Logic and theory of computation

theory of computation part, 1st lecture

Introduction

Algorithmic solution

Given a problem in real life our first task is to formulate it in the language of mathematics.

In most of the cases we are looking for an **algorithmic solution**, a solution that is working even if we change the input parameters.

According to the possible answers there are **decision problems** (yes/no answer) and **function problems** (arbritary type of answers, e.g., an integer type).

Sometimes we find an efficient algorithm, sometimes we do not.

Question 1: What is considered to be efficient?

Question 2: Is there always a solution?

Introduction

What algorithms are considered to be efficient?

Given 5 algorithms for a problem of input size n using $\log_2 n$, n, n^2, n^3, 2^n resources (e.g., time, space) respectively.
We can use up to K resources. Doubling the speed of our computer results in what increment in the maximum input size?
It is a squaring, doubling, $\sqrt{2}$-ing, $\sqrt[3]{2}$-ing the input size in the first 4 cases, while only +1 in the 5th case..

So at most polynomial algorithms are consider to be efficient, while others are considered to be non-efficient.

Remark 1: For space complexity, many times constant or logarithmic algothm is required.

Remark 2: For practical reasons it is possible that a $2^{n/100}$ algorthm outperforms a n^{80} algorithm. It is possible, too, that an exponential algorithm with polynomial expected value outperforms a polynomial one.

Introduction

Example problems

1. 12322+4566=? *Generally?*
   algorithmic solution: see elementary school
   Results can be calculated for all possible pairs.
   Efficiency: *linear* function of the number of digits.
2. Can i reach Roger Federer on a social site by friend links?
   Probably yes, unless i freshly registered. If yes, what is the minimum number of links needed?
   REACHABILITY problem: *Is node v reachable from node u in graph G? (Reachable: there is a path)*
   Algorithmic solution: breadth first search for *v* from *u*
   Efficiency: *linear* function of the number of edges.

Introduction

Example problems

3. Decide in propositional logic whether a formula is a consequence of a finite set of formulas. *It is enough to decide satsfiability for a set of clauses (or a formula in CNF)*
   SAT problem: given a formula in CNF, is it satisfiable?
   An **algorithmic solution:** truth table.
   Efficiency: Exponential (2^n for n atoms). No polynomial solution is known (and expected to have no such in the future).
4. A tourist want to visit a list of n sights in a city, but does not want to visit any of them twice. He starts from his hotel, and can return to his hotel again only after the tour. This corresponds to the problem of finding a HAMILTONIAN CYCLE in a graph.
   An **algorithmic solution:** try each possibility
   Efficiency: $n!$, no polynomial algorithm is known (and expected to have no such in the future).

Introduction

Example problems

5. Traveling salesman problem (TSP): EA traveling salesman wants to visit each of n cities. There are air links between some of the cities with a cost. What is minimum of the total cost?
   algorithmic solution: try each permutation and register the minimum found so far. *generalization of Hamilton cycle*.
6. Membership problem of generative grammars
   efficiency of an algorithmic solution:
   Type 3: linear,
   Type 2: cubic,
   Type 1 exponential,
   Type 0 ??? (to be continued...)

Introduction

Is there an algorithm for all problems?

We may have a feeling, that there is either an efficient (see REACHABILITY), or at least an inefficient (non-polynomial) solution (TSP and SAT) .

David Hilbert had the same feeling and announced his project in 1920 (*Hilbert's program*).

Axiomatize all theories of mathematics with a finite set of axioms.

part of Hilbert's program was the following as well:

Give a Universal Algorithm (UA) deciding the truth value of any mathematical statement.

Introduction

Birth of computer science

Tétel (Gödel's first incompleteness theorem, 1931) No effectively produced theory that includes arithmetic of natural numbers can be sound and complete. $\Rightarrow$ fail of Hilbert's program.
What about UA? What is exactly an algorithm? 1930's: introducing several models for algorithm

- Gödel: recursive functions
- Church, Kleene, Rosser: λ-calculus
- Turing: Turing machine

Which of them is the "real" one?

From the middle of 1930's several theorems were proven stating equivalence of some of these models

Introduction

A Church-Turing thesis

Church-Turing thesis

The several models for computable functions all describe the effectively calculable functions.

Not a theorem, a thesis!!!

