Fundamentals of theory of computation 2

1st lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Brief syllabus

- Two models of mathematical logic. Propositional logic and first order logic.
- Asympthotical properties of functions.
- Turing machines (TM) as a model of algorithms
- Multitape TM, nondeterministic TM, counting TM
- Cardinality of infinite sets.
- Algorithmic language classes RE,R and their properties
- Undecidable problems, reduction of a language (problem) to another language
- Algorithmic vs. Chomsky language classes.
- Basic concepts of complexity theory, time complexity classes, NP-completeness, polynomial time reduction
- NP-complete problems (SAT, graph problems, ...)
- Offline TM, space complexity.

Two models of mathematical logic

Basic concept: atomic **proposition** (or **statement**): either **true** or **false** independent of context. **true** and **false** are called **truth values**. Compound propositions consist of atomic propositions and linguistic connectors corresponding to Boolean operators.

I. Propositional logic (propositional calculus, zeroth-order logic)

The formulas are built from atomic propositions (variables). *Internal structure* of atomic propositions are not considered. Formulas can be combined using Boolean operators (and, or, etc.). Atomic propositions can be either true or false.

II. First-order logic (predicate logic, predicate calculus)

Statements have inner structure. Allows the use of sentences that contain variables and symbols that can be interpreted as functions and relations. Variables range over a domain. Formulas can be combined using Boolean operators (and, or, etc.) and quantifiers.

First order logic has more expressive power but a less simple model.

Propositional logic

Formal syntax

Definition

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a (countably) infinite set, its elements are called **atoms** (also called: atomic propositions, variables, atomic variables).

We define an **alphabet** (the set of terminals) as follows:

- elements of $\mathcal{P}$,
- Boolean operators:

negation	$\neg$	conjunction	$\wedge$
disjunction	$\vee$	implication	$\rightarrow$

- $($, $)$ (in the case of a string representation).

Note: sometimes further Boolean operators are used as well, such as equivalence ($\leftrightarrow$), exclusive or ($\oplus$), nor ($\downarrow$), nand ($\uparrow$).

Syntax of propositional logic

Tree representation of formulas

Definition

A **formula** is one of the following rooted node-labelled binary trees

- a single node (root) labelled by an atom
- a node (root) labelled by $\neg$ with a single child that is a formula
- a node (root) labelled by one of the binary operations with two children both of which are formulas

Definition

A **(proper) subformula** is a (proper) subtree.

Definition

Principal operator of a formula is the operator at the root of the tree.

(Formulas, that are atoms have no principal operator).

Syntax of propositional logic

String representation of formulas

Input: a formula F given by its tree representation
Output: string representation of F

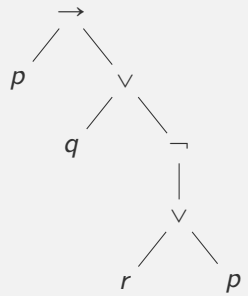
Algorithm INORDER(F)

1: **if** F is a leaf **then**
2: **write** its label
3: **if** the root of F is labelled by $\neg$ and its only child is the root of a subtree F_1 **then**
4: **write** $\neg$
5: INORDER(F_1)
6: **else** let F_1 and F_2 be the left and right subtrees of F
7: **write** a left parenthesis '('
8: INORDER(F_1)
9: **write** the label of the root of F
10: INORDER(F_2)
11: **write** a right parenthesis ')'

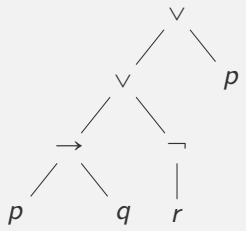
Syntax of propositional logic

The need of parenthesis

Example:



String representation:
$(p \to (q \vee \neg(r \vee p)))$



String representation:
$(((p \to q) \vee \neg r) \vee p)$

Without parenthesis there would be an ambiguity as both string representation were $p \to q \vee \neg r \vee p$.

Syntax of propositional logic

Leaving parenthesis

Another way to resolve ambiguous formulas is to define precedence and associativity conventions.

Analogy: arithmetic expression, e.g., $3 + 8 \cdot 7$.

Order of precedence from high to low: $\neg$, $\wedge$, $\vee$, $\to$.

Operators are assumed to associate to the right, that is, $A \vee B \vee C$ means $(A \vee (B \vee C))$.

Parentheses are used only if needed to indicate an order different from that imposed by the rules for precedence and associativity.

Examples, minimum number of parenthesis:

$(p \to (q \vee \neg(r \vee p))) \quad \Rightarrow \quad p \to q \vee \neg(r \vee p)$
$(((p \to q) \vee \neg r) \vee p) \quad \Rightarrow \quad ((p \to q) \vee \neg r) \vee p$

Semantics of propositional logic

Interpretation

Analogy: arithmetic expressions. E.g., $y = a + 2 \cdot b$. Assign values to a and b then evaluate y.

Definition

Let A be a formula and let P_A be the set of atoms appearing in A. An **interpretation** for A is a total function $I_A : P_A \to \{T, F\}$ that assigns one of the truth values T or F to every atom in P_A.

We will use the short notation I instead of I_A whenever the formula is clear from the context. Example:

$A = p \to q \vee \neg(r \vee p)$. $P_A = \{p, q, r\}$. One possibility is the following: $I(p) = T$, $I(q) = F$, $I(r) = F$.
There are 8 possibilities for I in this case.

Semantics of propositional logic

Evaluation of a formula

Definition

Let I be an interpretation for a formula A. $v_I(A)$, the **truth value of A under I** is defined recursively as follows:

$v_I(A) = I(A)$	if A is an atom
$v_I(\neg A) = T$	if $v_I(A) = F$
$v_I(\neg A) = F$	if $v_I(A) = T$
$v_I(A_1 \wedge A_2) = T$	if $v_I(A_1) = T$ and $v_I(A_2) = T$
$v_I(A_1 \wedge A_2) = F$	in the other three cases
$v_I(A_1 \vee A_2) = F$	if $v_I(A_1) = F$ and $v_I(A_2) = F$
$v_I(A_1 \vee A_2) = T$	in the other three cases
$v_I(A_1 \to A_2) = F$	if $v_I(A_1) = T$ and $v_I(A_2) = F$
$v_I(A_1 \to A_2) = T$	in the other three cases

Semantics of propositional logic

Example for an evaluation

We will use the short notation v instead of v_I or v_{I_A} whenever the formula A and the interpretation I is clear from the context.

Example:

$A = p \to q \vee \neg(r \vee p)$.

$P_A = \{p, q, r\}$.

$I(p) = T$, $I(q) = F$, $I(r) = F$.

$v(r \vee p) = v(r) \vee v(p) = F \vee T = T$.

$v(\neg(r \vee p)) = \neg v(r \vee p) = \neg T = F$.

$v(q) = F$, $v(\neg(r \vee p)) = F$, so $v(q \vee \neg(r \vee p)) = F$

$v(p) = T$, $v(q \vee \neg(r \vee p)) = F$, so $v(A) = F$.

Semantics of propositional logic

Truth table

We may be interested in the truth value for every possible interpretation of a formula.

Let A be a formula and suppose, that $|P_A| = n$.

Since each of the n atoms can be assigned T or F independently, there are 2^n possible interpretations.

Definition

A **truth table** for a formula A is a table with $n + 1$ columns and 2^n rows, where $n = |P_A|$. There is a column for each atom in P_A , plus a column for the formula A. The first n columns specify the interpretation I that maps atoms in P_A to $\{T, F\}$. The last column shows $v_I(A)$, the truth value of A for the interpretation I.

Semantics of propositional logic

Example for truth table

Example:

Let us make the truth table for $A = p \to q \vee \neg(r \vee p)$.

p	q	r	$r \vee p$	$\neg(r \vee p)$	$q \vee \neg(r \vee p)$	$p \to q \vee \neg(r \vee p)$
T	T	T	T	F	T	T
T	T	F	T	F	T	T
T	F	T	T	F	F	F
T	F	F	T	F	F	F
F	T	T	T	F	T	T
F	T	F	F	T	T	T
F	F	T	T	F	F	T
F	F	F	F	T	T	T

Semantics of propositional logic

Semantic properties of formulas

Definition

Let A be a formula

- A is **satisfiable** iff $v_I(A) = T$ for some interpretation I .
- A satisfying interpretation I is a **model** for A, denoted $I \models A$.
- A is **valid**, denoted $\models A$, iff $v_I(A) = T$ for all interpretations I . A valid propositional formula is also called a **tautology**.
- A is **unsatisfiable** iff it is not satisfiable, that is, if $v_I(A) = F$ for all interpretations I.
- A is **falsifiable**, denoted $\not\models A$, iff it is not valid, that is, if $v_I(A) = F$ for some interpretation I.

Semantics of propositional logic

Semantic properties of formulas, examples

Let A_1, A_2 be formulas.

Definition

If $v_I(A_1) = v_I(A_2)$ for all interpretations I , then A_1 is **logically equivalent** to A_2 , denoted $A_1 \equiv A_2$.

Examples:

- Let $A = p \to q \vee \neg(r \vee p)$.

  Then A is satisfiable, falsifiable, interpretation TTT is a model for A. (See its truth table.)

  On the other hand A is not valid and not unsatisfiable.

- formula $p \vee \neg p$ is valid, on the other hand $p \wedge \neg p$ is unsatisfiable.

- $p \vee q$ and $q \vee p$ are logically equivalent formulas.

Laws of propositional logic

Let us extend the syntax of Boolean formulas to include the two constant atomic propositions $\top$ (true) and $\bot$ (false).Their semantics are defined as $I(\top) = T$ and $I(\bot) = F$ for any interpretation I.

- $A \vee \top \equiv \top$ and $A \wedge \bot \equiv \bot$ (domination laws),
- $A \vee \bot \equiv A$ and $A \wedge \top \equiv A$ (identity laws),
- $A \vee \neg A \equiv \top$ and $A \wedge \neg A \equiv \bot$,
- $\neg\neg A \equiv A$ (double negation law),
- $A \vee A \equiv A$ and $A \wedge A \equiv A$ (idempotent laws),
- $A \to B \equiv \neg A \vee B$,
- $A \to B \equiv \neg B \to \neg A$ (law of transposition),

Laws of propositional logic
(cont'd)

- $A \vee B \equiv B \vee A$ and $A \wedge B \equiv B \wedge A$ (commutative laws),
- $(A \vee B) \vee C \equiv A \vee (B \vee C)$ and $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ (associative laws),
- $(A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$ and $(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$ (distributive laws),
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$ and $\neg(A \vee B) \equiv \neg A \wedge \neg B$ (De Morgan laws),
- $(A \vee B) \wedge B \equiv B$ and $(A \wedge B) \vee B \equiv B$ (absorption laws).

Definitions of other Boolean ops:

- $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$,
- $A \oplus B \equiv (A \vee B) \wedge \neg(A \wedge B)$,
- $A \uparrow B \equiv \neg(A \wedge B)$,
- $A \downarrow B \equiv \neg(A \vee B)$.

Substitution

Definition

Let A be a subformula of B and let A' be any formula. $B\{A \leftarrow A'\}$, the substitution of A for A' in B, is the formula obtained by replacing all occurrences of the subtree for A in B by A'.

Example:
$B = (p \rightarrow q) \vee \neg(p \rightarrow q) \vee (\neg q \rightarrow \neg p)$.
$A = p \rightarrow q$, $A' = \neg p \vee q$
$B\{A \leftarrow A'\} = (\neg p \vee q) \vee \neg(\neg p \vee q) \vee (\neg q \rightarrow \neg p)$.
$B\{p \leftarrow \neg r\} = (\neg r \rightarrow q) \vee \neg(\neg r \rightarrow q) \vee (\neg q \rightarrow \neg\neg r)$,

Theorem

Let A be a subformula of B and let A' be a formula such that $A \equiv A'$. Then $B \equiv B\{A \leftarrow A'\}$.

Substitution

Proof:
Let $B' = B\{A \leftarrow A'\}$ and I be an arbitrary interpretation. Since $A \equiv A'$ we have $v(A) = v(A')$. We have to show that $v(B) = v(B')$.

The proof is by induction on the depth d of the highest occurrence of the subtree A in B.

If $d = 0$, there is only one occurrence of A, namely B itself. Obviously, $v(B) = v(A) = v(A') = v(B')$.

If $d \neq 0$, then B is either $\neg B_1$ or $B_1 \, op \, B_2$ for some formulas B_1, B_2 and binary operator op.

In B_1 , the depth of A is less than d. By the inductive hypothesis, $v(B_1) = v(B_1') = v(B_1\{A \leftarrow A'\})$, and similarly $v(B_2) = v(B_2') = v(B_2\{A \leftarrow A'\})$.

By the definition of v, $v(B)$ depends only on $v(B_1)$ and $v(B_2)$, so $v(B) = v(B')$ proving the theorem.

Substitution

We can prove validity/unsatisfiability of a formula or logical equivalence of two formulas by applying laws of propositional logic and the substitution theorem.

Example: $A \rightarrow (B \rightarrow A) \equiv \neg A \vee (\neg B \vee A) \equiv (\neg A \vee \neg B) \vee A \equiv (\neg B \vee \neg A) \vee A \equiv \neg B \vee (\neg A \vee A) \equiv \neg B \vee (A \vee \neg A) \equiv \neg B \vee \top \equiv \top$.

So $A \rightarrow (B \rightarrow A)$ is a valid formula.

Semantic properties of a set of formulas

Satisfiability

Definition

A set of formulas $U = \{A_1, \ldots\}$ is (simultaneously) **satisfiable** iff there exists an interpretation I such that $v_I(A_i) = T$ for all i. The satisfying interpretation is a **model** of U, denoted $I \models U$. U is **unsatisfiable** iff for every interpretation I, there exists an i such that $v_I(A_i) = F$.

Example:

$U_1 = \{p, \neg p \vee q, q \wedge r\}$,
$U_2 = \{p, \neg p \vee q, \neg p\}$.
$U_3 = \{p, \neg p \vee q, \neg q\}$.
Which of the three is satisfiable?
$I \models U_1$, where $I(p) = I(q) = I(r) = T$.
p and $\neg p$ can not be T simultaneously, so for all interpretations I: $I \not\models U_2$.
One can check that for all interpretations I: $I \not\models U_3$.

Semantic properties of a set of formulas

Logical consequence

Proposition

If U is satisfiable, then U' is satisfiable, too, for all $U' \subseteq U$.
If U is unsatisfiable, then U'' is unsatisfiable for all $U \subseteq U''$.

Definition

Let U be a set of formulas and A a formula. A is a **logical consequence** of U, denoted $U \models A$, iff every model of U is a model for A.

Example:
Let $A = (p \vee r) \wedge (\neg q \vee \neg r)$. Then A is a logical consequence of $\{p, \neg q\}$, denoted $\{p, \neg q\} \models A$, since A is true in all interpretations I such that $I(p) = T$ and $I(q) = F$.
Note, that A is not valid, since it is not true in the interpretation I where $I(p) = F, I(q) = T, I(r) = T$.

Conjunctive normal form (CNF)

Definition

A **literal** is either an atom or a negation of an atom.

Definition

A formula is in **conjunctive normal form (CNF)** iff it is a conjunction of disjunctions of literals.

Example:

$(\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$	CNF
$(\neg p \vee q \vee r) \wedge ((p \wedge \neg q) \vee r) \wedge (\neg r)$	not in CNF
$(\neg p \vee q \vee r) \wedge \neg(\neg q \vee r) \wedge (\neg r)$	not in CNF
$\neg p \vee q \vee r$	CNF

Conjunctive normal form (CNF)

Theorem

For every formula A in propositional logic there is a logically equivalent formula in CNF.

Proof Let $P_A = \{p_1, \ldots, p_n\}$ and let $A^F = \{I \mid v_I(A) = F\}$ be the set of those interpretation evaluating A for false.

For every $I \in A^F$

$$B_I = \bigvee_{x:I(x)=T} \neg x \vee \bigvee_{x:I(x)=F} x$$

is a disjunction of literals with the property $B_I^F = \{I\}$.

$B = \bigwedge_{I \in A^F} B_I$ has the property

$$B^F = \bigcup_{I \in A^F} B_I^F = \bigcup_{I \in A^F} \{I\} = A^F,$$

i.e., $B \equiv A$ and B is in CNF proving the theorem.

Conjunctive normal form (CNF)

Example: Let $A = (p \to q) \to r$. Then the truth table for A is the following

p	q	r	A
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	F

According the previous proof

$$(\neg p \vee \neg q \vee r) \wedge (p \vee \neg q \vee r) \wedge (p \vee q \vee r)$$

is a CNF equivalent with A.

Conjunctive normal form (CNF)

Heuristic algorithm (can be made formal):
1st step: Eliminate implications by the law $A \to B \equiv \neg A \vee B$

2nd step: Use De Morgan's laws and double negation law to tranform the formula into a formula with the property that negations occur only right before the atoms
$\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$ and $\neg\neg A \equiv A$,

3rd step: Now, the formula is in the form of literals connected by $\wedge$'s and $\vee$'s. Use distributive laws,
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ and
$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$
to eliminate conjunctions within disjunctions or vica versa.

Example:

$$\begin{aligned}(\neg p \to \neg q) \to (p \to q) &\equiv \neg(\neg\neg p \vee \neg q) \vee (\neg p \vee q)\\ &\equiv (\neg\neg\neg p \wedge \neg\neg q) \vee (\neg p \vee q)\\ &\equiv (\neg p \wedge q) \vee (\neg p \vee q)\\ &\equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q).\end{aligned}$$

Clausal form

Clausal form is another representation of a CNF.

