a turing machine might loop, but this is actually an important property, that enables entirely new possibilities

let us do some obvious exercises to start with

- $\Sigma = \{a, b\}$
- $L = \{w \mid w = aba\}$
- $L = \{w \mid w = aba$ 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

back to the chomsky hierarchy

- from the chomsky hierarchy
  - type 3 – regular
  - type 0 – recursively enumerable
- a finite state machine is basically just a turing machine that does not write to the tape and always moves one step to the right
- regular languages are therefore a subset of recursively enumerable languages

and now something more advanced

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

- $L = \{w \mid w$ is of the form $a^n b^n$ with $n \geq 0\}$
- $L = \{w \mid w$ is of the form $a^n b^n c^n$ with $n \geq 0\}$

- since these languages are not regular, regular languages are a proper subest of recursively enumerable languages

describing turing machines on a higher level

- describing the controlling finite automaton is a rather dull and inconvenient task
- equivalent to programming languages, we can introduce a layer of abstraction by describing an algorithm on a higher level
    - there is no specific notion of such languages for turing machines
    - a higher level description is therefore sufficient, once it is convincing enough

let us practice this higher level descriptions

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

- $L = \{w \mid w$ contains an equal number of a's and b's$\}$
- $L = \{w \mid w$ contains twice as many a's as b's$\}$
- $L = \{w \mid w$ does not contain as many a's as b's$\}$

a possible solution for the first language

1. start by shifting everything on the tape one cell to the right and placing a special symbol into the leftmost cell
   - we are therefore able to detect the lower bound of our tape
2. scan for an a from the lower to the upper bound of the tape
   - if an a has been found, cross it out and go to 3
   - if no a has been found, go to 4
3. scan for a b from the lower to the upper bound of the tape
   - if a b has been found, cross it out and go to 2
   - If no b has been found, reject
4. scan for a b from the lower to the upper bound of the tape
   - if a b has been found, fail
   - if no b has been found, accept

definition – configurations

- a configuration represents the entire state of a turing machine – it is therefore basically a snapshot
- what we have to store in a configuration
  - the content of the tape
  - the state of the controlling finite automaton
  - the current position of the head
- this can be done with a single string, where the state is simply inserted in front of the currently located cell
  - $a\ b\ q_7\ c\ d$

note: obviously, Q and Γ have to be disjoint

definition – computation histories

- a sequence of configurations represent a computation history, if
  - the sequence starts with the start configuration
  - there were only legal transitions between two consecutive configurations
  - the sequence ends with an accepting or rejecting configuration
- note that this model of computation is very similar to the computational model of finite automata

extending the chomsky hierarchy

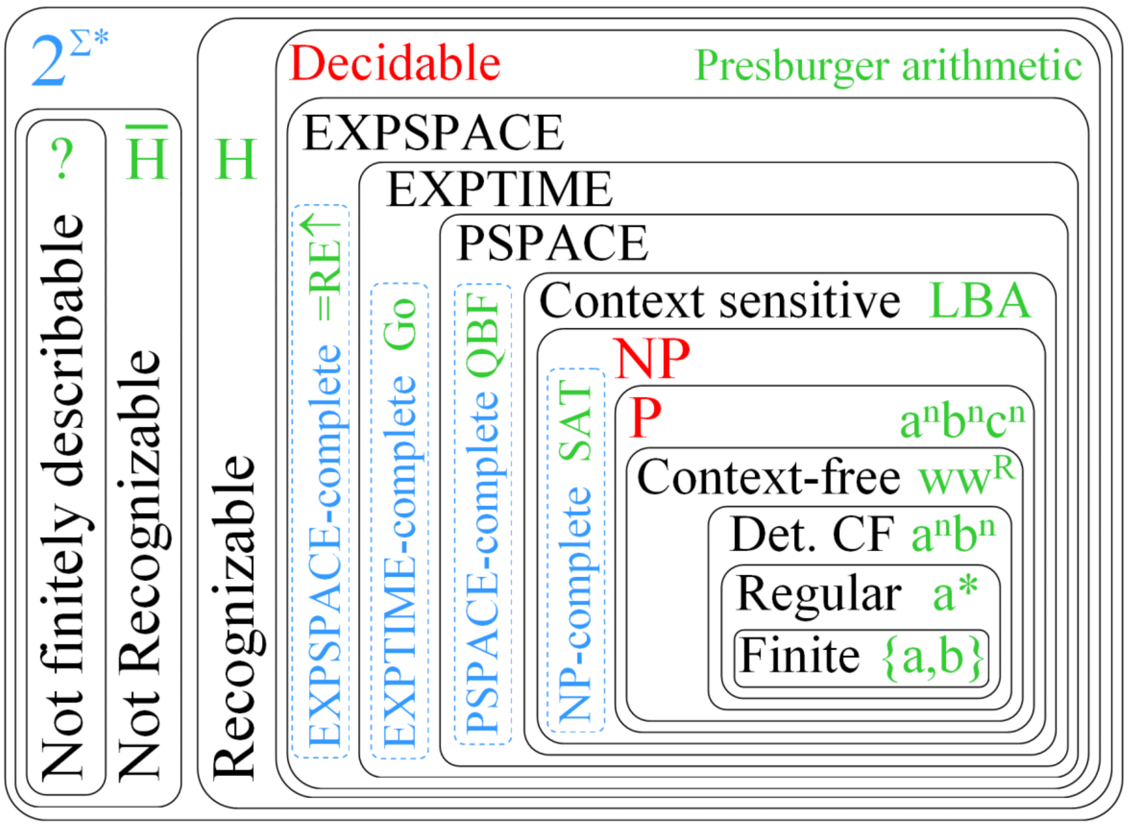
- languages that are recursive / decidable
  - given an input string, the turing machine will always halt
  - it will therefore either end in an accepting or rejecting state and therefore accept or reject the input string
- languages that are recursively enumerable / turing recognizable / semi decidable
  - given an input string that is in the language of the turing machine, the turing machine will always halt and accept
  - given an input string that is not in the language, the turing machine will either reject or loop