Accepting this thesis we can consider any of the above concepts a mathematical model for algorithms. We choose Turing machines.

Some further models equivalent with Turing machines.

- Type 0 grammars
- pushdown automaton with at least 2 stacks
- C, Java, etc.

Introduction

A negative answer

Church and Turing independently had the following

Theorem (Church, 1936)

Equivalence of two expressions in λ-calculus is algorithmically undecidable.

Theorem (Turing, 1936)

Halting problem for Turing machines is algorithmically undecidable.

Introduction

Example problems

6. (con'd.) membership problem,
   Type 0: algorithmically undecidable
   only a partially recursive algorithm is known (in the case 'yes' it terminates with 'yes', in the case 'no' either does not terminate or terminates with 'no').
7. Decision problem for first order logic (Entscheidungsproblem).
   Church és Turing (1936) proved that validity and satisfiability are algorithmically undecidable.
   (Entscheidungsproblem ~ existence of an UA).

Introduction

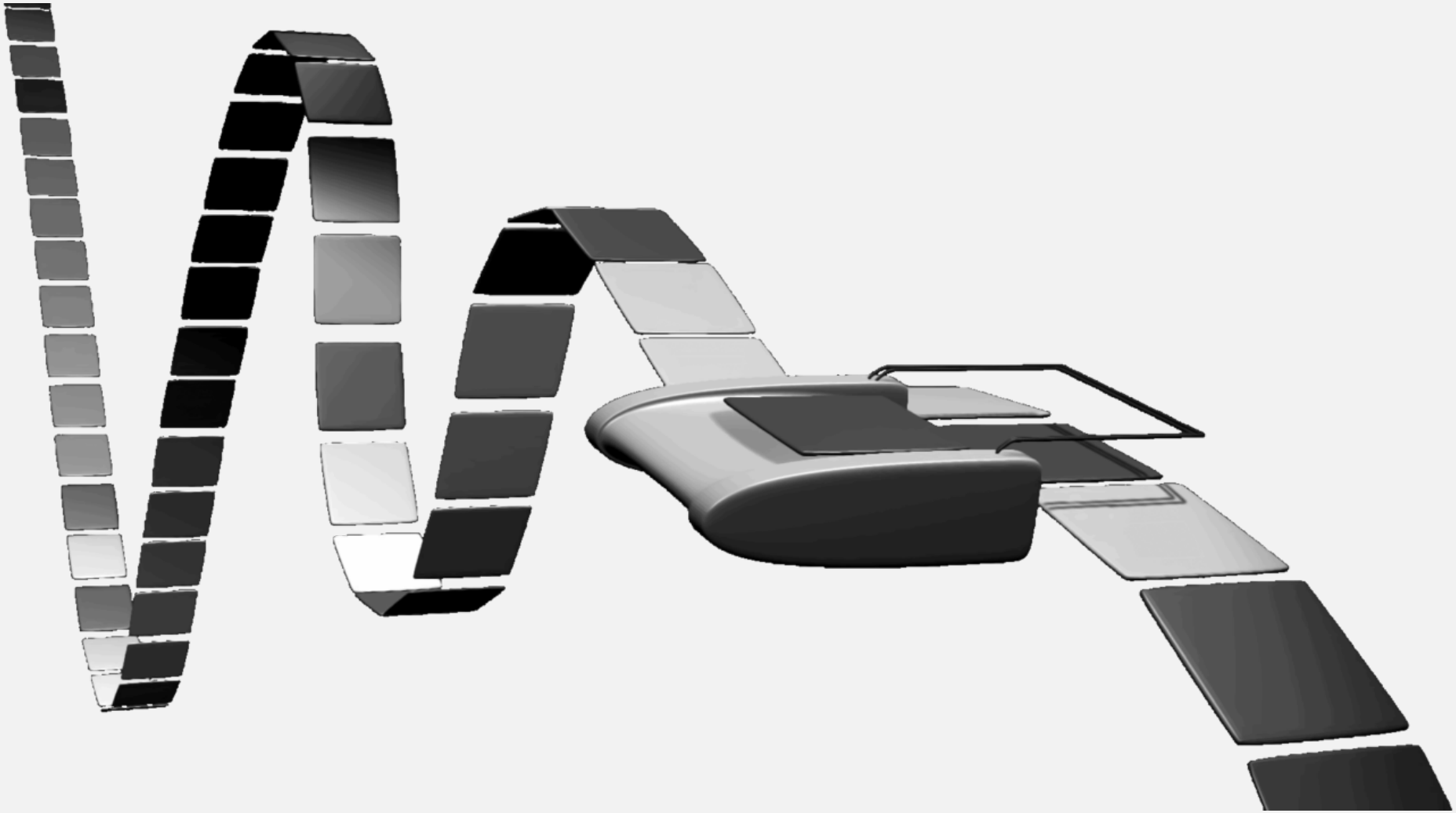
Example problems

8. Hilbert's 23 problems (1900), this is the 10th:
   Diophantine equations. Give the integer solutions of a multivariable equation with integer coefficients. E.g.,:
   $3xy - 2x^2 + 2z + 4$, output: $x = y = 0, z = -2$
   Includes "Fermat's last theorem". Wiles (1995) proved, that "Fermat's last theorem" is true, no positive integer solution for the diophantine equation $x^n + y^n = z^n$ for $n \geq 3$.
   Matiyasevich's theorem (1970): No algorithmical solution for Hilbert's 10th problem.

Short syllabus

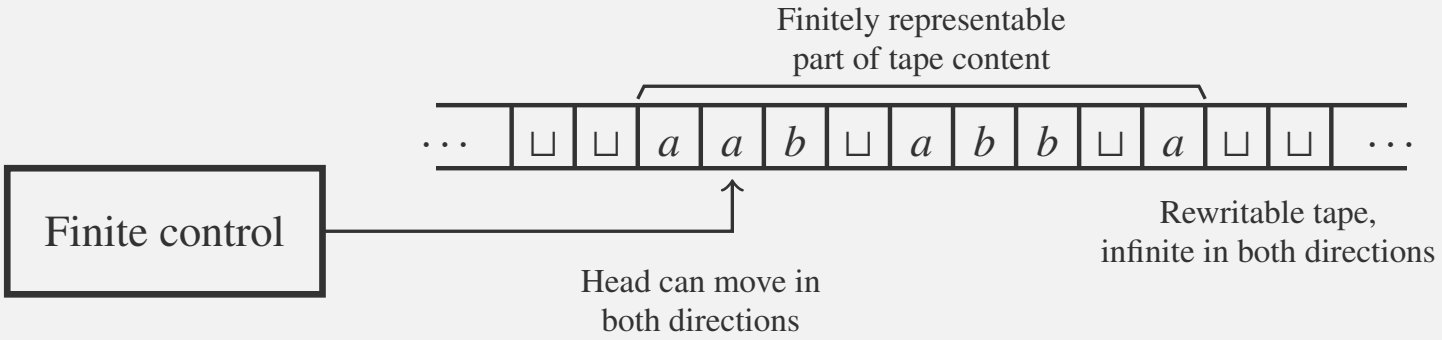
1. Turing machines as a model for algorithm
2. Algorithmically undecidable problems
3. Efficiency of algorithms for decidable problems: introduction to complexity theory