- A *clause* is a set of literals. Example: $\{\neg q, \neg p, q\}$.
- A clause is considered to be an implicit disjunction of its literals. Example: $\{\neg q, \neg p, q\}$ is $\neg q \vee \neg p \vee q$.
- A *unit clause* is a clause consisting of exactly one literal. Example: $\{\neg q\}$.
- The empty set of literals is the *empty clause*, denoted by $\square$.
- A formula in *clausal form* is a set of clauses. Example: $\{\{p, r\}, \{\neg q, \neg p, q\}\}$.
- A formula is considered to be an implicit conjunction of its clauses. Example: $(p \vee r) \wedge (\neg q \vee \neg p \vee q)$ for the previous one.
- The formula that is the empty set of clauses is denoted by $\varnothing$.

Removing trivial clauses

Corollary

Every formula in propositional logic can be transformed into an logically equivalent formula in clausal form.

multiple occurrences of literals and clauses $\Rightarrow$ single occurence equivalence due to idempotent laws ($A \vee A \equiv A$, $A \wedge A \equiv A$)

Example: CNF:
$(p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p) \wedge (r \vee p)$

Clausal form: $\{\{p, r\}, \{\neg q, \neg p, q\}, \{p, \neg p, q\}\}$

A clause is called **trivial** if it contains a pair of clashing literals.

Proposition

Let S be a set of clauses and let $C \in S$ be a trivial clause. Then $S - \{C\}$ is logically equivalent to S.

True, because of $A \vee \top \equiv \top$ and $A \wedge \top \equiv A$. So we can delete trivial clauses.

Empty clause and the empty set of clauses

Proposition

$\square$ (empty clause) is unsatisfiable. $\varnothing$ (the empty set of clauses) is valid.

Proof: A clause is satisfiable iff there is some interpretation under which at least one literal in the clause is true.

Let I be an arbitrary interpretation. Since there are no literals in $\square$, there are no literals whose value is true under I.

But I was an arbitrary interpretation, so is unsatisfiable.

A set of clauses is valid iff every clause in the set is true in every interpretation.

But there are no clauses in $\varnothing$ that need to be true, so $\varnothing$ is valid.

A short notation for clausal form

Notation

The set delimiters { and } are removed from each clause and a negated literal is denoted by a bar over the atomic proposition.

Let CNF(A) and cf(A) denote a CNF and a clausal form for a formula A, respectively.

Example:

$A = (p \vee r) \wedge (q \rightarrow \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p) \wedge (r \vee p)$

$\text{CNF}(A) = (p \vee r) \wedge (\neg q \vee \neg p \vee q) \wedge (p \vee \neg p \vee q \vee p \vee \neg p) \wedge (r \vee p)$

$\text{cf}(A) = \{\{p, r\}, \{\neg q, \neg p, q\}, \{p, \neg p, q\}\}$ which becomes

$\text{cf}(A) = \{pr, \bar{q}\bar{p}q, p\bar{p}q\}$ with the shorter notation.

Logical consequence in propositional logic

Theorem

Let $U = \{A_1, \ldots A_n\}$ be a finite set of formulas and let B be a formula. Then the following statements are equivalent.

- $\{A_1, \ldots A_n\} \models B$
- $\{A_1 \wedge \cdots \wedge A_n\} \models B$
- $\models A_1 \wedge \cdots \wedge A_n \rightarrow B$
- $\models \neg A_1 \vee \cdots \vee \neg A_n \vee B$
- $\models A_1 \rightarrow (A_2 \rightarrow (\cdots (A_n \rightarrow B) \cdots)$
- $A_1 \wedge \cdots \wedge A_n \wedge \neg B$ unsatisfiable
- $\{A_1, \ldots, A_n, \neg B\}$ unsatisfiable
- $\{\text{CNF}(A_1), \ldots, \text{CNF}(A_n), \text{CNF}(\neg B)\}$ unsatisfiable
- $\text{cf}(A_1) \cup \cdots \cup \text{cf}(A_n) \cup \text{cf}(\neg B)$ unsatisfiable

In order to decide logical consequence it is enough to decide whether a set of clauses is unsatisfiable.

Resolution rule

Definition and example

If ℓ is a literal, let ℓ^c denote its complementary pair.

Definition

Let C_1, C_2 be clauses such that $\ell \in C_1, \ell^c \in C_2$. The clauses C_1, C_2 are said to be clashing clauses and to clash on the complementary pair of literals ℓ, ℓ^c . C, the resolvent of C_1 and C_2 , is the clause:

$$\text{Res}(C_1, C_2) = (C_1 - \{\ell\}) \cup (C_2 - \{\ell^c\}).$$

C_1 and C_2 are the parent clauses of C.

Example: $C_1 = ab\bar{c}$ and $C_2 = bc\bar{e}$

They clash on the pair of complementary literals $c, \bar{c}$.

$\text{Res}(C_1, C_2) = (ab\bar{c} - \{\bar{c}\}) \cup (bc\bar{e} - \{c\}) = ab \cup b\bar{e} = ab\bar{e}$.

Recall that a clause is a set so duplicated literals are removed when taking the union.

Resolution

Case of more than one pair of clashing clauses

Resolution is only performed if the pair of clauses clash on exactly one pair of complementary literals due to the following.

Proposition

If two clauses clash on more than one literal, their resolvent is a trivial clause.

Remark: It is not strictly incorrect to perform resolution on such clauses, but since trivial clauses contribute nothing to the satisfiability or unsatisfiability of a set of clauses, we agree to delete them from any set of clauses and not to perform resolution on clauses with two clashing pairs of literals.

Resolution procedure

Lemma

Let I be an interpretation. If $I \models \{C_1, C_2\}$ then $I \models \mathrm{Res}(C_1, C_2)$. If $I \models \mathrm{Res}(C_1, C_2)$, then I can be extended to $\hat{I}$, such that $\hat{I} \models \{C_1, C_2\}$.

Corollary

Let S be a set of clauses and let $C_1, C_2 \in S$ be a pair of clashing clauses. Then S is satisfiable if and only if $S \cup \{\mathrm{Res}(C_1, C_2)\}$ is satisfiable.

If a set of clauses S contains $\square$ then S is unsatisfiable.

Resolution procedure

Algorithm

Let $\binom{S}{2}$ denote the set of 2-element subsets of S.

Algorithm RESOLUTION PROCEDURE(S)

```
1: while there is an unmarked pair of (S 2) do
2:    choose an unmarked pair {C1, C2} of (S 2) and mark it
3:    if {C1, C2} is a clashing pair of clauses then
4:       C ← Res(C1, C2)
5:       if C = □ then
6:          return 'S is unsatisfiable'
7:       else
8:          if C is not the trivial clause then
9:             S ← S ∪ {C}
10: return 'S is satisfiable'
```

Resolution procedure

Example

Consider the set of clauses

$$S = \{\, (1)\, p,\ (2)\, \bar{p}q,\ (3)\, \bar{r},\ (4)\, \bar{p}\bar{q}r \,\},$$

where the clauses have been numbered. Here is a resolution derivation of $\square$ from S, where the justification for each line is the pair of the numbers of the parent clauses that have been resolved to give the resolvent clause:

(5)	$\bar{p}\bar{q}$	Res((3),(4))
(6)	$\bar{p}$	Res((5),(2))
(7)	$\square$	Res((6),(1))

Resolution procedure

An algorithm for a decision problem (only 'yes' or 'no' outputs) is said to be **sound** if it never gives wrong answer for the 'yes' instances. An algorithm is said to be **complete** if it gives correct answer for all 'yes' instances. So a sound and complete algorithm for a decision problem gives a 'yes' answer exactly for the 'yes' instances.

Theorem

The resolution procedure always halts and it is sound and complete.

(without proof)

Fundamentals of theory of computation 2

2nd lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Syntax of first-order logic

First-order logic is an extension of propositional logic that includes predicates interpreted as relations on a domain.

Definition

Let $\mathcal{P}$, $\mathcal{F}$, $\mathcal{A}$ and $\mathcal{V}$ be countable sets of **predicate symbols**, **function symbols**, **constant symbols** and **variables**. Each predicate symbol $p^n \in \mathcal{P}$ and function symbol $f^n \in \mathcal{F}$ is associated with an **arity**, the number $n \geqslant 1$ of arguments that it takes. p^n is called an n-ary predicate (symbol), while f^n is called an n-ary function (symbol).

For $n = 1, 2$ we can use unary and binary respectively for n-ary.

Terms

Terms are defined recursively as follows:

Definition

- A variable or a constant is a **term**.
- If f^n is an n-ary function symbol ($n \geqslant 0$) and $t_1, t_2, \ldots, t_n$ are terms, then $f^n(t_1, t_2, \ldots, t_n)$ is a term.

Note, that 0-ary functions and constants are basically the same. The superscript denoting the arity of the function will not be written since the arity can be inferred from the number of arguments.

Example:
Let $f, g \in \mathcal{F}$ be a binary and a unary function symbol, respectively.
Let $a \in \mathcal{A}$ be a constant and $x, y \in \mathcal{V}$ be variables.
The following strings are terms:
$a, y, f(x, y), g(g(x)), f(g(f(x, y)), a)$.
The following strings are not: $f(f(x), x), g(x, x, x)$.

Formulas

Definition

An **atomic formula** is an n-ary predicate followed by a list of n arguments in parentheses $p(t_1, t_2, \ldots, t_n)$ where each argument t_i is a term.

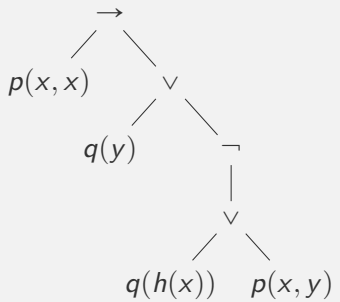
A formula in first-order logic is a tree defined recursively as follows.

Definition

- A **formula** is a leaf labeled by an atomic formula.
- A formula is a node labeled by $\neg$ with a single child that is a formula.
- A formula is a node labeled by a binary Boolean operator $(\wedge, \vee, \rightarrow)$ with two children both of which are formulas.
- A formula is a node labeled by $\forall x$ or $\exists x$ (for some variable x) with a single child that is a formula.

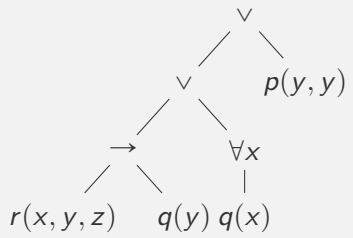
Example: formula

Let p be a binary, q be a unary and r be a 3-ary predicate symbol.
Let h be a unary function symbol and let x, y, z be variables.



String representation:

$(p(x,x) \to (q(y) \vee (\neg(q(h(x)) \vee p(x,y)))))$



String representation:

$(((r(x,y,z) \to q(y)) \vee (\forall x q(x))) \vee p(y,y))$

The following strings are not formulas in the same first order logic:
$\forall x h(x)$ h is not a predicate symbol
$\neg q(x,y)$ arity of q is not 2
$q(q(x))$ $q(x)$ is not a term

Subformula, principal operator, leaving parantheses, scope

$\forall$ is the universal quantifier and is read **for all**. $\exists$ is the existential quantifier and is read **there exists**.

A formula of the form $\forall xA$ is called a **universal formula**. Similarly, a formula of the form $\exists xA$ is an **existential formula**.

Subformula and principal operator: same as in prop. logic.

Leaving parantheses: quantifiers are considered to have the same precedence as negation and a higher precedence than the binary operators, otherwise the same.

Definition

A universal or existential formula $\forall xA$ or $\exists xA$ is a **quantified formula**, x is called a **quantified variable** and its **scope** is the formula A.

It is not required that x actually appear in the scope of its quantification.

Free and bound variables

Definition

Let A be a formula. $x \in \mathcal{V}$ is a **free variable** of A iff x has a non-quantified occurance in A, such that x is not within the scope of a quantified variable x. A variable which is not free is called **bound**.

Example: $A = \forall x(\exists y(p(x,y) \to q(x)) \wedge q(y))$.

The scope of $\exists y$ is $p(x,y) \to q(x)$.

The scope of $\forall x$ is $\exists y(p(x,y) \to q(x)) \wedge q(y)$.

Both occurance of x in A is in the scope of $\forall x$. So x is a bound variable of A.

The second occurance of y is not in the scope of an $\exists y$ or a $\forall y$, so y is a free variable of A.

Closed formula

Definition

If a formula has no free variables, it is called a **closed** formula.

Definition

If $x_1, \ldots, x_n$ are all the free variables of A, the **universal closure** of A is $\forall x_1 \cdots \forall x_n A$ and the **existential closure** is $\exists x_1 \cdots \exists x_n A$. $A(x_1, \ldots, x_n)$ indicates that the set of free variables of the formula A is a subset of $\{x_1, \ldots, x_n\}$.

Example: $A = \forall x(\exists y(p(x,y) \to q(x)) \wedge q(y))$.

y is a free variable of A, so $A = A(y)$ is not closed,

Existential closure of $A(y)$:

$\exists y A(y) = \exists y \forall x(\exists y(p(x,y) \to q(x)) \wedge q(y))$.

Universal closure of $A(y)$:

$\forall y A(y) = \forall y \forall x(\exists y(p(x,y) \to q(x)) \wedge q(y))$.

Semantics of first-order logic

Interpretation

Definition

Let U be a set of formulas such that $\{p_1, \ldots, p_k\}$ are all the predicate symbols, $\{f_1, \ldots, f_\ell\}$ are all the funtion symbols and $\{a_1, \ldots, a_m\}$ are all the constants appearing in U. An **interpretation** $\mathcal{I}$ for U is a 4-tuple:

$$(D, \{R_1, \ldots, R_k\}, \{F_1, \ldots, F_\ell\}, \{d_1, \ldots, d_m\}),$$

consisting of a non-empty set D called the **domain**, an assignment of an n_i-ary relation R_i on D to the n_i-ary predicate symbol p_i $(1 \leqslant i \leqslant k)$, an assignment of an n_j-ary function F_j on D to the n_j-ary function symbol f_j $(1 \leqslant j \leqslant \ell)$, and an assignment of an element $d_n \in D$ to the constant a_n $(1 \leqslant n \leqslant m)$.

If $U = \{A\}$, we say that $\mathcal{I}$ is an interpretation for A.

Interpretation – examples

Here are three interpretations for the formula $\forall x p(a, x)$:

$\mathcal{I}_1 = (\mathbb{N}, \{\leqslant\}, \{\}, \{0\})$,
$\mathcal{I}_2 = (\mathbb{N}, \{\leqslant\}, \{\}, \{1\})$,
$\mathcal{I}_3 = (\mathbb{Z}, \{\leqslant\}, \{\}, \{0\})$.
The domain is either $\mathbb{N}$, the set of natural numbers, or $\mathbb{Z}$, the set of integers.
The binary relation $\leqslant$ (less-than-or-equal-to) is assigned to the binary predicate p and either 0 or 1 is assigned to the constant a.

The formula can also be interpreted over strings:
$\mathcal{I}_4 = (\mathcal{S}, \{\sqsubseteq\}, \{\}, \varepsilon)$.
The domain $\mathcal{S}$ is a set of strings, $\sqsubseteq$ is the binary relation such that $(s_1, s_2) \in \sqsubseteq$ iff s_1 is a substring of s_2, and ε is the empty string of length 0.
Note, that no function was needed in the interpretations.

Evaluating terms

Definition

Let $\mathcal{I}$ be an interpretation for a formula A. An **assignment** $\sigma_\mathcal{I} : \mathcal{V} \to D$ is a function which maps every free variable $v \in V$ to an element $d \in D$, where D is the domain of $\mathcal{I}$.

In a given interpretation $\mathcal{I}$ we may write σ for $\sigma_\mathcal{I}$.

Definition

$\mathcal{D}_{\mathcal{I},\sigma}(t)$, the **value of a term** t given an interpretation $\mathcal{I}$ and assignment σ is defined recursively as follows

- for a constant $a \in \mathcal{A}$ that is interpreted for $d \in D$ let $\mathcal{D}_{\mathcal{I},\sigma}(a) = d$,
- for a variable $v \in \mathcal{V}$ let $\mathcal{D}_{\mathcal{I},\sigma}(v) = \sigma(v)$,
- for a term $f(t_1, \ldots, t_n)$ where f is interpreted for F let $\mathcal{D}_{\mathcal{I},\sigma}(f(t_1, \ldots, t_n)) = F(\mathcal{D}_{\mathcal{I},\sigma}(t_1), \ldots \mathcal{D}_{\mathcal{I},\sigma}(t_n))$.

Evaluating terms – example

Example:

Let $t = f(f(x, g(a)), g(y))$ be a term. Consider the interpretations

$\mathcal{I}_5 = (\mathbb{N}, \{\}, \{+, \mathit{next}\}, \{0\})$,

$\mathcal{I}_6 = (\{0, 1\}, \{\}, \{+_{\text{mod } 2}, \mathit{next}_{\text{mod } 2}\}, \{0\})$,
where $\mathit{next}(x)$ assigns the next number to x, e.g., 13 for 12.

Let $\sigma(x) = 7, \sigma(y) = 5$. Then $\mathcal{D}_{\mathcal{I}_5,\sigma}(t) = 14$.

Let $\sigma'(x) = 1, \sigma'(y) = 0$. Then $\mathcal{D}_{\mathcal{I}_6,\sigma'}(t) = 1$.

Note, that the result is always an element of the respective domain.

Notation

For an assignment σ for an interpretation $\mathcal{I}$, variable x and $d \in D$ let $\sigma[x \leftarrow d]$ denote the assignment that is the same as σ except that x is mapped to d.

