Logic and Theoretical Foundation of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University

Dependable Systems Group

Intractable Problems

Efficiency

- we are able to distinguish between decidable and undecidable problems

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:
  - has a graph a clique on k nodes?

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:
  - has a graph a clique on k nodes?
  - is a word in a context-free language?

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:
  - has a graph a clique on k nodes?
  - is a word in a context-free language?
  - is a boolean formula satisfiable?

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:
  - has a graph a clique on k nodes?
  - is a word in a context-free language?
  - is a boolean formula satisfiable?
  - is a boolean formula in conjunctive normal form satisfiable?

Efficiency

- we are able to distinguish between decidable and undecidable problems
- are all decidable problems *easy* to solve?
- Examples:
  - has a graph a clique on k nodes?
  - is a word in a context-free language?
  - is a boolean formula satisfiable?
  - is a boolean formula in conjunctive normal form satisfiable?
  - is a boolean formula with only two literals per conjunct satisfiable?

Determinism and Non-Determinism in TMs

- as in finite automata: expressive power is the same

Determinism and Non-Determinism in TMs

- as in finite automata: expressive power is the same
- recall the finite automata: NFA $\longrightarrow$ DFA $\hat{=}$ potential exponential blowup

Determinism and Non-Determinism in TMs

- as in finite automata: expressive power is the same
- recall the finite automata: NFA $\longrightarrow$ DFA $\hat{=}$ potential exponential blowup
- even if it works, it says nothing about the efficiency!

The Class P

Definition

time complexity $T(n)$ of a Turing machine M on input w with length n: number of M's moves until M halts

The Class P

Definition

time complexity $T(n)$ of a Turing machine M on input w with length n: number of M's moves until M halts

Definition

complexity class P contains all languages L such that there exists a DTM $\mathscr{A}$ deciding L with a time complexity being polynomial in the input size

The Problem with the Input

- Consider $n \in \mathbb{N}$ and the problem of doubling it. How many moves needs a TM?

The Problem with the Input

- Consider $n \in \mathbb{N}$ and the problem of doubling it. How many moves needs a TM?
- Consider $n \in \mathbb{N}$ and decide which subset of $\mathcal{S}_n$ is an abelian group. How long is the input?

The Problem with the Input

- Consider $n \in \mathbb{N}$ and the problem of doubling it. How many moves needs a TM?
- Consider $n \in \mathbb{N}$ and decide which subset of $\mathcal{S}_n$ is an abelian group. How long is the input?
- Important: the input has to be encodable polynomially as well!

The Class NP

Definition

complexity class NP contains all languages sucht that the time complexity of of an NTM is a polynomial in the input size

The Class NP

Definition

complexity class NP contains all languages sucht that the time complexity of of an NTM is a polynomial in the input size

Intuition: in contrast to DTM, NTM can guess among exponentially many alternatives and check each in polynomial time (in the input size) in parallel

Relation between P and NP

- NTM is DTM $\Rightarrow \mathrm{P} \subseteq \mathrm{NP}$
- it is not proven whether $\mathrm{P} \subset \mathrm{NP}$ or $\mathrm{P} = \mathrm{NP}$ holds!

Alternative Definition of P

Lemma

A language $L \in \mathrm{P}$ iff there exists polynomial-time algorithm calculating the solution

Alternative Definition of P

Lemma

A language $L \in \mathrm{P}$ iff there exists polynomial-time algorithm calculating the solution

Examples

- Calculate a minimal spanning tree for a given graph.

Alternative Definition of P

Lemma

A language $L \in \mathrm{P}$ iff there exists polynomial-time algorithm calculating the solution

Examples

- Calculate a minimal spanning tree for a given graph.
- Test if a word w is in a context-free language.

Alternative Definition of P

Lemma

A language $L \in \mathrm{P}$ iff there exists polynomial-time algorithm calculating the solution

Examples

- Calculate a minimal spanning tree for a given graph.
- Test if a word w is in a context-free language.
- Determine whether a number is prime.