Turing machines



Turing machines

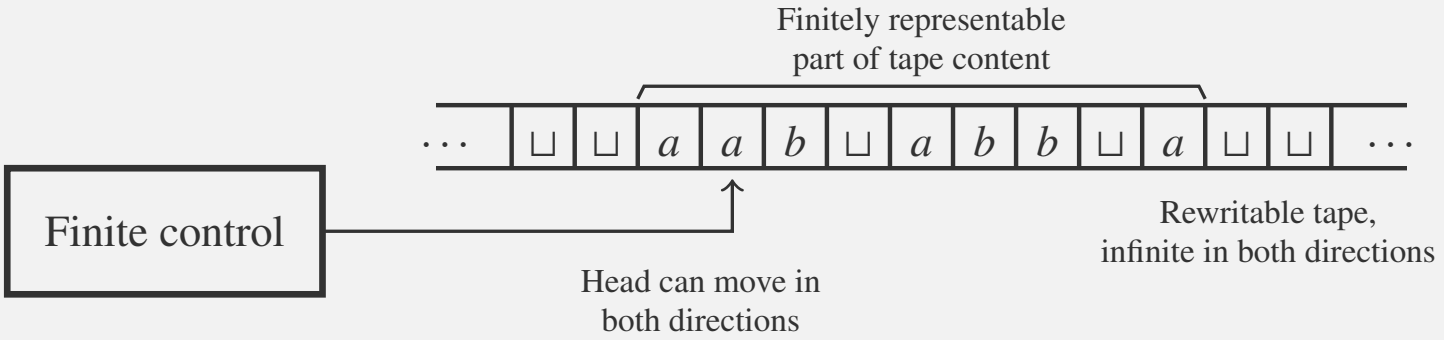
Informal introduction



- a Turing machine (TM) is a model for algorithm
- a TM is programmed to solve a single problem (but for any input)
- parts of the machine informally: finite control (with finite states), infinite tape in both directions, a head capable for moving in both directions
- initially there's a word on the tape (it's empty in case of ε), the head starts from the first letter and moves according to its rules. It accepts in its single accepting state, rejects in its single rejecting state. There's a 3rd possibility: "infinite loop"

Turing machines

Informal introduction



- the machine is deterministic, transition is well defined in all cases
- infinite tape infinite space
- for a problem $\mathcal{P}$ the yes-inputs (according to an appropiate coding) form a formal language $L_{\mathcal{P}}$. $L_{\mathcal{P}}$ (and the problem itself as well) is decidable if there's an always terminating TM accepting exactly the words of $L_{\mathcal{P}}$.
- according to the Church-Turing thesis the problems decidable by a TM are exactly the algorithmically decidable problems.

Turing machines

Definition

Turing machine

A **Turing machine** (TM from now) is a $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ 7-tuple, where

- Q is a finite, non-empty set of states,
- $q_0, q_a, q_r \in Q$, q_0 is the starting q_a is the accepting q_r is the rejecting state,
- Σ and Γ are alphabets, the input alphabet and the tape alphabet respectively $\Sigma \subseteq \Gamma$ és $\sqcup \in \Gamma \setminus \Sigma$.
- $\delta : (Q \setminus \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$ is the transition function.

The set $\{L, S, R\}$ is the set of direction (left, stay right).

Turing machines

Configurations

Configuration

A **configuration** of a TM is a word uqv , where $q \in Q$ and $u, v \in \Gamma^*, v \neq \varepsilon$.

A configuration uqv briefly describes the current situation with the TM. It contains all relevant information: the machine is in state q, the content of the tape is uv (only $\sqcup$'s before and after) and the head is on the first letter of v. Two configurations are considered to be the same if they differ only in $\sqcup$'s on the left or on the right.

For a word $u \in \Sigma^*$ the **starting configuration** is the word $q_0 u \sqcup$ (i.e, it is $q_0 u$ if $u \neq \varepsilon$, and $q_0 \sqcup$, if $u = \varepsilon$).
Accepting configurations are the configurations, where $q = q_a$.
Rejecting configurations are the configurations, where $q = q_r$.
A **halting configuration** is either an accepting or a rejecting configuration.

Turing machines

One-step transition of configurations

Let C_M be the set of all possible configurations for TM M. M the $\vdash \subseteq C_M \times C_M$ **one step transition relation** is defined as follows.

$\vdash \subseteq C_M \times C_M$ one-step transition relation

Let $uqav$ be a configuration, where $a \in \Gamma$, $u, v \in \Gamma^*$.

- If $\delta(q, a) = (r, b, R)$, then $uqav \vdash ubrv'$, where $v' = v$, if $v \neq \varepsilon$, otherwise $v' = \sqcup$,
- if $\delta(q, a) = (r, b, S)$, then $uqav \vdash urbv$,
- if $\delta(q, a) = (r, b, L)$, then $uqav \vdash u'rcbv$, where $c \in \Gamma$ and $u'c = u$, if $u \neq \varepsilon$, otherwise $u' = u$ and $c = \sqcup$.

If $C \vdash C'$ we say that C **yields** C' **in one step**.

Turing machines

Multistep transition of configurations, accepted language

Multistep transition relation: reflexive, tansitive closure of $\vdash$, i.e.

$\vdash^* \subseteq C_M \times C_M$ multistep transition relation

$C \vdash^* C' \Leftrightarrow$

- if $C = C'$ or
- if $\exists n > 0 \wedge C_1, C_2, \ldots \in C_n \in C_M$, then $\forall\, 1 \leq i \leq n-1\ C_i \vdash C_{i+1}$ holds, furthermore $C_1 = C$ and $C_n = C'$.