Truth value of a formula of first-order logic

Definition

Let A be a formula, $\mathcal{I}$ an interpretation and $\sigma_{\mathcal{I}}$ an assignment. $v_{\mathcal{I},\sigma}(A)$, the **truth value of A under $\mathcal{I}$ and $\sigma_{\mathcal{I}}$**, is defined by recursion on the structure of A as follows

- Let $A = p(t_1, \ldots, t_n)$ be an atomic formula where each t_i is a term. $v_{\mathcal{I},\sigma}(A) = T$ iff $(\mathcal{D}_{\mathcal{I},\sigma}(t_1), \ldots, \mathcal{D}_{\mathcal{I},\sigma}(t_n)) \in R$ where R is the relation assigned by $\mathcal{I}_A$ to p.
- $v_{\mathcal{I},\sigma}(\neg A_1) = T$ iff $v_{\mathcal{I},\sigma}(A_1) = F$.
- $v_{\mathcal{I},\sigma}(A_1 \vee A_2) = T$ iff $v_{\mathcal{I},\sigma}(A_1) = T$ or $v_{\mathcal{I},\sigma}(A_2) = T$, and similarly for the other Boolean operators.
- $v_{\mathcal{I},\sigma}(\forall x A_1) = T$ iff $v_{\mathcal{I},\sigma[x \leftarrow d]}(A_1) = T$ for all $d \in D$.
- $v_{\mathcal{I},\sigma}(\exists x A_1) = T$ iff $v_{\mathcal{I},\sigma[x \leftarrow d]}(A_1) = T$ for some $d \in D$.

Truth value of a formula – examples

$\mathcal{I}_1 = (\mathbb{N}, \{\leqslant\}, \{\}, \{0\})$,
$\mathcal{I}_2 = (\mathbb{N}, \{\leqslant\}, \{\}, \{1\})$,
$\mathcal{I}_3 = (\mathbb{Z}, \{\leqslant\}, \{\}, \{0\})$.
$\mathcal{I}_4 = (\mathcal{S}, \{\sqsubseteq\}, \{\}, \varepsilon)$.

Example 1: Let $\sigma(x) = 7,\ \sigma(y) = 3$
$v_{\mathcal{I}_1,\sigma}(p(a,x) \rightarrow p(x,x)) = T \rightarrow T = T$.
$v_{\mathcal{I}_1,\sigma}(\neg p(x,y) \rightarrow p(x,x) \wedge p(y,a)) = \neg F \rightarrow T \wedge F = T \rightarrow F = F$.

Example 2: $A = \forall x p(a,x)$:
$v_{\mathcal{I}_1,\sigma}(A) = \mathsf{T} \quad \forall x \in \mathbb{N} : 0 \leqslant x$
$v_{\mathcal{I}_2,\sigma}(A) = \mathsf{F} \quad \forall x \in \mathbb{N} : 1 \leqslant x$
$v_{\mathcal{I}_3,\sigma}(A) = \mathsf{F} \quad \forall x \in \mathbb{Z} : 0 \leqslant x$
$v_{\mathcal{I}_4,\sigma}(A) = \mathsf{T} \quad \forall x \in \mathcal{S} : \varepsilon \sqsubseteq x$

Truth value of a closed formula

Theorem

Let A be a closed formula and let $\mathcal{I}$ be an interpretation for A. Then $v_{\mathcal{I},\sigma}(A)$ does not depend on σ.

Theorem

Let $A = A(x_1, \ldots, x_n)$ be a (non-closed) formula with free variables $x_1, \ldots, x_n$, and let $\mathcal{I}$ be an interpretation. Then:

- $v_{\mathcal{I},\sigma}(A) = T$ for some assignment σ iff $v_{\mathcal{I}}(\exists x_1 \cdots \exists x_n A) = T$.
- $v_{\mathcal{I},\sigma}(A) = T$ for all assignments σ iff $v_{\mathcal{I}}(\forall x_1 \cdots \forall x_n A) = T$.

Semantic properties of formulas

ONLY for closed formulas

Definition

Let A be a **closed** formula of first-order logic.

- A is **true** in $\mathcal{I}$ or $\mathcal{I}$ is a **model** for A iff $v_{\mathcal{I}}(A) = T$. Notation: $\mathcal{I} \models A$.
- A is **valid** if for all interpretations $\mathcal{I}$, $\mathcal{I} \models A$. *Notation:* $\models A$.
- A is **satisfiable** if for some interpretation $\mathcal{I}$, $\mathcal{I} \models A$.
- A is **unsatisfiable** if it is not satisfiable.
- A is **falsifiable** if it is not valid.

Definition

A_1 is **logically equivalent** to A_2 iff $v_{\mathcal{I}}(A_1) = v_{\mathcal{I}}(A_2)$ for all interpretations $\mathcal{I}$ for $\{A_1, A_2\}$. *Notation:* $A_1 \equiv A_2$.

Semantic properties of sets of formulas

Definition

A set of **closed** formulas $U = \{A_1, \ldots\}$ is **(simultaneously) satisfiable** iff there exists an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A_i) = T$ for all i. The satisfying interpretation is a model of U.

U is **valid** iff for every interpretation $\mathcal{I}$, $v_{\mathcal{I}}(A_i) = T$ for all i.

Definition

Let A be a **closed** formula and U be a set of **closed** formulas. A is a **logical consequence** of U iff for all interpretations $\mathcal{I}$ for $U \cup \{A\}$, $v_{\mathcal{I}}(A_i) = T$ for all $A_i \in U$ implies $v_{\mathcal{I}}(A) = T$. *Notation:* $U \models A$.

Semantic properties of sets of formulas

Similarly to propositional logic:

Theorem

Let $U = \{A_1, \ldots, A_n\}$ and A be a formula.

$$\begin{aligned} U \models A &\Leftrightarrow \models A_1 \wedge \cdots \wedge A_n \rightarrow A \\ &\Leftrightarrow A_1 \wedge \cdots \wedge A_n \wedge \neg A \text{ is unsatisfiable.} \end{aligned}$$

Remark: Definitions regarding semantic properties can be extended to **open** formulas as well by taking assingments into consideration. E.g.

Definition: A(n open) formula A is **true** in an interpretation $\mathcal{I}$ and assignment σ iff $v_{\mathcal{I},\sigma}(A) = T$. Notation: $\mathcal{I}, \sigma \models A$.

Definition: A(n open) formula is **valid**, iff $\mathcal{I}, \sigma \models A$ holds for all interpretations $\mathcal{I}$ and assignment σ. Notation: $\models A$.

etc.

Laws of first-order logic

- laws of propositional logic
- $\forall x \forall y A \equiv \forall y \forall x A$,
- $\exists x \exists y A \equiv \exists y \exists x A$,
- $\neg \exists x A \equiv \forall x \neg A$,
- $\neg \forall x A \equiv \exists x \neg A$,
- $\forall x A \wedge \forall x B \equiv \forall x (A \wedge B)$
- $\exists x A \vee \exists x B \equiv \exists x (A \vee B)$.

- $\models \exists x \forall y A(x, y) \rightarrow \forall y \exists x A(x, y)$
- $\models \forall x A(x) \vee \forall x B(x) \rightarrow \forall x (A(x) \vee B(x))$,
- $\models \exists x (A(x) \wedge B(x)) \rightarrow \exists x A(x) \wedge \exists x B(x)$.

Laws of first-order logic II.

If x is not free in B

- $\exists x A(x) \vee B \equiv \exists x (A(x) \vee B)$,
- $\forall x A(x) \vee B \equiv \forall x (A(x) \vee B)$,
- $B \vee \exists x A(x) \equiv \exists x (B \vee A(x))$,
- $B \vee \forall x A(x) \equiv \forall x (B \vee A(x))$,
- $\exists x A(x) \wedge B \equiv \exists x (A(x) \wedge B)$,
- $\forall x A(x) \wedge B \equiv \forall x (A(x) \wedge B)$,
- $B \wedge \exists x A(x) \equiv \exists x (B \wedge A(x))$,
- $B \wedge \forall x A(x) \equiv \forall x (B \wedge A(x))$.

Proving logical equivalence – example

Proposition: $\forall x A(x) \equiv \neg\exists x \neg A(x)$.

Proof:

For an arbitrary interpretation $\mathcal{I}$ and assignment σ

$v_{\mathcal{I},\sigma}(\forall x A(x)) = T$

$\Leftrightarrow v_{\mathcal{I},\sigma[x\leftarrow d]} A(x) = T$ for all $d \in D$.

$\Leftrightarrow v_{\mathcal{I},\sigma[x\leftarrow d]} \neg A(x) = F$ for all $d \in D$.

$\Leftrightarrow$ there is no $d \in D$, such that $v_{\mathcal{I},\sigma[x\leftarrow d]} \neg A(x) = T$.

$\Leftrightarrow v_{\mathcal{I},\sigma}(\exists x \neg A(x)) = F$.

$\Leftrightarrow v_{\mathcal{I},\sigma}(\neg\exists x \neg A(x)) = T$.

Proving logical consequence – example

Proposition: $\{\forall x(A(x) \to B(x)), A(a)\} \models B(a)$

Proof: Suppose, that $\mathcal{I} \models \{\forall x(A(x) \to B(x)), A(a)\}$, i.e

$v_{\mathcal{I}}(\forall x(A(x) \to B(x))) = T$ and $v_{\mathcal{I}} A(a) = T$ for some interpretation $\mathcal{I} = (D, \{R_A, R_B\}, \{\}, \{d_0\})$.

$v_{\mathcal{I},\sigma}(\forall x A(x) \to B(x)) = T$ iff $v_{\mathcal{I},\sigma[x\leftarrow d]}(A(x) \to B(x)) = T$ for all $d \in D$ (by the definition of $\forall$, for arbitrary σ).

As a specical case, if $d = d_0$ have $v_{\mathcal{I},\sigma[x\leftarrow d_0]}(A(x) \to B(x)) = T$, so the implication $R_A(d_0) \to R_B(d_0)$ holds.

Since $v_{\mathcal{I}} A(a) = T$, we have $R_A(d_0)$ implying $R_B(d_0)$. Therefore $v_{\mathcal{I}} B(a) = T$.

Remark: This is how to formalize the sentences "Every human is mortal.", "Socrates is human", "Therefore Socrates is mortal."

$A(x)$: x is human; $B(x)$: x is mortal; a: Socrates.

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 $f(n) \leqslant c \cdot g(n)$ holds for all $n \geqslant N$.
- 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 $f(n) \geqslant c \cdot g(n)$ holds for all $n \geqslant N$.
- 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 $c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n)$ holds for all $n \geqslant N$.

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

Asymptotic behaviour of functions

Classifying functions by asymptotic magnitude

One can consider O, Ω, Θ as relations of arity 2 over the universe of $\mathbb{N} \to \mathbb{R}_0^+$ 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 $\log n \neq \Omega(n^c)$.

Remark: These notations are due to German mathematician Edmund Landau.

Mathematically more precise to use the following notation instead of $f = O(g)$:

$O(g) := \{f \mid \exists c > 0\ \exists N \in \mathbb{N}\ \forall n \geqslant N : f(n) \leqslant c \cdot g(n)\}$.

Using this modern notation if g is an asymptotic upper bound of f we should write $f \in O(g)$.

Later lectures use the classical notation of Landau.

Fundamentals of theory of computation 2

3rd lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Basic notions, notations – Summary

A SUMMARY OF FOTOC 1

Alphabet: A finite, nonempty set

Letters: Elements of the alphabet.

Finite sequences over an alphabet V are called **words** or strings.

The **length of a word** $u = t_1 \cdots t_n$ is n, the number of letters of u. Notation: $|u| = n$. The word of length 0 is denoted by ε, and is called the **empty word** ($|\varepsilon| = 0$).

V^* is the notation for the **set of all words over** V including the empty word.

$V^+ = V^* \backslash \{\varepsilon\}$ is the **set of all non-empty words over** V.

Example: $V = \{a, b\}$, then
$V^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.

Basic notions, notations – Summary

Let V be an alphabet, a subset L of V^* is called a **language** over V.

The empty language (the language containing no words) is denoted by $\varnothing$.
A language is called a **finite language** if it contains finite words.

Let X be a set. $\mathcal{P}(X)$ denotes the **powerset** of X, i.e., $\mathcal{P}(X) = \{A \mid A \subseteq X\}$.

Family of languages (or class of languages) is a set of languages.

So for an alphabet V
- $a \in V$: letter
- $u \in V^*$: word
- $L \subseteq V^*$ or $L \in \mathcal{P}(V^*)$: language
- $\mathcal{L} \subseteq \mathcal{P}(V^*)$ or $\mathcal{L} \in \mathcal{P}(\mathcal{P}(V^*))$: family of languages

Grammars – Summary

Definition

A 4-tuple $G = \langle N, T, S, P \rangle$ is called a **grammar** if

- N and T are disjoint alphabets (i.e., $N \cap T = \varnothing$). Elements of N are called **nonterminals**, while elements of T are called **terminals**.
- $S \in N$ is the **start symbol** of the grammar.
- P is a **set of production rules**, each of them having the form $x \to y$, where $x \in (N \cup T)^* N (N \cup T)^*, y \in (N \cup T)^*$.

Example: For a grammar $G = \langle \{S\}, \{a\}, S, \{S \to aaS, S \to \varepsilon\} \rangle$
$S \Rightarrow aaS \Rightarrow aaaaS \Rightarrow aaaa$ is a **derivation** of the word $aaaa$.

The set of terminal words that can be generated from the start symbol of grammar G is called **the language generated by** G and denoted by $L(G)$.

For this example $L(G) = \{a^{2n} \mid n \in \mathbb{N}\}$.

Chomsky grammar classes – Summary

Let $G = \langle N, T, S, P \rangle$ be a grammar. G is of **type** i $(i = 0, 1, 2, 3)$, if P satisfies the following conditions

- for $i = 0$: no restriction,
- for $i = 1$:
  (1) the rules of P are of the form $u_1 A u_2 \to u_1 v u_2$, where $u_1, u_2, v \in (N \cup T)^*$, $A \in N$, and $v \neq \varepsilon$,
  (2) there is one possible exception: $S \to \varepsilon$ can be included in P, but only in the case when S does not appear on the right hand side of any rule of P.
- for $i = 2$: all rules of P are of the form $A \to v$ where $A \in N$ and $v \in (N \cup T)^*$,
- for $i = 3$: all rules of P are either of the form $A \to uB$ or of the form $A \to u$, where $A, B \in N$ and $u \in T^*$.

Let $\mathcal{G}_i$ denote the class of grammars of type i $(i = 0, 1, 2, 3)$.

Chomsky hierarchy – Summary

$\mathcal{L}_i := \{L \mid \exists G \in \mathcal{G}_i,$ such that $L = L(G)\}$ denotes the **family of languages of type** i. $(i = 0, 1, 2, 3)$.

Type 0 grammars are called **phrase-structure** grammars, type 1 grammars are called **context-sensitive** grammars, type 2 grammars are called **context-free** grammars. Type 3 grammars are called **regular** grammars.

The corresponding language classes are called the class of **recursively enumerable**, the class of **context-sensitive**, the class of **context-free**, and the class of **regular** languages.

Chomsky hierarchy

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0.$$

Remark: Inclusion between $\mathcal{L}_2$ and $\mathcal{L}_1$ does not follow immediately form the forms of the corresponding grammars. Proper inclusions can be proved by pumping (Bar Hillel) lemmas.

Defining formal languages – Summary

• by (generative) grammars

Grammars are **synthetizing** tools, starting form a single symbol they can build words. The set of terminal words that can be built forms a formal language.

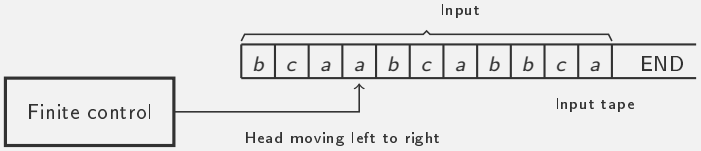
• by automata

Automata are parsing, **analyzing** tools. An input word is processed by the automata and a single bit ('yes' or 'no') is given as an output. The set of those input words giving a 'yes' output forms a formal language.

• by other ways: listing elements, regular expressions, etc.

Attention! It is not true, that all languages can be defined by each of these tools. E.g., regular expressions can not describe all type 0 languages. On the other hand, not even type 0 grammars are strong enough to describe all languages over $\{0, 1\}$.

Finite automata – Summary



- Finite automata (FA) start from their initial state. Initially an input word is written on the input tape. A head starts from the first letter and scans the input from left to right.
- The automaton works in discrete steps, in one step it reads a symbol from the tape, the state changes according to the transition function and the head moves to the right.
- While there are symbols left from the input the automaton keeps working. It does not matter if it reaches an accepting state if there are still letters to be processed. It decides on the input only at the point when the input is fully processed.

Finite automata – Summary

Definition

A (nondeterministic) **finite automaton** is a 5-tuple, $A = \langle Q, T, \delta, Q_0, F \rangle$, where

- Q is a non-empty set, the set of states
- T is the alphabet of input symbols,
- $\delta : \ Q \times T \to \mathcal{P}(Q)$ is the transition function,
- $Q_0 \subseteq Q$ is the set of initial states,
- $F \subseteq Q$ is the set of accepting states.

Definition

If $|\delta(q, a)| = 1$ and $|Q_0| = 1$ holds $\forall (q, a) \in Q \times T$ then A is called a **deterministic finite automaton** (DFA).

Type 3 languages – Summary

The **language accepted by** a finite automaton $A = \langle Q, T, \delta, Q_0, F \rangle$ consists of those words w of T for which a state of F can be reached by fully processing w.