Alternative Definition of NP

Lemma

A language $L \in \mathrm{NP}$ *iff there exists polynomial-time algorithm (***verifier***) testing if a given certificate is a solution.*

Alternative Definition of NP

Lemma

*A language $L \in \mathrm{NP}$ iff there exists polynomial-time algorithm (**verifier**) testing if a given certificate is a solution.*

- SAT

Alternative Definition of NP

Lemma

*A language $L \in \mathrm{NP}$ iff there exists polynomial-time algorithm (**verifier**) testing if a given certificate is a solution.*

- SAT
- Clique

Alternative Definition of NP

Lemma

*A language $L \in \mathrm{NP}$ iff there exists polynomial-time algorithm (**verifier**) testing if a given certificate is a solution.*

- SAT
- Clique
- Minesweeper

Difference between P and NP

In P we are able to find a solution in polynomial-time, whereas in NP we are only able to verify that an instance is a solution.

Idea for Polynomial-Time Reductions

- we are using the same idea as we used with decidability:

Idea for Polynomial-Time Reductions

- we are using the same idea as we used with decidability:
  - $L_1 \in \mathrm{NP}, L_1 \leq_p L_2 \Rightarrow L_2 \in \mathrm{NP}$

Idea for Polynomial-Time Reductions

- we are using the same idea as we used with decidability:
  - $L_1 \in \mathrm{NP}, L_1 \leq_p L_2 \Rightarrow L_2 \in \mathrm{NP}$
  - $L_2 \in \mathrm{P}, L_1 \leq_p L_2 \Rightarrow L_1 \in \mathrm{P}$

Polynomial-Time Reduction

Definition

$L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$: $f : \Sigma_1^* \to \Sigma_2^*$ **polynomial-time reduction** from L_1 to L_2 ($L_1 \leq_p L_2$) iff

$\exists$ pol-time TM $A : \Sigma_1^* \to \Sigma_2^* \exists$ polynomial $p \forall w \in \Sigma_1^*$:

1. $f(w) = A(w)$
2. $T_A(w) \leq p(|w|)$
3. $|f(w)| \leq p(|w|)$
4. $w \in L_1 \Leftrightarrow f(w) \in L_2$

NP-hardness and NP-completeness

Definition

- L NP-hard: $\forall M \in \mathrm{NP} : \ M \leq_p L$

NP-hardness and NP-completeness

Definition

- L NP-hard: $\forall M \in \mathrm{NP} : \; M \leq_p L$
- L NP-complete: L NP-hard and $L \in \mathrm{NP}$

NP-hardness and NP-completeness

Definition

- L NP-hard: $\forall M \in \mathrm{NP} : M \leq_p L$
- L NP-complete: L NP-hard and $L \in \mathrm{NP}$
- NPC: set of all NP-complete languages

Properties of the polynomial time-reduction

Lemma

- $L_1 \leq_p L_2 \wedge L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$

Properties of the polynomial time-reduction

Lemma

- $L_1 \leq_p L_2 \wedge L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$
- $\leq_p$ *is transitive*

Properties of the polynomial time-reduction

Lemma

- $L_1 \leq_p L_2 \wedge L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$
- $\leq_p$ *is transitive*
- L_1 NP-*hard and* $L_1 \leq_p L_2 \Rightarrow L_2$ NP-*hard*

Properties of the polynomial time-reduction

Lemma

- $L_1 \leq_p L_2 \land L_2 \in \mathrm{P} \Rightarrow L_1 \in \mathrm{P}$
- $\leq_p$ *is transitive*
- L_1 NP-*hard and* $L_1 \leq_p L_2 \Rightarrow L_2$ NP-*hard*
- $\mathrm{P} \cap \mathrm{NPC} \neq \emptyset \Rightarrow \mathrm{P} = \mathrm{NP}$