If $C \vdash^* C'$ we say that C **yields** C' **in finite steps**.

The language recognised by a TM M

$L(M) = \{u \in \Sigma^* \mid q_0 u \sqcup \vdash^* x q_a y \text{ for some } x, y \in \Gamma^*, y \neq \varepsilon\}$.

Turing machines

Language classes RE and R

$L \subseteq \Sigma^*$ is **Turing-recognisable**, if $L = L(M)$ holds for some TM M.
In this case we say that such a TM M **recognises** L.
$L \subseteq \Sigma^*$ is **decidable**, if there is a TM M halting on all inpunts and $L(M) = L$. In this case we say that such a TM M **decides** L.

Other names for Turing-recognisable are **recursively enumerable** (or *partially decidable*, or *semidecidable*). Another name for decidable is **recursive**.

The class of recursively enumerable languages is denoted by RE, the class of recursive languages is denoted by R.

Obviously $R \subseteq RE$ holds. Is it true, that $R \subset RE$?

Turing machines

Runtime, time complexity

The **runtime** of a TM M on a word u equals to n ($n \geq 0$), if M reaches a halting configuration from its starting configuration for u by exactly n steps (transitions). If there is no such number, we say that the running time of M on u is infinity.

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We say that M has **time complexity** $f(n)$ (or M is an $f(n)$ time-bounded Turing machine), if for all input words $u \in \Sigma^*$ M has a runtime of at most $f(|u|)$ on u.

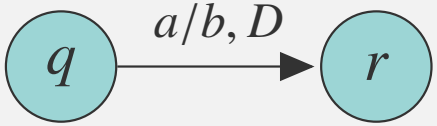
In several cases we are satisfied by a good asymptotic upper bound on the time complexity.

Turing machines

An example

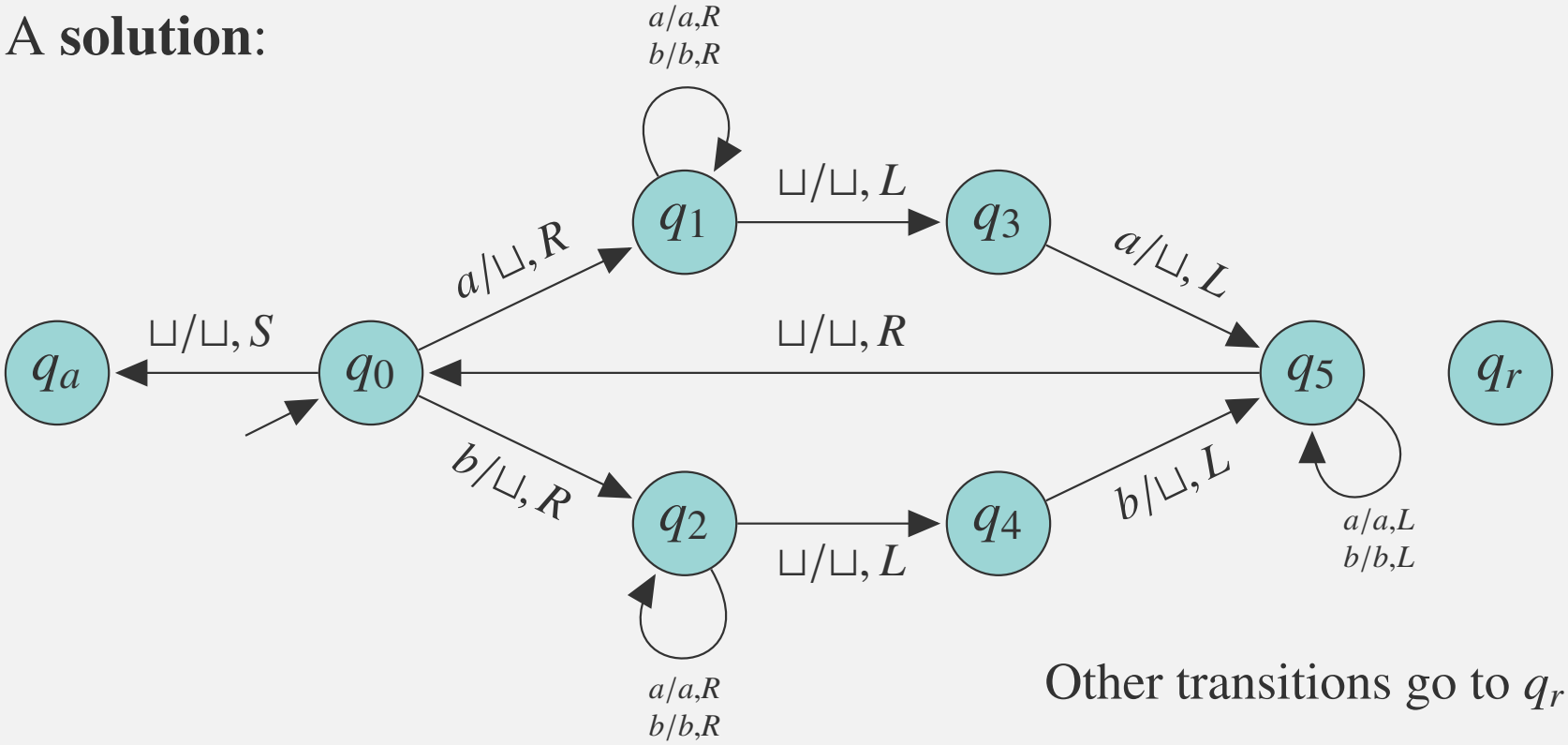
Exercise: Construct a Turing machine M so, that $L(M) = \{ww^{-1} \mid w \in \{a, b\}^*\}$!