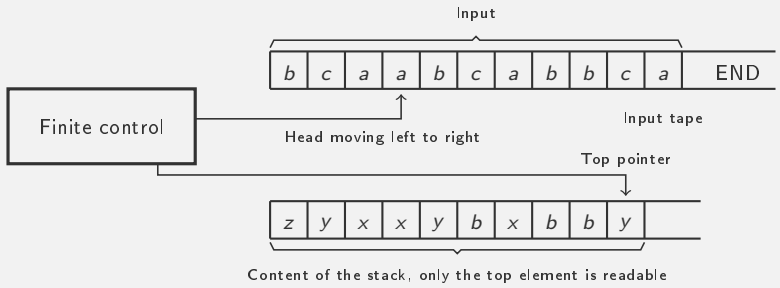
For DFA's there is exactly one way to process the input words.

Theorem

The following formal tools define exactly the type 3 languages.

- Type 3 grammars
- Type 3 grammars in normal form
- Regular expressions
- Deterministic finite automata
- Nondeterministic finite automata

Pushdown automata – Summary



- Pushdown automaton (PDA) is a generalization of FA, with a stack of unbounded capacity and finite control.
- New data is pushed into the stack over the current content, elements can popped one by one in an opposite order as they were pushed into the stack.
- pushdown automata are nondeterministic by default

Pushdown automaton – Summary

Notation: Let X be a set, $\mathcal{P}_{\text{fin}}(X)$ denotes the set of finite subsets of X.

Definition

A **pushdown automaton** is a 7-tuple $A = \langle Z, Q, T, \delta, z_0, q_0, F \rangle$, where

- Z is the stack alphabet
- Q is a non-empty finite set of states,
- T is the input alphabet,
- $\delta : Z \times Q \times (T \cup \{\varepsilon\}) \to \mathcal{P}_{\text{fin}}(Z^* \times Q)$ is the transition function
- $z_0 \in Z$ is the initial stack symbol,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states.

Pushdown automata – Summary

- Transitions depend on the current state, the symbol processed from the input and the top element of the stack.
- In every step the top element of the stack is removed and replaced by some $(0, 1, 2, \ldots)$ letters from Z.
- If $\delta(z, q, \varepsilon)$ is non-empty, then so called **ε-moves** are possible. Such moves change the state and the content of the stack without processing any symbol of the input.
- ε-moves are possible even before processing the first letter and even after processing the last one.

Pushdown automata – Summary

There are 2 acceptance modes:

The **language accepted by** a pushdown automaton $A = \langle Z, Q, T, \delta, z_0, q_0, F \rangle$ **by accepting states** consists of words w over T having a computation ending in a state from F and fully processing w.

Computation is nondeterministic and the content of the stack is irrelevant at the end of computation. This language is denoted by $L(A)$.

The **language accepted by** a pushdown automaton $A = \langle Z, Q, T, \delta, z_0, q_0, F \rangle$ **by empty stack** consists of words w over T having a computation ending with an empty stack and fully processing w.

Computation is nondeterministic and the current state is irrelevant at the end of computation. This language is denoted by $N(A)$.

Pushdown automata – Summary

Deterministic pushdown automaton

Definition

A pushdown automaton $A = \langle Z, Q, T, \delta, z_0, q_0, F \rangle$ is called **deterministic** if $|\delta(z, q, a)| + |\delta(z, q, \varepsilon)| = 1$ holds for every $(z, q, a) \in Z \times Q \times T$.

Therefore for every $q \in Q$ and $z \in Z$ we have

- either $\delta(z, q, a)$ contains exactly one element for every input symbol $a \in T$ and $\delta(z, q, \varepsilon) = \varnothing$,
- or $\delta(z, q, \varepsilon)$ contains exactly one element and $\delta(z, q, a) = \varnothing$ for every input symbol $a \in T$.

Remark: If $|\delta(z, q, a)| + |\delta(z, q, \varepsilon)| \leqslant 1$ for every $(z, q, a) \in Z \times Q \times T$ then, by adding extra transitions to a new, trap state, the pushdown automaton can be extended to a deterministic PDA of the same accepted language.

Pushdown automata – Summary

PDA's and type 2 languages

Theorem

The following statements are equivalent for every language L

- L is context-free (can be generated by a context-free grammar),
- L can be generated by a grammar in Chomsky NF,
- $L = L(A)$ for some (nondeterministic) PDA A,
- $L = N(A)$ for some (nondeterministic) PDA A

Theorem

Every regular (type 3) language can be recognized by a deterministic PDA. On the other hand, there are CF languages which can not be recognized by a deterministic PDA.

Strict types of languages – Summary

Notation: $|u|_t$: number of occurances of letter t in u

Examples:

$\in \mathcal{L}_3$	$\in \mathcal{L}_2 - \mathcal{L}_3$	$\in \mathcal{L}_1 - \mathcal{L}_2$
$\{u \mid abbab \subseteq u\}$	$\{u \in \{a,b\}^* \mid u = u^R\}$	$\{uu \mid u \in \{a,b\}^*\}$
$\{u \mid abbab \not\subseteq u\}$	$\{a^n b^n \mid n \in \mathbb{N}\}$	$\{a^n b^n c^n \mid n \in \mathbb{N}\}$
numbers divisible by 7	$\{u \in \{a,b\}^* \mid \lvert u\rvert_a = \lvert u\rvert_b\}$	$\{a^{n^2} \mid n \in \mathbb{N}\}$
$((a+bb)^* + ab)^*$	well-parenthized expr.	$\{a^{2^n} \mid n \in \mathbb{N}\}$

Algorithmic problems

For a problem $\mathcal{P} : \mathcal{I} \to \mathcal{O}$ (where $\mathcal{I}$ is the set of inputs, $\mathcal{O}$ is the set of outputs) an **algorithmic solution** is a procedure with a common list of instructions calculating $\mathcal{P}(I) \in \mathcal{O}$ for every $I \in \mathcal{I}$ input of $\mathcal{P}$.

If $\mathcal{O} = \{\text{'yes'}, \text{'no'}\}$, then we say that $\mathcal{P}$ is a **decision problem**, for the general case we say $\mathcal{P}$ is a **counting problem**.

Examples:

- $\mathcal{I} = \mathbb{N} \times \mathbb{N}$, $\mathcal{O} = \mathbb{N}$. $\mathcal{P}$ is addition. Algorithmic solution: the algorithm we learn in elementary school.
- $\mathcal{I} = \{G \mid G \text{ is a CF grammar}\} \times T^*$, $\mathcal{O} = \{\text{'yes'}, \text{'no'}\}$, where T is an alphabet. $\mathcal{P}$ is membership problem. Algorithmic solution: CYK algorithm.
- $\mathcal{I} = \{\text{first order formulas}\}$, $\mathcal{O} = \{\text{'yes'}, \text{'no'}\}$, $\mathcal{P}(\varphi) = $'yes', iff $\models \varphi$. No algorithmic solution is known.

Decision problems

Given an algorithmic decision problem $\mathcal{P}$. **yes-instances** are those inputs having the output 'yes', **no-instances** are those inputs having the output 'no'.

Yes-instances can be considered as a formal language $L_{\mathcal{P}} = \{I \in \mathcal{I} \mid \mathcal{P}(I) = \text{'yes'}\}$ over an appropriate alphabet.

A **partial algorithmic solution** is an algorithm giving the answer 'yes' exactly for $L_{\mathcal{P}}$ not necessarily terminating for no-instances. A **decision algorithm** is an algorthm terminating for all inputs and giving the answer 'yes' exactly for the words of $L_{\mathcal{P}}$.

Decision problems and automata

So far we have learnt some types of parsing automata (DFA, deterministic PDA) giving a 'yes'/'no' output for the input words. These automata can be considered as decision algorthms (they follow a finite set of instructions) for the language recognized by the automata as they terminate for all words and give the answer 'yes' for exactly the words of this language.

So these automata can be considered as a kind of restricted models for algorithms.

We can say that a problem $\mathcal{P}$ is algorhmically solved (decided) if there is an automaton recognizing exactly the words of $L_{\mathcal{P}}$.

Decision problems and automata

PDA is a generalization of FA, more problems can be decided by a PDA than by a FA.

Is PDA general enough to model the concept of algorithm?

No, PDA's can recognize only CF (type 2) languages. On the other hand, elements of an $\mathcal{L}_0(\supset \mathcal{L}_2)$ language can be synthetized by an algorithm (by a type 0 grammar).

Is there an automata (or any formal tool, model of computation) strong enough to correspond algorithms?

Models of computation (algorithm)

From 1930's several models of computation were introduced with the intention to define what is an algorithm, what is computability. Some of these models were

- Kurt Gödel's recursive functions
- Alonso Church's (Kleene, Rosser): λ-calculus
- Alan Turing's Turing machines

Which of them is the "real" one?

From the middle of 1930's several theorems were proven stating equivalence of some of these models.
Later further models were introduced. Several of them turned to have the same computational power as Turing machines. Some of these were

- type 0 grammars
- PDA's with 2 or more stacks
- C, Java, etc.

The Church-Turing thesis

Actually, we do not know any model of computation with a greater computational power than TM's. For most models it is proved to have the same or less computational power than Turing machines.

The following thesis appeared already in the 1930's

Church-Turing thesis

The above models for computable functions all describe the informal notion of effectively calculable functions.

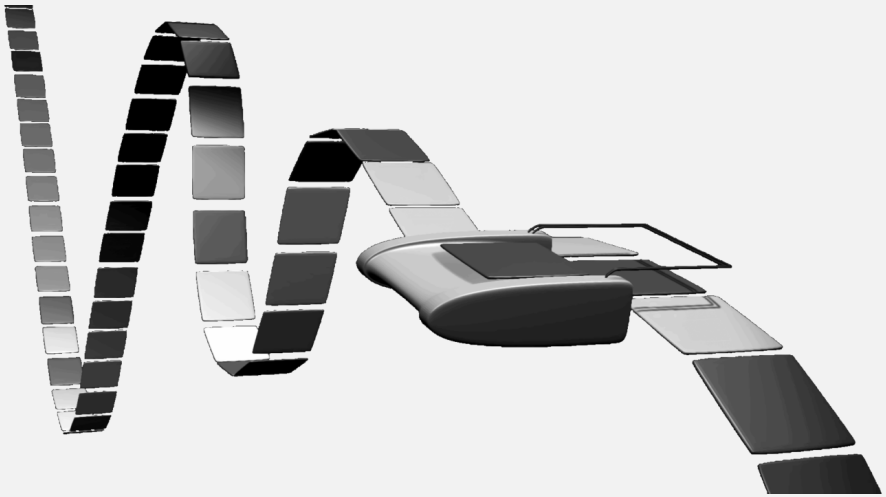
'effectively calculable': can be computed by a 'paper-pencil' method.

By other words: every formalizable problem, that can be solved by an algorithm can be solved by a Turing machine as well (or in any computational model equivalent with Turing machines)

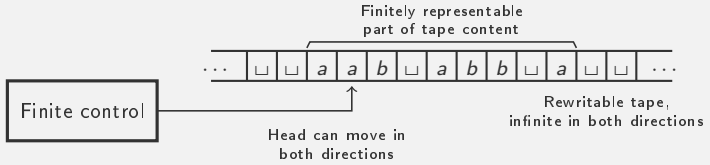
Not a theorem!!!. An intutive hypothesis, can not be proved.

Accepting the thesis, intuitively, we can consider any of the above concepts as a mathematical model for algorithms (computability).

Turing machines

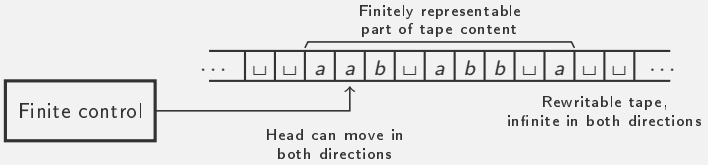


Turing machines – Informal introduction



- a Turing machine (TM) is a model for the concept of algorithm
- a TM is programmed to solve a specific problem (but for any input)
- parts of the machine are (informally): finite control (finitely many states); a tape, infinite in both directions; a head capable for moving in both directions
- initially there's an input word w on the tape (the tape is empty in the case of $w = \varepsilon$ input), the head starts from the first letter of w and the head moves according to transition 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 by default, transitions are well defined in all cases
- infinite tape means infinite storage
- for a problem $\mathcal{P}$ yes-instances form a formal language $L_{\mathcal{P}}$ (with an appropriate coding). $L_{\mathcal{P}}$ (and so the problem itself as well) is decidable if there's an always terminating TM accepting exactly the words of $L_{\mathcal{P}}$.
- according to Church-Turing thesis the problems decidable by a TM corresponds to the algorithmically decidable problems.

Turing machines – Formal definition

Definition

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 $\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma \backslash \Sigma$.
- $\delta : (Q \backslash \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$ is the transition function.

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

Configurations of Turing machines

Definition

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.

One step transition of TM's

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**.

Example: Suppose, that $\delta(q_2, a) = (q_5, b, L)$ and $\delta(q_5, c) = (q_1, \sqcup, R)$. Furthermore let $C_1 = bcq_2a\sqcup b$, $C_2 = bq_5cb\sqcup b$, $C_3 = b\sqcup q_1b\sqcup b$.
Then $C_1 \vdash C_2$ és $C_2 \vdash C_3$.

Multistep transition of TM's; recognized language

Multistep transition relation: reflexive, transitive closure of $\vdash$ denoted by $\vdash^* \subseteq C_M \times C_M$.

Definition

$C \vdash^* C' \Leftrightarrow$

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

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

Example (cont'd): Let C_1, C_2, C_3 as in the previous example. We have seen, that $C_1 \vdash C_2$ and $C_2 \vdash C_3$ holds, so $C_1 \vdash^* C_1$, $C_1 \vdash^* C_2$, $C_1 \vdash^* C_3$ hold, too.

Definition

The **language recognized by a TM** $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ is $L(M) = \{u \in \Sigma^* \mid q_0 u \sqcup \vdash^* xq_a y \text{ for some } x, y \in \Gamma^*, y \neq \varepsilon\}$.

RE and R

Definition

$L \subseteq \Sigma^*$ is **Turing-recognizable**, if $L = L(M)$ holds for some TM M. In this case we say that such a TM M **recognizes** L.

$L \subseteq \Sigma^*$ is **decidable**, if there is a TM M halting on all inputs and $L(M) = L$. In this case we say that such a TM M **decides** L.

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

Definition

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 – Running time

Definition

The **running time** of a TM M on a word u is the number of steps (transitions) M takes from the starting configuration of u to a halting configuration. If there is no such number, then running time of M on u is infinity.

Definition

Time complexity of a TM M is a function $f : \mathbb{N} \to \mathbb{N}$, such that $f(n)$ is the maximum running time on any input of length n.

We also say that M is an $f(n)$ **time** Turing machine or the **running time** of M is $f(n)$.

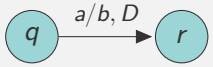
Note, that if M is an $f(n)$ time TM, then for any input word $u \in \Sigma^*$ M has a running time of at most $f(|u|)$ on u.

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

Turing machines – An example

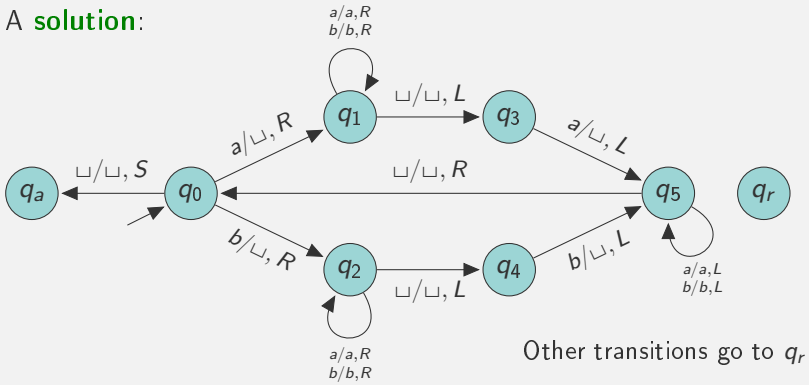
Exercise: Construct a Turing machine M so, that $L(M) = \{ww^R \mid w \in \{a,b\}^*\}$!

Transition diagram.



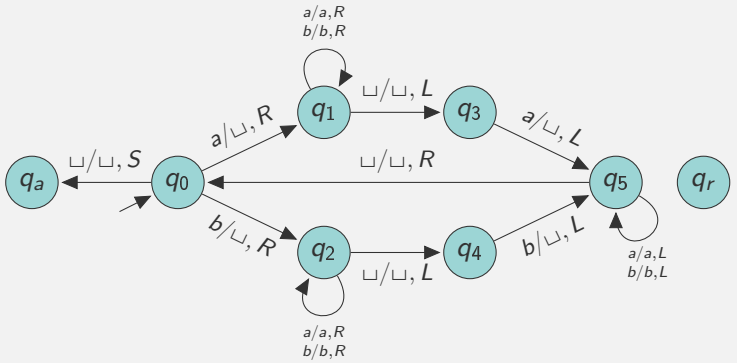
notation of $\delta(q,a) = (r,b,D)$
$(q, r \in Q, a, b \in \Gamma, D \in \{L,S,R\})$

A **solution**:



Other transitions go to q_r.

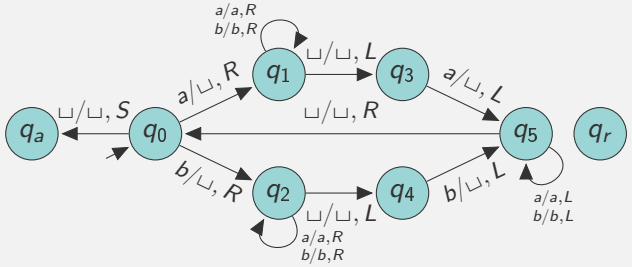
Turing machines – An example (cont'd)



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 able to compute the exact time complexity. But sometimes it is simpler (and enough to) give a good asymptotic upper bound.