extending the chomsky hierarchy

- languages that are not recursively enumerable / not turing recognizable
  - the turing machine does not even reliably halt for input strings that are within the language of the turing machine
  - it is quite hard to imagine such a language, but proving its existance is rather easy – we are going to do this in a moment

note: semi decidable is already not decidable

extending the chomsky hierarchy



source: http://www.cs.virginia.edu/~robins/cs6160/

the acceptance problem for turing machine

- given a turing machine, is it possible to decide whether this turing machine halts on a certain input
  - of course not, otherwise every turing machine could be converted into one that always halts
  - in order to prove that it is recognizable, we have to present a turing machine that recognizes this language
  - this can easily be done by simply emulating this turing machine on our turing machine
- this language is commonly defined and used
  - $A_{TM} = \{\langle M, w\rangle \mid M$ is a turing machine and M accepts $w\}$

the acceptance problem for turing machine

- since we know that A_{TM} is turing recognizable, $\overline{A_{TM}}$ has to be not turing recognizable
  - otherwise we could simultaneously start the turing machine for both languages on a given input string
  - per definition of turing recognizable languages, one of them would have to halt
  - we could therefore decide A_{TM}

variants of turing machines

- as already mentioned, there are variations within the definition of touring machines from textboox to textbook – these variants are all equivalent
- what about a tape that has no lower bound
  - every time we detect that we are at the leftmost cell and want to go further left, we shift the content of the tape to the right
- but how do we detect the left hand side of the tape
  - we initially shift the whole tape at the beginning of the computation to the right and place a special symbol into the leftmost cell

variants of turing machines

- what about a multitape turing machine – the transition function therefore has to be extended
  - it is quite exhausting to prove this
  - the tapes are concatenated and stored on a single tape, separated by a special symbol
  - the position of each head in each subtape is preserved through a marking mechanism of symbols
  - in order to perform a transition, all the subtapes have to be scanned and updated, once the appropriate transition has been found
  - whenever a head moves off the rightmost cell of a subtape, the following tapes have to be shifted

note: i do not expect you to be able to reproduce this proof

turing completeness

- the turing machine is the most universal automaton that we know so far
- if a computational model is able to simulate a turing machine, this model is turing complete
- this is quite easy to achive though, since mov for example is already turing complete

source: www.cl.cam.ac.uk/~sd601/papers/mov.pdf

nondeterministic turing machines

- $M = \left(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\right)$

- Q – a set of states – finite
- Σ – the input alphabet – finite with $\sqcup \notin \Sigma$ and $\varepsilon \notin \Sigma$
- Γ – the tape alphabet – finite with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
- δ – a transition function – $Q \times \Gamma \rightarrow P(Q \ \times \Gamma \ \times \{\mathcal{L}, \mathcal{R}\})$
- q_0 – a start state – with $q_0 \in Q$
- q_{accept} – a accept state – with $q_{acc} \in Q$ and $q_{acc} \neq q_{rej}$
- q_{reject} – a reject state – with $q_{rej} \in Q$ and $q_{rej} \neq q_{acc}$

note: only the transition function has changed

configurations and computation histories

- at any given point, a configuration of a nondeterministic turing machine could have more than one successor
- the computation historie is therefore not necessarily a sequence of configurations any more
  - it instead resembles a tree
  - this tree therefore contains all possible choices that are implied through the nondeterminisim

possible outcomes of a computation

- the nondeterministic turing machine can either
  - halt and accept, if any branch of the computation history accepts
  - halt and reject, if all branches of the computation history reject
  - loop, if no accepting configuration has occured that and there are still ongoing branches within the history
- note, that these three possibilities are the same as for deterministic turing machines – they are just defined

is a nondeterministic turing machine more powerful

- of course not, otherwise we would have a problem with the definition of the turing completeness
  - we can simulate a nondeterministic turing machine on a deterministic one, by emulating the computational history
  - we therefore have to do a state based search
  - a depth first search wo be simple, but would fail to perform the task, because we could end up in an branch that never halts
  - a breath first search has therefore been done, which is rather complex – the usual proof for example already requires a deterministic turing machine with three tapes

note: i do not expect you to be able to reproduce this proof

i hope you had fun, at least i had

Simon Niklaus