Transition diagram.



notation of $\delta(q, a) = (r, b, D)$
$(q, r \in Q, a, b \in \Gamma, D \in \{L, S, R\})$

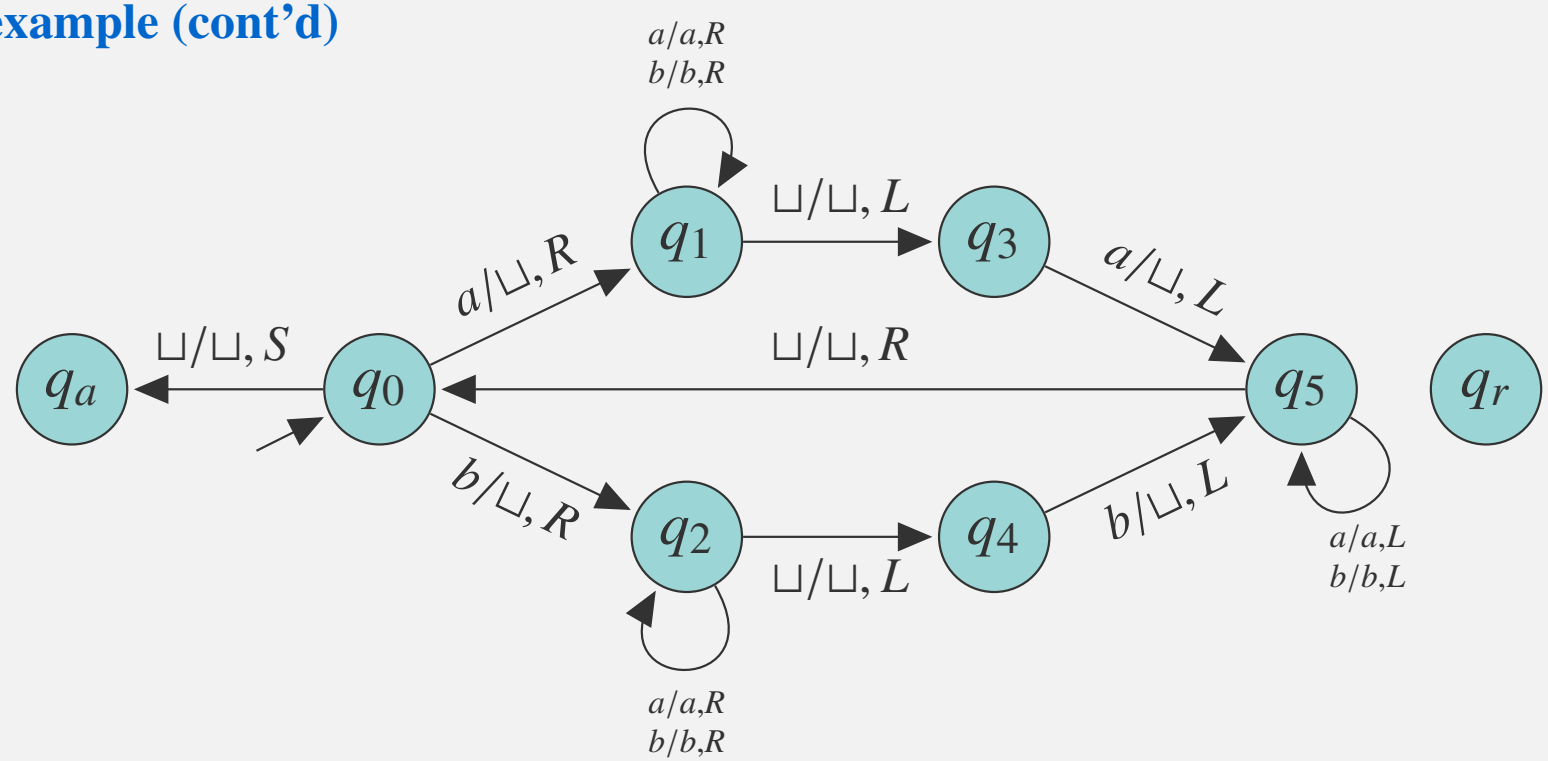
A **solution**:



Other transitions go to q_r.

Turing machines

An example (cont'd)



An example for the sequence of transitions for the input *aba*:
$q_0aba \vdash q_1ba \vdash bq_1a \vdash baq_1\sqcup \vdash bq_3a \vdash q_5b \vdash q_5\sqcup b \vdash q_0b \vdash q_2\sqcup \vdash q_4\sqcup \vdash q_r\sqcup$.
For the input *aba* the machine reaches a halting configuration in 10 steps. In this example we may have been compute the exact runtime for arbitrary n. But sometimes it is simpler (and enough to) give a good asymptotic upper bound.

Asymptotic behaviour of functions

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}_0^+$ be functions, where $\mathbb{N}$ is the set of natural numbers and $\mathbb{R}_0^+$ is the set of nonnegative numbers.

- g is an asymptotic upper bound for f (notation: $f(n) = O(g(n))$; say: $f(n)$ is big O of $g(n)$) if there is a constant $c > 0$ and a threshold $N \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ holds for all $n \geq N$.
- g is an asymptotic lower bound for f (notation: $f(n) = \Omega(g(n))$) if there is a constant $c > 0$ and a threshold $N \in \mathbb{N}$ such that $f(n) \geq c \cdot g(n)$ holds for all $n \geq N$.
- g is an asymptotic sharp bound for f (notation: $f(n) = \Theta(g(n))$) if there are constants $c_1, c_2 > 0$ and a threshold $N \in \mathbb{N}$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ holds for all $n \geq N$.

Remark: these definitions can be extended to asymptotic nonnegative functions (i.e., for functions, that are nonnegative after a threshold).

Asymptotic behaviour of functions

Classifying functions by asymptotic magnitude

One can consider O, Ω, Θ as relations of arity 2 $\mathbb{N} \to \mathbb{R}^+$ on the universe of functions.

- O, Ω, Θ are transitive (e.g.,. $f = O(g),\ g = O(h) \ \Rightarrow \ f = O(h)$)
- O, Ω, Θ are reflexive
- Θ is symmetric
- O, Ω are reversed symmetric ($f = O(g) \ \Leftrightarrow g = \Omega(f)$)
- (corr.) Θ is an equivalence relation so it partitions the class of functions of the $\mathbb{N} \to \mathbb{R}_0^+$. These classes can be represented by its "simplest" member. E.g., 1 (bounded functions), n (linear functions), n^2 (quadratic functions), etc.