Turing machines – An example (cont'd)



It is a $O(n^2)$ time 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 this TM decides the language $L = \{ww^R \mid w \in \{a,b\}^*\}$ or "just" recognizes it? **This TM decides L.**

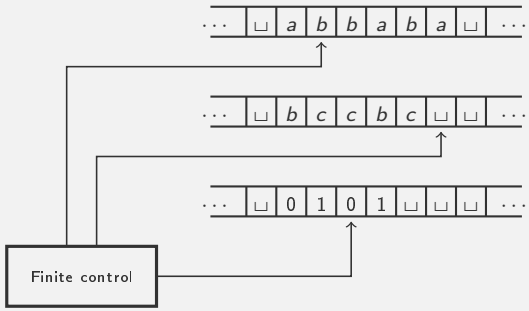
Is there a TM, which recognizes L but does not decide it? **Yes**, change the transitions going to q_r by redirecting them to a new state with an infinite loop, i.e., no transition going out of that state.

Fundamentals of theory of computation 2

4th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Multitape Turing machines



- Reading k tape symbols and the current state the TM moves to a new state, rewrites the k tape symbols and each of the k tape heads move to neighboring cells (or stay).
- The concept of accepting a word is analogous to that of 1-tape TM's.
- The concepts of running time and time complexity are analogous to those of 1-tape TM's.

Multitape Turing machines

Definition

A k-tape TM is a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ where

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

Definition

$(q, u_1, v_1, \ldots, u_k, v_k)$ is called a configuration of a k-tape TM, where $q \in Q$ and $u_i, v_i \in \Gamma^*$, $v_i \neq \varepsilon$ $(1 \leqslant i \leqslant k)$.

Multitape Turing machines

Configuration is a finite representation of the machine at a given time. It represents the current state q, the content of the ith tape $u_i v_i$, and the position of the ith head as the first letter of v_i $(1 \leqslant i \leqslant k)$.

Definition

Starting configuration of the word u is $u_i = \varepsilon$ $(1 \leqslant i \leqslant k)$, $v_1 = u\sqcup$, and $v_i = \sqcup$ $(2 \leqslant i \leqslant k)$.

[Why v_1 is defined to be $u\sqcup$ and not u? To avoid $u = \varepsilon$ being another case. The two words represent the same tape content.]

Definition

For a configuration $(q, u_1, v_1, \ldots, u_k, v_k)$ where $q \in Q$ and $u_i, v_i \in \Gamma^*$, $v_i \neq \varepsilon$ $(1 \leqslant i \leqslant k)$, it is an accepting configuration if $q = q_a$, rejecting configuration, if $q = q_r$, halting configuration, if $q = q_a$ or $q = q_r$.

Multitape Turing machines

Definition

For a k-tape Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_i, q_n \rangle$ let us introduce the **one-step transition relation** $\vdash \subseteq C_M \times C_M$ as follows

Let $C = (q, u_1, a_1 v_1, \ldots, u_k, a_k v_k)$ be a configuration, where $a_i \in \Gamma$, $u_i, v_i \in \Gamma^*$ $(1 \leqslant i \leqslant k)$. Let furthermore $\delta(q, a_1, \ldots, a_k) = (r, b_1, \ldots, b_k, D_1, \ldots, D_k)$ be a transition, where $q, r \in Q$, $b_i \in \Gamma, D_i \in \{L, S, R\}$ $(1 \leqslant i \leqslant k)$. Then $C \vdash (r, u_1', v_1', \ldots, u_k', v_k')$, where for every tape $1 \leqslant i \leqslant k$

- if $D_i = R$ then $u_i' = u_i b_i$ and $v_i' = v_i$ in the case of $v_i \neq \varepsilon$, otherwise $v_i' = \sqcup$,
- if $D_i = S$ then $u_i' = u_i$ and $v_i' = b_i v_i$,
- if $D_i = L$ then $u_i = u_i' c$ $(c \in \Gamma)$ and $v_i' = c b_i v_i$ in the case of $u_i \neq \varepsilon$, otherwise $u_i' = \varepsilon$ és $v_i' = \sqcup b_i v_i$.

Multitape Turing machines

So by the definitions $C \vdash C'$ holds for configurations C, C' if we can get C' from C by following the instructions of δ for every tape.

Example:

Let k=2 and $\delta(q, a_1, a_2) = (r, b_1, b_2, R, S)$ be a transition of a TM. Then $(q, u_1, a_1 v_1, u_2, a_2 v_2) \vdash (r, u_1 b_1, v_1', u_2, b_2 v_2)$, where $v_1' = v_1$, if $v_1 \neq \varepsilon$, otherwise $v_1' = \sqcup$.

Notice, that the heads can move independently of each other.

Definition

Multistep transition relation (a configuration yields an other one in finitely many steps) is formally defined the same way as we did it for one-tape TM's. Notation: $\vdash^*$.

Multitape Turing machines

Definition

The **language recognized by a k-tape Turing machine** $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ is the following:
$L(M) = \{u \in \Sigma^* \mid (q_0, \varepsilon, u\sqcup, \varepsilon, \sqcup, \ldots, \varepsilon, \sqcup) \vdash^*$
$(q_a, x_1, y_1, \ldots x_k, y_k),\ x_1, y_1 \ldots, x_k, y_k \in \Gamma^*, y_1, \ldots, , y_k \neq \varepsilon\}$.

As in the case of one tape TM's multitape TM's are accepting those words for which the TM can reach its accepting state q_a.

Languages that can be **recognized** or can be **decided** by a multitape TM is defined the same way as for one-tape TM's.

Definition

Running time of word u on TM M is the number of computational steps from the starting configuration of u to a halting configuration.

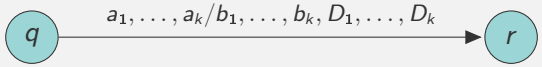
Time complexity of a multitape TM is defined the same way as we did for 1-tape TM's.

Multitape Turing machines

An Example

Exercise: Construct a 2-tape TM M having $L(M) = \{ww^R \mid w \in \{a, b\}^*\}$.

Transition diagram is a vertex and edge labelled directed graph, where



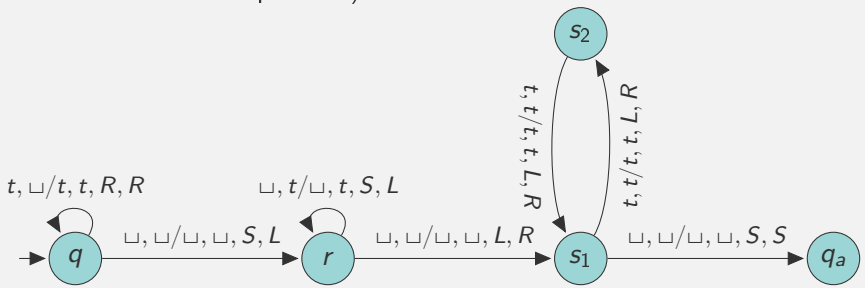
$$\delta(q, a_1, \ldots, a_k) = (r, b_1, \ldots, b_k, D_1, \ldots, D_k)$$
$$(q, r \in Q, a_1, \ldots, a_k, b_1, \ldots, b_k \in \Gamma, D_1, \ldots, D_k \in \{L, S, R\})$$

Multitape Turing machines

An Example

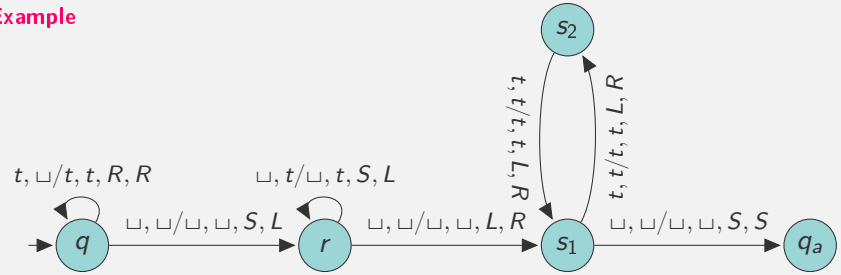
Exercise: Construct a 2-tape TM M having $L(M) = \{ww^R \mid w \in \{a, b\}^*\}$.

Solution:
(All other transitions go to q_r. Let $t \in \{a, b\}$ be arbitrary for the transitions where t is present.)



Multitape Turing machines

An Example



$(q, \varepsilon, abba, \varepsilon, \sqcup) \vdash (q, a, bba, a, \sqcup) \vdash (q, ab, ba, ab, \sqcup) \vdash$
$(q, abb, a, abb, \sqcup) \vdash (q, abba, \sqcup, abba, \sqcup) \vdash (r, abba, \sqcup, abb, a) \vdash$
$(r, abba, \sqcup, ab, ba) \vdash (r, abba, \sqcup, a, bba) \vdash (r, abba, \sqcup, \varepsilon, abba) \vdash$
$(r, abba, \sqcup, \varepsilon, \sqcup abba) \vdash (s_1, abb, a, \varepsilon, abba) \vdash (s_2, ab, ba, a, bba) \vdash$
$(s_1, a, bba, ab, ba) \vdash (s_2, \varepsilon, abba, abb, a) \vdash$
$(s_1, \varepsilon, \sqcup abba, abba, \sqcup) \vdash (q_i, \varepsilon, \sqcup abba, abba, \sqcup)$

What is the time complexity of M? It is a $3n + 3 = O(n)$ time TM.

Multitape Turing machines – 1-tape simulation

Definition

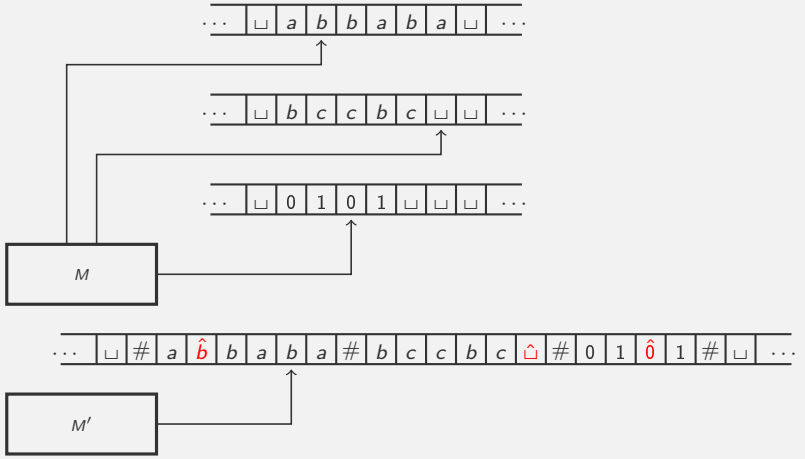
Two TM's are considered to be **equivalent**, if they recognize the same language.

Theorem

For any k-tape TM M there is an equivalent 1-tape TM M'. Furthermore, if M has time complexity $f(n)$, which is at least linear, i.e, $f(n) = \Omega(n)$, then M' has time complexity $O(f(n)^2)$.

Multitape Turing machines – 1-tape simulation

Proof (sketch): Main idea of the simulation



Multitape Turing machines – 1-tape simulation

Steps of the simulation on input $a_1 \cdots a_n$:

1. Let the starting configuration M' be $q_0' \# \hat{a}_1 a_2 \cdots a_n \# \hat{\sqcup} \# \cdots \hat{\sqcup} \#$
2. As M' scans its tape for the first time counts the #'s and stores symbols denoted by a $\hat{}$ in its states. (E.g., for the case on the figure, M is in state q then M' changes its state from (q) to (q, b), to $(q, b, \sqcup)$ and finishing in $(q, b, \sqcup, 0)$.)
3. M' goes through its tape for the another time updating it's content according to its transition function.
4. if the length of the content on a tape of M increases M' shifts it's content by 1 cell to make room for the new letter (for $\sqcup$ in fact). This can be done in O(# letters to be moved).
5. If M reaches its accepting or rejecting state so does M'.
6. Otherwise M' continues with step 2.

Multitape Turing machines – 1-tape simulation

The following holds for simulating a single step of M:

- Used up space (number of used cells) is an asymptotic upper bound for the number of steps M' takes. (Goes through its content twice, it needs to make space for a $\sqcup$ at most k times which is O(used up space))
- used up space is increased by $O(1)$ cells. (by $\leqslant k$, in fact)

In the beginning M' uses $\Theta(n)$ cells. In each step the used up space is increeased by $O(1)$, so $O(n + f(n)O(1)) = O(n + f(n))$ is a common asymptotic upper bound for the used up cells after each step of the simulation.

So $O(n + f(n))$ is a general asympotic upper bound for the time complexity of a single step.

Altogether M' has a time complexity of $f(n) \cdot O(n + f(n))$, which is $O(f(n)^2)$, if $f(n) = \Omega(n)$.

Nondeterministic Turing machines

Definition

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

- $Q, \Sigma, \Gamma, q_0, q_a, q_r$ are the same as for deterministic TM's
- $\delta : (Q \backslash \{q_a, q_r\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, S, R\})$

Let C_M be the set of all possible configurations for a NTM M.

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

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

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

Nondeterministic Turing machines

Multistep transition relation, denoted by $\vdash^*$, is the reflexive, transitive closure of one-step transition relation $\vdash$. I.e.,

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

Let $C, C' \in C_M$ be configurations of a NTM M. $C \vdash^* C' \Leftrightarrow$

- $C = C'$ or
- $\exists\, n > 0 \wedge C_1, C_2, \ldots \in C_n \in C_M$, such that $\forall\, 1 \leqslant i \leqslant n-1$ $C_i \vdash C_{i+1}$ holds. Furthermore $C_1 = C$ and $C_n = C'$.

If $C \vdash^* C'$ we say that C **yields** C' **in finitely many steps**

NTM's may have several computations for the same word. It accepts a word if and only if it has at least one computation ending in q_a.

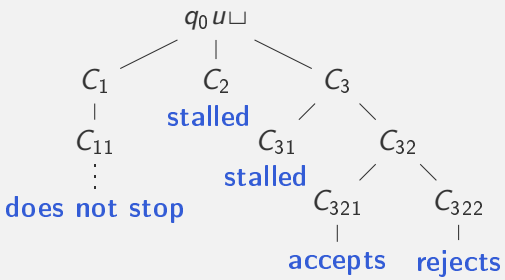
Example: Let $\delta(q_2, a) = \{(q_5, b, L), (q_1, d, R)\}$ and $C_1 = bcq_2a\sqcup b$, $C_2 = bq_5cb\sqcup b$, $C_3 = bcdq_1\sqcup b$. Then $C_1 \vdash C_2$ and $C_1 \vdash C_3$. If we have another configuration C_4 with the property $C_2 \vdash C_4$. Then $C_1 \vdash^* C_1$, $C_1 \vdash^* C_2$, $C_1 \vdash^* C_3$ and $C_1 \vdash^* C_4$.

Nondeterministic Turing machines

Definition

The **nondeterministic configuration tree** for $u \in \Sigma^*$ is a vertex labelled directed tree with the following properties. The root has label $q_0 u \sqcup$. If C is a label of a node, then it has $|\{C' \mid C \vdash C'\}|$ children labelled by the elements of $\{C' \mid C \vdash C'\}$.

Example:



This TM accepts u since $q_0 u \sqcup \vdash C_3 \vdash C_{32} \vdash C_{321}$ is a computation leading to an accepting configuration. Only one accepting computation is needed to accept a word.

Nondeterministic Turing machines

Definition

The **language recognized by a NTM** $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ is $L(M) = \{u \in \Sigma^* \mid q_0 u \sqcup \vdash^* x q_a y \text{ for some } x, y \in \Gamma^*, y \neq \varepsilon\}$.

Definition

A NTM M **recognizes** L if $L(M) = L$. A NTM M **decides** a language L if M recognizes L and for all $u \in \Sigma^*$ the configuration tree has a finite height and all leaves are halting configurations.

Definition

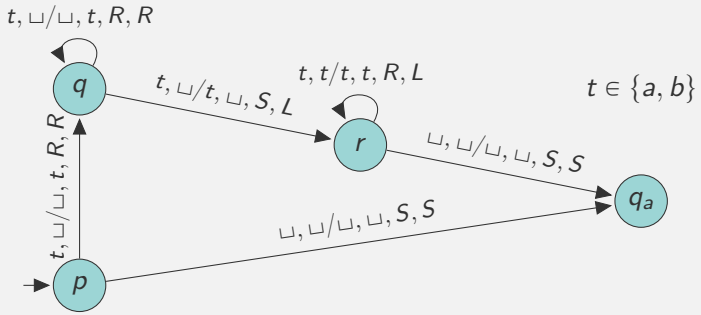
M has **time complexity** $f(n)$, if for all $u \in \Sigma^*$ of length n the height of the configuration tree is at most $f(n)$.

Remark: Definition of a multitape NTM is analogous to that of previous definitions.

Nondeterministic TM

Example

Excercise: Construct a NTM M with $L(M) = \{ww^R \mid w \in \{a, b\}^*\}$.



$(p, \varepsilon, abba, \varepsilon, \sqcup) \vdash (q, \varepsilon, bba, a, \sqcup) \vdash (r, \varepsilon, bba, \varepsilon, a) \vdash (q_r, \varepsilon, bba, \varepsilon, a)$

$(p, \varepsilon, abba, \varepsilon, \sqcup) \vdash (q, \varepsilon, bba, a, \sqcup) \vdash (q, \varepsilon, ba, ab, \sqcup) \vdash (r, \varepsilon, ba, a, b) \vdash (r, b, a, \varepsilon, ab) \vdash (r, ba, \sqcup, \varepsilon, \sqcup ab) \vdash (q_a, ba, \sqcup, \varepsilon, \sqcup ab)$

Nondeterministic TM

Simulating NTM's by deterministic TM's

Theorem

For all $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ NTM's of time complexity $f(n)$ there is an equivalent deterministic TM of time complexity $2^{O(f(n))}$.

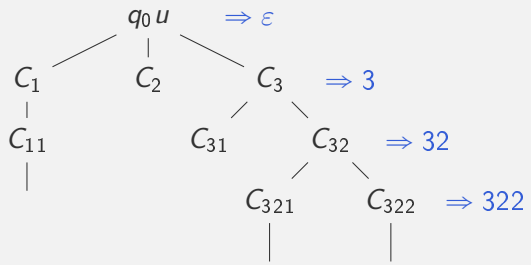
Proof (sketch): M' simulates all partial computations for an input $u \in \Sigma^*$ by doing a breadth first search on its configuration tree.

- Let d be the following number. $d = \max_{(q,a) \in Q \times \Gamma} |\delta(q, a)|$.
- Let $T = \{1, 2, \ldots, d\}$ be an alphabet.
- for all $(q, a) \in Q \times \Gamma$ let us fix an order of the set $\delta(q, a)$

Nondeterministic TM

Simulating NTM's by deterministic TM's

For each node of the configuration tree one can associate a unique word over T, the **selector** of that partial computation.



Shortlex order

Definition

Let $X = \{x_1 \prec x_2 \prec \cdots \prec x_s\}$ be an ordered alphabet. The short-lexicographic (shortlex) order of X^* is the following order, denoted by $\prec_{\text{shortlex}}$. For every $u_1 \cdots u_n, v_1 \cdots v_m \in X^*$ let $u_1 \cdots u_n \prec_{\text{shortlex}} v_1 \cdots v_m \Leftrightarrow (n < m) \vee ((n = m) \wedge (u_k < v_k)$, where k is the smallest i having $u_i \neq v_i$).

Example 1 Let $X = \{a, b\}$ and $a \prec b$, then the shortlex order of X^* is

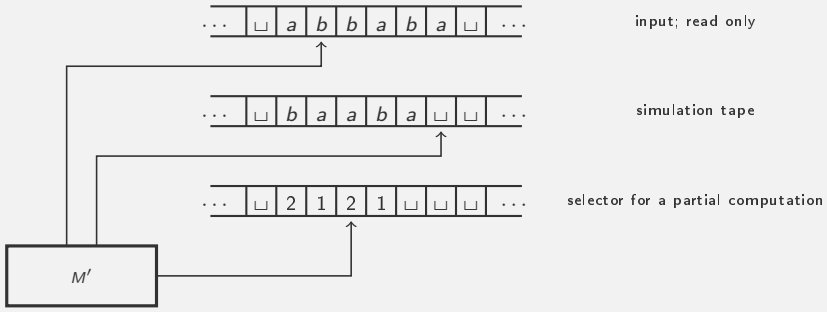
$\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots$

Example 2 Consider natural numbers as finite sequences of digits (no extra opening 0's, except for the number 0 itself).

Then $n < m$ if and only if $n \prec_{\text{shortlex}} m$ for the ordered alphabet $X = \{0 \prec 1 \prec 2 \prec 3 \prec 4 \prec 5 \prec 6 \prec 7 \prec 8 \prec 9\}$.

Nondeterministic TM

Simulating NTM's by deterministic TM's



How does M' work?

Nondeterministic TM

- starting configuration of M': input on 1st tape, the other tapes are empty
- WHILE there's no accept
  - copy the content of tape 1 of M' to tape 2
  - WHILE the head of the 3rd tape does not read $\sqcup$
    - Let k be the current letter on tape 3
    - Let a be the current letter on tape 2 and q be the current state of M
    - if $\delta(q, a)$ has a kth element, then
      - M' simulates one step of M according to this
      - if this leads to q_a of M, then M' accepts
      - if this leads to q_r of M, then abort loop

      else ($\delta(q, a)$ has no kth element) abort loop
    - M' moves one cell to the right on the 3rd tape
  - M' deletes the content of tape 2 and creates the next word on tape 3 according to shortlex order over T.

Nondeterministic TM

Simulating NTM's by deterministic TM's

- M' goes to it accepting state if and only if M do so, so they are equivalent TM's,
- the number of partial computations of length at most $f(n)$ is at most the number of nodes of complete d-ary tree, i.e. at most
$$\sum_{i=0}^{f(n)} d^i = \frac{d^{f(n)+1}-1}{d-1} = O(d^{f(n)}),$$
- for every partial computation M' takes $O(n+f(n))$ steps,
- altogether the time complexity of M' is $O(n+f(n))O(d^{f(n)}) = 2^{O(f(n))}$.

Remarks:

- The above simulation has exponential time complexity. Other simulations may do better.
- Conjecture: there's no efficient (polynomial) simulation of a NTM by a deterministic one.

Fundamentals of theory of computation 2

5th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Cardinality

An important property of finite sets is their size. ($\Rightarrow$ *natural numbers*). Goal: find a generalization for infinite sets. One such generalization is **cardinality** *(G. Cantor, 1845-1918)*.

Definition

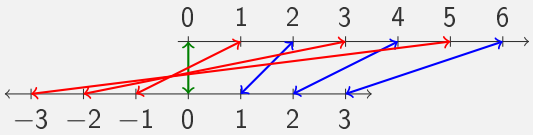
- Sets A and B have the same cardinality, if there's a bijection between them. Notation: $|A| = |B|$.
- The cardinality of A is greater or equal to the cardinality of B if there's an injective mapping from B to A. Notation: $|A| \geqslant |B|$.
- The cardinality of A is greater than the cardinality of B if there is an injective mapping from B to A, but there is no bijection between them. Notation: $|A| > |B|$.

Cantor-Bernstein-Schröder Theorem

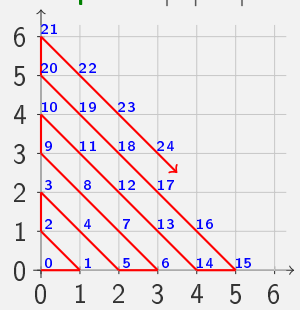
If $|A| \leqslant |B|$ and $|A| \geqslant |B|$, then $|A| = |B|$.

Cardinality

Example 1: $|\mathbb{N}| = |\mathbb{Z}|$.



Example 2: $|\mathbb{N}| = |\mathbb{N} \times \mathbb{N}|$.



Cardinality

Example 3: $|\mathbb{N}| = |\mathbb{Q}|$.

Proof:
$\mathbb{N} \subset \mathbb{Q}$, so $|\mathbb{N}| \leqslant |\mathbb{Q}|$.
$\mathbb{Q}^+ := \{\frac{p}{q} \mid p \in \mathbb{N}^+, q \in \mathbb{N}^+, \frac{p}{q}$ can not be simplified$\}$.
$\mathbb{Q}^- := \{-\frac{p}{q} \mid p \in \mathbb{N}^+, q \in \mathbb{N}^+, \frac{p}{q}$ can not be simplified$\}$.
$|\mathbb{Q}^+| = |\mathbb{Q}^-|$.
$\frac{p}{q} \in \mathbb{Q}^+ \mapsto (p, q) \in \mathbb{N} \times \mathbb{N}$ injective, so $|\mathbb{Q}^+| \leqslant |\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$.
Let $\mathbb{Q}^+ = \{a_1, a_2 \ldots, \}$, $\mathbb{Q}^- = \{b_1, b_2 \ldots, \}$, so
$\mathbb{Q} = \{0, a_1, b_1, a_2, b_2, \ldots\}$

Definition

The cardinality of $\mathbb{N}$ is called countably infinite. A set is countable if either it is finite or it is countably infinite.

In other words, set A is countably infinite if there is a one-to-one correspondence between A and $\mathbb{N}$, i.e., the elements of A can be indexed by the natural numbers.

Cardinality

Proposition

Union of countably many countable sets is countable.

Proof: Same idea as in Example 2. □

Are there further cardinalities?

Yes, $|\mathbb{R}| > |\mathbb{N}|$.

Definition

The cardinality of $\mathbb{R}$ is called **continuum**.

Example 4: $|\mathbb{R}| = |(0,1)|$.

$\operatorname{tg}(\pi(x-\frac{1}{2}))\big|_{(0,1)} : (0,1) \to \mathbb{R}$ is a bijection between $(0,1)$ and $\mathbb{R}$.

Remark: $|\mathbb{R}| = |(a,b)| = |[c,d]|$ and $|\mathbb{R}| = |\mathbb{R}^n|$.

Cardinality

Example 5: $|\{0,1\}^*| = |\mathbb{N}|$.

Shortlex ordering provides a bijection:
$\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots$

Example 6:
$\left|\left\{L \,|\, L \subseteq \{0,1\}^*\right\}\right| = \left|\left\{(b_1, \ldots, b_i, \ldots) \,|\, b_i \in \{0,1\},\ i \in \mathbb{N}\right\}\right|$

Proof: There's a natural bijection between them.

Order binary words according to the shortlex order.

We can associate an arbitrary language L with a 0-1 sequence of length countably infinite as follows. Let the ith bit be 1 if and only if the ith word is an element of L, let it be 0 otherwise. It is easy to see that this mapping is injective and surjective. □

The sequence mapped to language L is called the **characterestic sequence** of L.

Notation: Let the set on the RHS be denoted by $\{0,1\}^{\mathbb{N}}$.

Cardinality

Example 7: $|\{0,1\}^{\mathbb{N}}| = |[0,1)|$.

Proof: (sketch)

We can associate for any $x \in [0,1)$ an infinte binary sequence, namely (one of) the sequence after "0." in the binary representation of x. This an injective mapping, so $|[0,1)| \leqslant |\{0,1\}^{\mathbb{N}}|$.

For a $z \in \{0,1\}^{\mathbb{N}}$ replace all occurences of 1 by 2, then write "0." before the sequence and consider the result of this transformation as a ternary representation of a real number from $[0,1)$. This mapping is injective (easy to see, that no two real number has two different ternary representations of only 0's and 2's), therefore $|\{0,1\}^{\mathbb{N}}| \leqslant |[0,1)|$.

According to the theorem of Cantor-Bernstein-Schröder we have $|\{0,1\}^{\mathbb{N}}| = |[0,1)|$. □

Cardinality – Cantor's diagonal method

Theorem

$|\mathbb{R}| > |\mathbb{N}|$.

Proof: $\mathbb{R} \supset \mathbb{N}$, so $|\mathbb{R}| \geqslant |\mathbb{N}|$.

It is enough to prove, that $|\{0,1\}^{\mathbb{N}}| > |\mathbb{N}|$, since the cardinality of $\{0,1\}^{\mathbb{N}}$ is continuum.

Assume, on the contrary, that $|\{0,1\}^{\mathbb{N}}| = |\mathbb{N}|$. This means, that there's a bijection between $\{0,1\}^{\mathbb{N}}$ and $\mathbb{N}$, so $\{0,1\}^{\mathbb{N}} = \{u_i \,|\, i \in \mathbb{N}\} = \{u_1, u_2, \ldots\}$ is an enumaration of $\{0,1\}^{\mathbb{N}}$.

Let $u_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,j}, \ldots)$, where $u_{i,j} \in \{0,1\}$ holds for all $i, j \in \mathbb{N}$, i.e., $u_{i,j}$ is the jth bit of the infinite sequence u_i.

Consider the countably infinite sequence $u = \{\overline{u_{1,1}}, \overline{u_{2,2}}, \ldots, \overline{u_{i,i}}, \ldots)$, i.e., $u \in \{0,1\}^{\mathbb{N}}$, where $\overline{b} = 0$, if $b = 1$ and $\overline{b} = 1$, if $b = 0$.

Cardinality – Cantors's diagonal method

Since all countably infinite 0-1 sequences are listed there should be a $k \in \mathbb{N}$, such that $u = u_k$.

On one hand, the kth bit of u_k equals to $u_{k,k}$ (by the definition of $u_{i,j}$), on the other hand, it is $\overline{u_{k,k}}$ (by the definition of u).

But this is impossible, so our assumption $|\{0,1\}^{\mathbb{N}}| = |\mathbb{N}|$ was incorrect, proving the theorem. □

So, continuum is a greater cardinality than countably infinite.

Since there are countably infinite words but continuum languages over $\{0,1\}$ we have the following.

Corollary

There are more languages than words over the alphabet $\{0,1\}$.

Remark: $\{L \mid L \subseteq \{0,1\}^*\} = \mathcal{P}(\{0,1\}^*)$. Is it true in general, that $|\mathcal{P}(H)| > |H|$ holds?

Cardinality

Theorem

$|\mathcal{P}(H)| > |H|$ holds for all sets H.

Proof: $|\mathcal{P}(H)| \geqslant |H|$, since $\{\{h\} \mid h \in H\} \subseteq \mathcal{P}(H)$.

$|\mathcal{P}(H)| \neq |H|$: by Cantor's diagonal method
Assume $f : \mathcal{P}(H) \leftrightarrow H$ is a bijection. Let us define a set $A \subseteq H$:
$\forall x \in H : \quad x :\in A \Leftrightarrow x \notin f^{-1}(x)$

Is it true that $f(A) \in A$? If yes, $f(A) \notin A$, if not $f(A) \in A$, so $f(A)$ is neither in A nor outside A which is a contradiction. So our initial assumption was false. □

Corollary

There are infinitely many infinite cardinalities.

$\aleph_0 := |\mathbb{N}|, \mathfrak{c} := |\mathcal{P}(\mathbb{N})| = |\mathbb{R}|$.

Solving computational problems by TM's

We can use TM's for solving **computational problems**, too.

Computational problems are generalizations of decision problems. For decision problems the output is either 'yes' or 'no'. For computational problems the output for an input x is $f(x)$, which is not neccessarily 'yes' or 'no'.

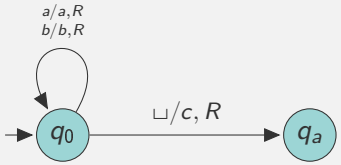
Definition

We say that a TM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, (q_r) \rangle$ **computes** the function $f : \Sigma^* \to \Delta^*$ if for all inputs $u \in \Sigma^*$ it halts, and $f(u) \in \Delta^*$ can be read on its **last** tape.

Remark: In the case of computational problems we do not have to distinguish between q_a and q_r, one halting state would be enough.

Solving computational problems by TM's

Example: $f(u) = uc \quad (u \in \{a,b\}^*)$.
$\Sigma = \{a,b\}, \ \Delta = \{a,b,c\}$



Problems as formal languages

If a problem has countable possible inputs we can code them over a finite alphabet.

How large should be the size of the alphabet? For an alphabet of size d we need words of length $\log_d n$ to code the first n words. Since $\log_d n = \Theta(\log_{d'} n)$ for $d, d' \geqslant 2$ the size of the alphabet does not really count.

Important! Never use unary codes for efficient representation. Unary representation of natural numbers is just drawing sticks.

For an input I let $\langle I \rangle$ denote the code of I.

Decision problems:
$L = \{\langle I \rangle \mid I \text{ is a 'yes' istance of the problem}\}$. Can L be decided by a TM?

Function problems (includes decision problems):
Is there a TM computing the function f, i.e., computing the function $\langle I \rangle \mapsto \langle f(I) \rangle$ for all possible inputs I.

Coding TM's

We may assume, that $\Sigma = \{0, 1\}$. Any set of inputs can be efficiently coded of Σ.

Definition

The code of a TM M (notation $\langle M \rangle$) is the following:
Let $M = (Q, \{0, 1\}, \Gamma, \delta, q_0, q_a, q_r)$, where

- $Q = \{p_1, \ldots, p_k\}$, $\Gamma = \{X_1, \ldots, X_m\}$, $D_1 = R$, $D_2 = S$, $D_3 = L$
- $k \geqslant 3$, $p_1 = q_0$, $p_{k-1} = q_a$, $p_k = q_r$,
- $m \geqslant 3$, $X_1 = 0$, $X_2 = 1$, $X_3 = \sqcup$.
- the code for a transition $\delta(p_i, X_j) = (p_r, X_s, D_t)$ is $0^i 1 0^j 1 0^r 1 0^s 1 0^t$.
- $\langle M \rangle$ is the concatenation of the codes of the transitions separated by 11's.

Observation: $\langle M \rangle$ always starts and ends with 0 and does not contain three consecutive 1's. $\langle M, w \rangle := \langle M \rangle 111 w$

Existance of a non-Turing-recognizable language

Notation: for all $i \geqslant 1$,

- Let w_i denote the ith element of $\{0, 1\}^*$ according to the shortlex ordering.
- Let M_i denote the TM defined by w_i (if w_i is not a code of a TM then let M_i be a TM accepting nothing)

Theorem

There exists a non-Turing-recognizable language.

Proof: Two different languages can not be recognized by the same TM. The cardinality of TM's is countably infinite (the above coding is an injection into a countable set $\{0, 1\}^*$). On the other hand, the number of languages over $\{0, 1\}$ has cardinality continuum. □

In fact, the 'majority' of the languages are unrecognizable by a TM. Can we give one specific unrecognizable language? Yes, the **diagonal language** $L_{\text{diag}} = \{w_i \mid w_i \notin L(M_i)\}$ is one of these.

The diagonal language is not Turing-recognizable

Theorem

$L_{\text{diag}} \notin RE$.