Asymptotic behaviour of functions

Theorems

The following properties hold

- $f, g = O(h) \Rightarrow f + g = O(h)$, similar statement holds for Ω and Θ.
- Let $c > 0$ be a constant, $f = O(g) \Rightarrow c \cdot f = O(g)$, similar statements holds for Ω and Θ.
- $f + g = \Theta(\max\{f, g\})$
- Suppose limes of f/g exists

  if $f(n)/g(n) \to +\infty \Rightarrow f(n)=\Omega(g(n))$ and $f(n)\neq O(g(n))$

  if $f(n)/g(n) \to c \quad (c > 0) \Rightarrow f(n) = \Theta(g(n))$

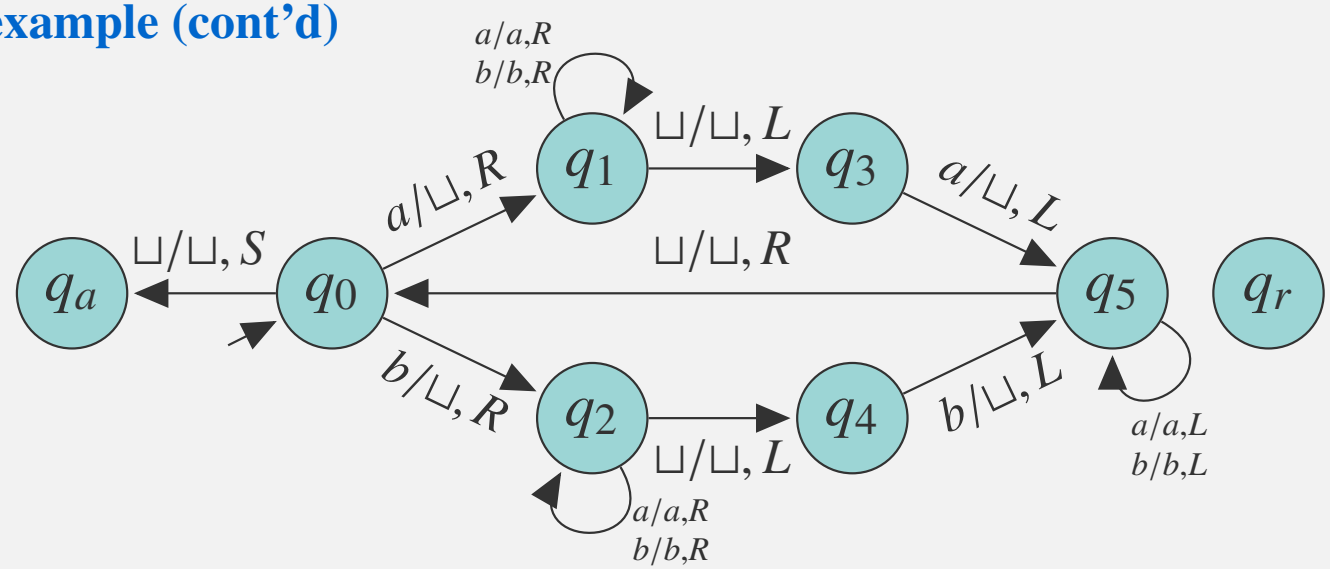
  if $f(n)/g(n) \to 0 \quad \Rightarrow f(n) = O(g(n))$ and $f(n) \neq \Omega(g(n))$

Asymptotic behaviour of functions

- let $p(n) = a_k n^k + \cdots + a_1 n + a_0$ $(a_k > 0)$, then $p(n) = \Theta(n^k)$,
- for all polynomials $p(n)$ and constant $c > 1$ $p(n) = O(c^n)$ holds, but $p(n) \neq \Omega(c^n)$,
- for all constants $c > d > 1$ $d^n = O(c^n)$ holds, but $d^n \neq \Omega(c^n)$,
- for all constants $a, b > 1$ $\log_a n = \Theta(\log_b n)$,
- for any constant $c > 0$ $\log n = O(n^c)$ holds, but $\log n \neq \Omega(n^c)$.

Turing machines

An example (cont'd)



It is a $O(n^2)$ time-bounded TM, since there are $O(n)$ steps in each of the $O(n)$ iterations, +1 step to go to either q_a or q_r.

Is there a better asymptotic upper bound for the time complexity? No, there are infinite words with $\Omega(n^2)$ steps.

Is it true, that this TM decides the language $L = \{ww^{-1} \mid w \in \{a, b\}^*\}$ or it "just" recognises it? This TM decides it.

Is there a TM, which recognises L but does not decide it? Yes, change the transitions going to q_r by redirecting them to a new state with an infinite loop, i.e., no transition going out of that state.