By Cantor's diagonal method:

Proof: Consider the following infinite table T of bits. It is of size $\mathbb{N}$ in both dimension. $T(i, j) := 1 \Leftrightarrow w_j \in L(M_i)$ $(i, j \geqslant 1)$.
Let z be the diagonal of T. Then z is string of countably infinite bits. $\bar{\mathsf{z}}$ is the bitwise complement of z. Then:

- for all $i \geqslant 1$ the ith row of T is characteristic sequence of $L(M_i)$.
- $\bar{\mathsf{z}}$ is the characteristic sequence of L_{diag}
- for all $L \in RE$ its characteristic sequence appears as a row of T
- $\bar{\mathsf{z}}$ is different from all rows of T
- so L_{diag} is different from all languages from RE □

The universal TM

Theorem

Universal language: $L_u = \{\langle M, w\rangle \mid w \in L(M)\}$.

Theorem

$L_u \in RE$

Proof: (sketch) We construct a "universal" 4-tape TM U which can simulate all TM's on each possible input.

1st tape: read only tape, U can always read the input, $\langle M, w\rangle$ here.

2nd tape: the current content of M's tape (coded as above)

3rd tape: the current state of M (coded as above)

4th tape: work tape

The universal TM

more details:

- Checks whether the input is of type $\langle M, w\rangle$. If not, it rejects the input.
- if yes, it copies w to its second tape, the code of q_0 to its 3rd tape
- Simulates a step of M:
  – Reads the current tape symbol on M's tape from its second tape
  – Reads the current state of M from its 3rd tape
  – Simulates a step of M (uses the 4th tape if neccessary) according to the description of M (can be read on tape 1).
- If M goes to its accepting/rejecting state, so does U. □

Remark: if M does not halt on w, then so does U for $\langle M, w\rangle$, so U "just" recognizes, but does not decides L_u.

Undecidability of the universal language

Theorem

$L_u \notin R$.

Proof: Assume on the contrary that there exists a TM M deciding L_u. Using M we construct a TM M' recognizing L_{diag}.

$w \in L(M') \Leftrightarrow w111w \notin L(M) \Leftrightarrow$ the TM coded by w does not accept $w \Leftrightarrow w \in L_{\text{diag}}$.
So $L(M') = L_{\text{diag}}$, contradiction. □

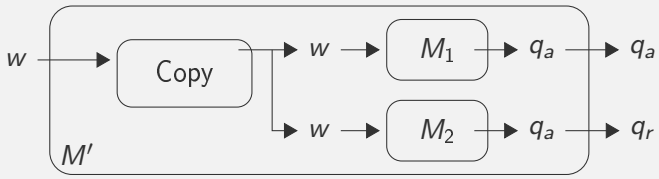
Properties of R and RE

Notation: For $L \subseteq \Sigma^*$, let $\bar{L} = \{u \in \Sigma^* \mid u \notin L\}$.

Theorem

If L and $\bar{L} \in RE$, then $L \in R$.

Proof: Let M_1 and M_2 be TM's recognizing L and $\bar{L}$ respectively. We construct a 2-tape TM M':



M' copies w to its second tape, then simulates M_1 and M_2 by switching between simulations step by step until one of them reaches its q_a.
So M' recognizes L, but also halts on every input, so $L \in R$. □

Properties of R and RE

Corollary

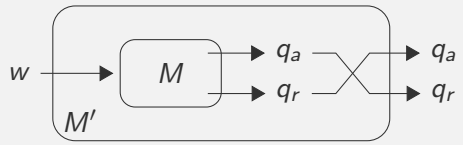
RE is not closed for the complement operation

Proof:
Let $L \in RE \backslash R$ (one such language is e.g., L_u). Then $\bar{L} \notin RE$, otherwise $\bar{L} \in RE$ but then $L \in R$ would follow, a contradiction. □

Theorem

R is closed for the complement operation.

Proof: Let $L \in R$ be a TM deciding M. Then M' decides $\bar{L}$:



□

Reduction

Definition

$f : \Sigma^* \to \Delta^*$ is **computable**, if there is a TM which computes it. [see TM's for computing functions]

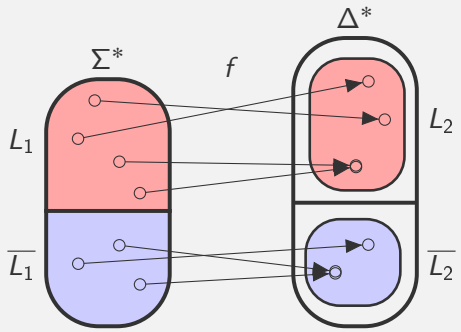
Definition

$L_1 \subseteq \Sigma^*$ is **reducible** to $L_2 \subseteq \Delta^*$ if there is a computable function $f : \Sigma^* \to \Delta^*$ such that $w \in L_1 \Leftrightarrow f(w) \in L_2$. Notation: $L_1 \leqslant L_2$

(Emil Post, 1944, many-one reducibility)

Reduction

$L_1 \leqslant L_2$



(1) f is a computable function (2) the domain of f is (the complete) Σ^* (3) $f(L_1) \subseteq L_2$, (4) $f(\overline{L_1}) \subseteq \overline{L_2}$.

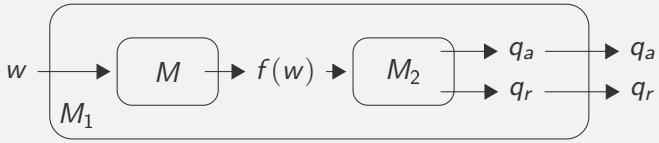
f does not need to be injective or surjective.

Theorem

- If $L_1 \leqslant L_2$ and $L_2 \in RE$, then $L_1 \in RE$.
- If $L_1 \leqslant L_2$ and $L_2 \in R$, then $L_1 \in R$.

Reduction

Proof:
If $L_2 \in RE$ (for 2nd statement: $\in R$) and $L_1 \leqslant L_2$, then there exists a TM M_2 recognizing (for 2nd statement: deciding) L_2 and a TM M computing the reduction. Construction of TM M_1:



Since $w \in L_1 \Leftrightarrow f(w) \in L_2 \Leftrightarrow f(w) \in L(M_2) \Leftrightarrow w \in L(M_1)$ we have that M_1 recognizes L_1, i.e., $L_1 \in RE$. In the case of the 2nd statement M_1 decides L_1 holds, too, therefore $L_1 \in R$. □

Corollary

- If $L_1 \leqslant L_2$ and $L_1 \notin RE$, then $L_2 \notin RE$.
- If $L_1 \leqslant L_2$ and $L_1 \notin R$, then $L_2 \notin R$.

The halting problem of TM's

Halting problem:

Definition

Halting language: $L_h = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$.

Observation: $L_u \subseteq L_h$
Is it true? $A \subseteq B$, and A is undecidable. Is B undecidable as well?
No.

Theorem

$L_h \notin R$.

Proof: It is enough to show that $L_u \leqslant L_h$.
For an arbitrary TM M let M' be the following. M' does the following for an arbitrary input u:

1. It runs M on u.
2. If M reaches q_a, then M' goes into its own q_a.
3. If M reaches q_r, then M' goes into an infite loop.

The halting problem of TM's

The following can be observed.

- $f : \langle M, w\rangle \to \langle M', w\rangle$ is a computable function
- for an arbitrary (TM,input) pair (M, w):
  $\langle M, w\rangle \in L_u \Leftrightarrow M$ accepts $w \Leftrightarrow M'$ halts on $w \Leftrightarrow$
  $\langle M', w\rangle \in L_h$

So the construction of M' gives a reduction of L_u to L_h. So $L_h \notin R$. □

Remark: In the proof of f being a reduction we've focused on the images of the 'interesting' objects only.
I.e., on words that are codes of a 2-tuple (TM, input). We can extend the function f for all words over $\{0, 1\}$ as follows.

$$f(x) = \begin{cases} \langle M', w\rangle & \text{if } x = \langle M, w\rangle \text{ for some TM } M \text{ and word } w, \\ \varepsilon & \text{otherwise.} \end{cases}$$

$(x \in \{0, 1\}^*)$

The halting problem of TM's

Theorem

$L_h \in RE$.

Proof: It's enough to show that $L_h \leqslant L_u$. For a TM M let M' be the following TM: M' works on an input u as follows:

1. It runs M on u.
2. If M reaches q_a, then M' goes to its own q_a.
3. if M reaches q_r, then M' goes to its own q_a.

The following can be observed.

- $f : \langle M\rangle \to \langle M'\rangle$ is a computable function
- for an arbitrary (TM,input) pair (M, w): $\langle M, w\rangle \in L_h \Leftrightarrow M$ halts on $w \Leftrightarrow M'$ accepts $w \Leftrightarrow \langle M', w\rangle \in L_u$

So the construction of M' is a reduction of L_h to L_u. We are done due to the previous theorems. □

Fundamentals of theory of computation 2

6th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Rice's theorem

Definition

- $\mathcal{P} \subseteq RE$ is called a **property** of recursively enumerable languages.
- A property is called **trivial** if $\mathcal{P} = \varnothing$ or $\mathcal{P} = RE$.
- $L_{\mathcal{P}} = \{\langle M \rangle \mid L(M) \in \mathcal{P}\}$.

Rice's theorem

$L_{\mathcal{P}} \notin R$ holds for all non-trivial properties $\mathcal{P} \subseteq RE$.

Rice's theorem

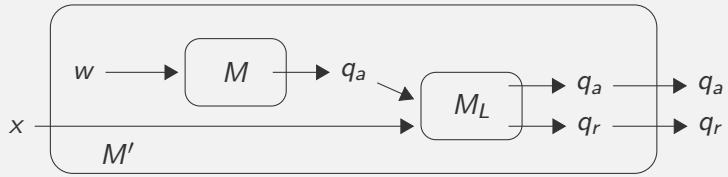
Proof:

Case 1: $\varnothing \notin \mathcal{P}$.
Since $L_u \notin R$, it is enough to prove, that $L_u \leqslant L_{\mathcal{P}}$.
As $\mathcal{P}$ is a non-trivial property, there exists $L \in \mathcal{P}$. $(L \neq \varnothing)$.
$L \in RE$, so there is a TM M_L satisfying $L(M_L) = L$.

For an arbitrary (TM,input) 2-tuple $\langle M, w \rangle$ we construct a 2-tape TM M'. M' works as follows for an arbitrary input x.

1. First, let M' run M for w on its 2nd tape.
2. If M does not halt for w, then so does M' for any input. Therefore we have $L(M') = \varnothing$ in this case.
3. If M rejects w-t, then let M' go to its rejecting state. So M' rejects every input and we have $L(M') = \varnothing$ in this case, too.
4. If M accepts w, then let M' run M_L for x. By the definition of M_L we have $L(M') = L$ in this case.

Rice's theorem



Summary

- $\langle M, w \rangle \in L_u \Rightarrow L(M') = L \Rightarrow L(M') \in \mathcal{P} \Rightarrow \langle M' \rangle \in L_{\mathcal{P}}$.
- $\langle M, w \rangle \notin L_u \Rightarrow L(M') = \varnothing \Rightarrow L(M') \notin \mathcal{P} \Rightarrow \langle M' \rangle \notin L_{\mathcal{P}}$.

Therefore
$\langle M, w \rangle \in L_u \Leftrightarrow \langle M' \rangle \in L_{\mathcal{P}}$. So $L_u \leqslant L_{\mathcal{P}}$ holds and we proved that $L_{\mathcal{P}} \notin R$ for $\varnothing \notin \mathcal{P}$.

Rice's theorem

Case 2: $\varnothing \in \mathcal{P}$.

- Since $\overline{\mathcal{P}} = \text{RE} \setminus \mathcal{P}$ in non-trivial and $\varnothing \notin \overline{\mathcal{P}}$ all conditions of case 1 holds for $\overline{\mathcal{P}}$. Since case 1 is already proved, we can apply Rice's theorem for $\overline{\mathcal{P}}$.
- So, we have $L_{\overline{\mathcal{P}}} \notin R$.
- $\overline{L_{\mathcal{P}}}$: those words over $\{0,1\}$, that are not a code of any TM M, such that $L(M)$ has property $\mathcal{P}$.
  $L_{\overline{\mathcal{P}}}$: the codes of such TM's M, that $L(M)$ does not have property $\mathcal{P}$.

  But according to our earlier agreement every word over $\{0,1\}$ is a code of a TM, the words that are not of the form of a code of a TM correspond to a TM accepting no words. Therefore $\overline{L_{\mathcal{P}}} = L_{\overline{\mathcal{P}}}$.
- $\overline{L_{\mathcal{P}}} \notin R \Rightarrow L_{\mathcal{P}} \notin R$ (according to a theorem from the previous lecture). □

Applications of Rice's theorem

Corollary

It's undecidable whether a TM M recognises

- the empty language ($\mathcal{P} = \{\varnothing\}$)
- a finite language ($\mathcal{P} = \{L \mid L \text{ is finite }\}$)
- a CF language ($\mathcal{P} = \{L \mid L \text{ context-free }\}$)
- a language containing ε ($\mathcal{P} = \{L \in RE \mid \varepsilon \in L\}$)
- ...

Post Correspondence Problem

Definition

Let $u_1, \ldots, u_n, v_1 \ldots, v_n \in \Sigma^+$ $(n \geqslant 1)$. A set $D = \left\{ \frac{u_1}{v_1}, \ldots, \frac{u_n}{v_n} \right\}$ of pairs of words is called a **set of dominoes**.

More precisely, the ith domino is a 2-tuple (u_i, v_i). We call u_i the top word, v_i the bottom word of the ith domino.

Definition

A sequence of dominoes $\frac{u_{i_1}}{v_{i_1}} \cdots \frac{u_{i_m}}{v_{i_m}}$ $(m \geqslant 1, 1 \leqslant i_1, \ldots, i_m \leqslant n)$ is called a **solution** for $D = \left\{ \frac{u_1}{v_1}, \ldots, \frac{u_n}{v_n} \right\}$ if $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$ holds.

Post Correspondence Problem

Example: One possible solution for $\left\{ \frac{b}{ca}, \frac{dd}{e}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c} \right\}$ is $\frac{a}{ab}\frac{b}{ca}\frac{ca}{a}\frac{a}{ab}\frac{abc}{c}$.

Another solution: $\frac{a}{ab}\frac{b}{ca}\frac{ca}{a}\frac{a}{ab}\frac{abc}{c}\frac{a}{ab}\frac{b}{ca}\frac{ca}{a}\frac{a}{ab}\frac{abc}{c}$.

Remark: So elements of a set of dominoes can be used multiple times in a solution and it is allowed not to use one ore more of the dominoes. There can be multiple solutions of the same set.

Solution can have arbitrary (but finite) length.

Notice, that despite of the fact that the set of dominoes is finite the search space, i.e. the set of possible solutions is infinite.

Post Correspondence Problem (PCP):

$L_{\text{PCP}} = \{\langle D \rangle \mid D \text{ has a solution}\}$.

Post Correspondence Problem

Theorem

$L_{PCP} \in RE$.

Proof: Observe, that possible solutions for D correspond to the elements of D^*. Make a TM M, which processes all words over D in the shortlex order. For a word w over D let M check whether the order of dominoes according to w gives a solution or not. In the first case let M halt in its accepting state q_a, otherwise the next word is processed.
So, M halts in q_a if and only if D has a solution. □

Theorem

$L_{PCP} \notin R$.

Proof:
Let us define a version of PCP called, MPCP. The yes-instances of MPCP are (set of dominoes, domino) pairs (D, d), where D has a solution starting with the domino d.

Post Correspondence Problem

$L_{MPCP} = \{\langle D, d\rangle \mid d \in D \wedge D \text{ has a solution starting with } d\}$.

First, we show that $L_{MPCP} \leqslant L_{PCP}$.

For $u = a_1 \cdots a_n \in \Sigma^+$ and $* \notin \Sigma$ let us introduce the notations:

$\text{leftstar}(u) := * a_1 * a_2 \cdots * a_n$
$\text{rightstar}(u) := a_1 * a_2 * \cdots a_n *$.
$\text{bothstar}(u) := * a_1 * a_2 * \cdots a_n *$.

Let $D = \{d_1, \ldots, d_n\}$ be a set of dominoes, where $d_i = \frac{u_i}{v_i}$ $(1 \leqslant i \leqslant n)$.

Let D' be the following set of dominoes of size $|D| + 2$ $(\# \notin \Sigma)$:

$$d'_i = \frac{\text{leftstar}(u_i)}{\text{rightstar}(v_i)} \quad (1 \leqslant i \leqslant n),$$

$$d'_0 = \frac{\text{leftstar}(u_1)}{\text{bothstar}(v_1)}, \quad d'_{n+1} = \frac{* \#}{\#}$$

.

Post Correspondence Problem

Example: If

$$D = \left\{ \frac{ab}{a}, \quad \frac{c}{bc} \right\},$$

then

$$D' = \left\{ \frac{* a * b}{* a *}, \quad \frac{* a * b}{a *}, \quad \frac{* c}{b * c *}, \quad \frac{* \#}{\#} \right\}$$

Claim: $\langle D, d_1\rangle \in L_{MPCP} \iff \langle D'\rangle \in L_{PCP}$.

Proof of the claim:

- if $d_{i_1} \cdots d_{i_m}$ is a solution for input (D, d_1) of MPCP, then $d'_0 d'_{i_2} \cdots d'_{i_m} d'_{n+1}$ is a solution for input D' of PCP.
- if $d'_{i_1} \cdots d'_{i_m}$ is a solution for input D' of PCP, then due to the matching constraint of the first and the last letters this can be a solution only if $d'_{i_1} = d'_0$ and $d'_{i_m} = d'_{n+1}$ holds. This gives a solution $d_{i_1} \cdots d_{i_{m-1}}$ for input (D, d_1) of MPCP.

So the claim holds. Mapping $D \mapsto D'$ is computable by a TM implying $L_{MPCP} \leqslant L_{PCP}$.

Post Correspondence Problem

Second, we show that $L_u \leqslant L_{MPCP}$.

For any $\langle M, w\rangle$ (TM, word) 2-tuple we associate a (set of dominoes, starting domino) 2-tuple $\langle D, d\rangle$ in such a way, that

$w \in L(M) \iff D$ has a solution starting with d.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ and $w = a_1 \cdots a_n \in \Sigma^*$.
Construction of (D, d):

- $d := \frac{\#}{\#q_0 a_1 \cdots a_n \#}$ (where $\# \notin \Sigma$) $\quad d :\in D$
- – if $\delta(p, a) = (q, b, R)$, then $\frac{pa}{bq} :\in D$
  – if $\delta(p, a) = (q, b, L)$, then $(\forall c \in \Gamma :)\ \frac{cpa}{qcb} :\in D$
  – if $\delta(p, a) = (q, b, S)$, then $\frac{pa}{qb} :\in D$
- $(\forall a \in \Gamma :)\ \frac{a}{a} :\in D$
- $\frac{\#}{\#}$, $\frac{\#}{\sqcup\#}$, $\frac{\#}{\#\sqcup} :\in D$
- $(\forall a \in \Gamma :)\ \frac{aq_a}{q_a}$, $\frac{q_a a}{q_a} :\in D$
- $\frac{q_a \# \#}{\#} :\in D$.

Post Correspondence Problem

Example:

If M has two transitions $\delta(q_0, b) = (q_2, a, R)$ and $\delta(q_2, a) = (q_a, b, S)$, then $q_0bab \vdash aq_2ab \vdash aq_abb$ is a computation of M accepting bab.

The set of dominoes for $\langle M, bab\rangle$ contains among others $\frac{\#}{\#q_0bab\#}$ starting domino, $\frac{q_0b}{aq_2}$ and $\frac{q_2a}{q_ab}$ for the transitions , $\frac{a}{a}$, $\frac{b}{b}$, $\frac{\sqcup}{\sqcup}$ and $\frac{\#}{\#}$ identical dominoes furthermore $\frac{aq_a}{q_a}$, $\frac{q_ab}{q_a}$ and $\frac{q_a\#\#}{\#}$.

A solution (| separates blocks):

$$\frac{\#}{\#q_0bab\#} \left| \frac{q_0b}{aq_2}\frac{a}{a}\frac{b}{b}\frac{\#}{\#} \right| \frac{a}{a}\frac{q_2a}{q_ab}\frac{b}{b}\frac{\#}{\#} \left| \frac{aq_a}{q_a}\frac{b}{b}\frac{b}{b}\frac{\#}{\#} \right| \frac{q_ab}{q_a}\frac{b}{b}\frac{\#}{\#} \left| \frac{q_ab}{q_a}\frac{\#}{\#} \right| \frac{q_a\#\#}{\#}$$

Post Correspondence Problem

$$\frac{\#}{\#q_0bab\#} \left| \frac{q_0b}{aq_2}\frac{a}{a}\frac{b}{b}\frac{\#}{\#} \right| \frac{a}{a}\frac{q_2a}{q_ab}\frac{b}{b}\frac{\#}{\#} \left| \frac{aq_a}{q_a}\frac{b}{b}\frac{b}{b}\frac{\#}{\#} \right| \frac{q_ab}{q_a}\frac{b}{b}\frac{\#}{\#} \left| \frac{q_ab}{q_a}\frac{\#}{\#} \right| \frac{q_a\#\#}{\#}$$

We demonstrate by this example, that $w \in L(M)$ implies a solution for $\langle D, d\rangle$.

The first block contains just the starting domino $d = \frac{\#}{\#bab\#}$.

In the next two blocks there are configurations of the TM both on the upper and the lower side. The upper side is "dropped behind" by one configuration.

In blocks 4 to 6 the upper side "makes up the lagging" letter by letter by the dominos of type $\frac{aq_a}{q_a}$ (and $\frac{q_aa}{q_a}$) till the difference is only $q_a\#$.

Finally, in the last block there is only one domino, making up the remaining lagging.

A solution can be constructed following this example for an arbitrary $w \in L(M)$. Therefore we have $w \in L(M) \Rightarrow \exists$ a solution of $\langle D, d\rangle$.

Post Correspondence Problem

On the other hand, if there is a solution starting with d then this solution must contain at least one domino containing the letter q_a in the lower word.

This is due to to the fact that in any solution the upper and lower words should be the same implying, that they should be of the same length as well. But the only dominoes with a longer top word are the dominoes containing q_a. To have a match q_a should appear least once in the bottom word of a domino of a solution.

But one can prove that the only way to have q_a in the lower word of a solution is to have a sequence of configurations from the starting configuration to a configuration containing q_a, i.e. an accepting configuration separated by #'s. This yields $w \in L(M)$.

$\langle D, d\rangle$ is computable from $\langle M, w\rangle$ by a TM, so we have $L_u \leqslant L_{\text{MPCP}}$.

Post Correspondence Problem

Claim: Reduction is transitive.

Proof of the claim: Let $L_i \subseteq \Sigma_i^*$ $(i = 1, 2, 3)$, $L_1 \leqslant L_2$ and $L_2 \leqslant L_3$. Furthermore, let $f : \Sigma_1^* \to \Sigma_2^*$ and $g : \Sigma_2^* \to \Sigma_3^*$ be computable functions with the properties $f(L_1) \subseteq L_2, f(\bar{L}_1) \subseteq \bar{L}_2, g(L_2) \subseteq L_3, g(\bar{L}_2) \subseteq \bar{L}_3,$. (Such functions exist due to the definition of reduction.)

Then $g \circ f$ is computabe and a reduction of L_1 to L_3 since
$g \circ f(L_1) = g(f(L_1)) \subseteq g(L_2) \subseteq L_3$ and
$g \circ f(\bar{L}_1) = g(f(\bar{L}_1)) \subseteq g(\bar{L}_2) \subseteq \bar{L}_3$. □

Proof of the theorem: $L_u \leqslant L_{\text{MPCP}}$, $L_{\text{MPCP}} \leqslant L_{\text{PCP}}$ and we know from the previous lecture, that $L_u \notin R$. By transitivity of reduction a theorem from the previous lecture we have $L_{\text{PCP}} \notin R$ proving the theorem. □

Undecidable problems on context-free grammars

Remember, a **CF** (context-free or type 2) **grammar** is a grammar with all rules having a single non-terminal on the left hand side.

In a **leftmost derivation** of a CF grammar it is always the leftmost non-terminal of a sequential form is chosen for rewriting.

An **unambiguous grammar** G is a CF grammar for which every word of $L(G)$ has a unique leftmost derivation.

$L_{\mathrm{UAG}} := \{\langle G\rangle \mid G \text{ is an unambiguous grammar}\}$.

Theorem

$L_{\mathrm{UAG}} \notin R$

Proof: We show, that $L_{\mathrm{PCP}} \leqslant \overline{L_{\mathrm{UAG}}} = \{\langle G\rangle \mid G \text{ is ambiguous}\}$.
Let $D = \left\{\frac{u_1}{v_1}, \ldots, \frac{u_n}{v_n}\right\}$ be a set of dominoes over alphabet Σ.

Let $\Delta = \{a_1, \ldots, a_n\}$ be an alphabet satisfying $\Sigma \cap \Delta = \varnothing$.

$P_A := \{A \to u_1 A a_1, \ldots, A \to u_n A a_n,\ A \to \varepsilon\}$.
$P_B := \{B \to v_1 B a_1, \ldots, B \to v_n B a_n,\ B \to \varepsilon\}$.

Undecidable problems on context-free grammars

$G_A = \langle A, \{A\}, \Sigma \cup \Delta, P_A\rangle$. $G_B = \langle B, \{B\}, \Sigma \cup \Delta, P_B\rangle$.
$G_D = \langle S, \{S, A, B\}, \Sigma \cup \Delta, \{S \to A, S \to B\} \cup P_A \cup P_B\rangle$.

Claim: $f(\langle D\rangle) = \langle G_D\rangle$ is a reduction.

- if $\frac{u_{i_1}}{v_{i_1}} \cdots \frac{u_{i_m}}{v_{i_m}}$ is a solution of D, then $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$. But the $u_{i_1} \cdots u_{i_m} a_{i_m} \cdots a_{i_1} = v_{i_1} \cdots v_{i_m} a_{i_m} \cdots a_{i_1}$ can be derived 2 different ways, so G_D is ambiguous.
- if G_D is ambiguous, then a word can be derived 2 different ways by leftmost derivations. One of these should start by applying $S \to A$ and the other one should start with applying $S \to B$ as G_A and G_B are unambiguous grammars. As generated words are of the form xy, $x \in \Sigma^*, y \in \Delta^*$ both the x and the y parts should be the same giving a solution of D (the indices of y^R gives the sequence of dominoes).

Since f is computable the claim holds. As $L_{\mathrm{PCP}} \notin R$ we have $\overline{L_{\mathrm{UAG}}} \notin R$ implying $L_{\mathrm{UAG}} \notin R$ by a theorem from the previous lecture. □

Undecidable problems on context-free grammars

Lemma

For the grammars G_A and G_B defined in the previous theorem the languages $\overline{L(G_A)}$ and $\overline{L(G_B)}$ are context-free.

Note, that the statement in non-trivial as $\mathcal{L}_2$, the class of context-free languages is not closed for the complement operation.

Proof: We prove it for G_A, the proof is similar for G_B. Let $n_i := |u_i|$ $(1 \leqslant i \leqslant |D|)$. We construct a deterministic pushdown automaton accepting $L(G_A)$.

Idea: we scan the input from left to right and push letters from Σ into the stack. For letters $a_i \in \Delta$ we try to pop u_i^R from the stack. So let $A = \langle \Sigma \cup \{\#\}, Q, \Sigma \cup \Delta, \delta, q_0, \#, \{s\}\rangle$ be a pushdown automaton, where

$$Q = \{q_0, r, s\} \cup \bigcup_{i=1}^{|D|} \{q_{i1}, \ldots, q_{i(n_i-1)}\}$$

Undecidable problems on context-free grammars

and

$(\#t, q_0) :\in \delta(\#, q_0, t) \qquad (t \in \Sigma)$

$(t_1 t_2, q_0) :\in \delta(t_1, q_0, t_2) \qquad (t_1, t_2 \in \Sigma)$

$(\varepsilon, q_{i(n_i-1)}) :\in \delta(t_{n_i}, x, a_i)$
$(1 \leqslant i \leqslant |D|, x \in \{q_0, r\}, u_i = t_1 \cdots t_{n_i}, n_i \geqslant 2)$

$(\varepsilon, q_{i(j-1)}) :\in \delta(t_j, q_{ij}, \varepsilon)$
$(1 \leqslant i \leqslant |D|, 2 \leqslant j \leqslant n_i - 1, u_i = t_1 \cdots t_{n_i}, n_i \geqslant 2)$

$(\varepsilon, r) :\in \delta(t_1, q_{i1}, \varepsilon) \qquad (1 \leqslant i \leqslant |D|, u_i = t_1 \cdots t_{n_i}, n_i \geqslant 2)$

$(\varepsilon, r) :\in \delta(t, r, a_i) \qquad (1 \leqslant i \leqslant |D|, u_i = t, t \in \Sigma)$

$(\#, s) :\in \delta(\#, r, \varepsilon)$

A is deterministic (missing transitions can be added going into a trap state) and accepts $L(G_A)$.

Undecidable problems on context-free grammars

Claim: The class of languages that can be recognised by deterministic pushdown automaton is closed under the complement operation.

Proof of the claim: Let L be a language recognised by a deterministic pushdown automaton with the set of states Q and the set of accepting states F. Changing F to $Q \backslash F$ changes the recognized language to $\bar{L}$.

Proof of the lemma: We've seen, that there exists a deterministic pushdown automaton recognising $L(G_A)$. According to the claim there exist a (deterministic) pushdown automaton recognising $\overline{L(G_A)}$. We know (from FOTOC1), that any language recognised by a pushdown automaton is a CF language. So $\overline{L(G_A)}$ have this property, too. □

Undecidable problems on context-free grammars

Theorem

The following problems on CF grammars G_1 and G_2 are undecidable.

(1) $L(G_1) \cap L(G_2) \stackrel{?}{=} \varnothing$

(2) $L(G_1) \stackrel{?}{=} L(G_2)$

(3) $L(G_1) \stackrel{?}{=} \Gamma^*$ for some alphabet Γ.

(4) $L(G_1) \stackrel{?}{\subseteq} L(G_2)$

Proof:

(1) L_{PCP} can be reduced to this problem. Let $D = \left\{ \frac{u_1}{v_1}, \ldots, \frac{u_n}{v_n} \right\}$ be a set of dominoes and let G_A and G_B be the CF grammars defined in the proof of an earlier theorem. Easy to see, that D has a solution if and only if the intersection of $L(G_A)$ and $L(G_B)$ is non-empty.

Undecidable problems on context-free grammars

(2) L_{PCP} can be reduced to this problem. Let D, G_A and G_B be as in (1).

$L = \overline{L(G_A) \cap L(G_B)} = \overline{L(G_A)} \cup \overline{L(G_B)} \in \mathcal{L}_2$ holds by the previous lemma, $\overline{L(G_A)} \in \mathcal{L}_2$ and $\overline{L(G_B)} \in \mathcal{L}_2$ and the fact that $\mathcal{L}_2$ is closed for the union operation.

Let G_1 and G_2 be CF grammars satisfying $L(G_1) = L$ and $L(G_2) = (\Sigma \cup \Delta)^*$ (such grammars exist since both languages are in $\mathcal{L}_2$). $L(G_1) = L(G_2) \Leftrightarrow L(G_A) \cap L(G_B) = \varnothing$, so if (2) were decidable then so would L_{PCP} be according to a similar argument as in (1).

(3) Let G_1 be the same as in (2) and $\Gamma = \Sigma \cup \Delta$. A very similar argument as in (2) show, that decidability of this problem would imply the decidability of L_{PCP}.

(4) Since $L(G_1) = L(G_2) \Leftrightarrow L(G_1) \subseteq L(G_2) \wedge L(G_2) \subseteq L(G_1)$ decidability of (4) would imply decidability of (2). □

Fundamentals of theory of computation 2

7th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

Decidable problems in propositional logic

Proposition: The following algorithmic problems of propositional logic are decidable.

- Is a propositional formula φ satisfiable?
- Is a propositional formula φ unsatisfiable?
- Is a propositional formula φ valid?
- Is $\varphi \equiv \psi$ hold for propositional formulas φ, ψ?
- Is $\mathcal{F} \models \varphi$ hold for a finite set of propositional formulas $\mathcal{F}$ and a propositional formula φ?

Proof: Make all truth tables. The answers for all of the above questions can be read from the truth tables. □

Remark: Decidability of the above algorithmic problems is based on the fact that number of interpretations is finite. (2^n for n variables.) Each of these interpretations can be checked in finite steps even by the "brute force" method of making the turth tables.

There are some methods known being more efficient in practice. Worst case time complexity of these methods is still exponential.

Undecidable problems in first order logic

Next, we shall prove that the most basic algorithmic problems of first order logic, like whether a formula is satisfiable is undecidable, i.e., there is no TM halting for every input and giving the answer 'yes' for satisfiable formulas and 'no' for the unsatisfiable ones.

Definition

$\textsc{ValidityPred} := \{\langle\varphi\rangle \mid \varphi \text{ is a valid first order formula}\}$.
$\textsc{UnsatPred} := \{\langle\varphi\rangle \mid \varphi \text{ is an unsatisfiable first order formula}\}$.
$\textsc{SatPred} := \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable first order formula}\}$.
$\textsc{EqivPred} := \{\langle\varphi,\psi\rangle \mid \varphi,\psi \text{ are first order formulas and } \varphi \equiv \psi\}$.
$\textsc{ConsPred} := \{\langle\mathcal{F},\varphi\rangle \mid \mathcal{F} \text{ is a finite set of first order formulas, } \varphi \text{ is a first order formula}, \mathcal{F} \models \varphi\}$.

Here, $\langle\varphi\rangle$ is a code of φ over $\{0,1\}$.

Remark: Analogously to what we did with the codes of TM's we can associate all non-codes with $\bot$, the constant unsatisfiable formula, so we can assume that $\overline{\textsc{UnsatPred}} = \textsc{SatPred}$.

Undecidable problems in first order logic

Theorem

$\textsc{ValidityPred} \notin \mathrm{R}$

Proof: We give a reduction of L_{PCP} to $\textsc{ValidityPred}$. According to earlier theorems this implies the statement of the theorem. So, for an arbitrary set of dominoes D we should give a first order formula φ_D with the property that D has a solution if and only if $\models \varphi_D$.

So let $D = \left\{\frac{u_1}{v_1}, \ldots, \frac{u_k}{v_k}\right\}$ $(k \geqslant 1)$ be a set of dominoes over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$.

Let $p \in \mathcal{P}$ be a predicate symbol of arity 2, $f_{a_1}, \ldots, f_{a_n} \in \mathcal{F}$ be function symbols of arity 1 and $c \in \mathcal{A}$ be a constant.

Notation: $f_{b_1 \cdots b_m}(t) := f_{b_1}(f_{b_2}(\cdots(f_{b_m}(t))\cdots))$ where $b_1, \ldots, b_m \in \Sigma$ and t is a term.

Undecidable problems in first order logic

Let $\varphi_D := \varphi_1 \wedge \varphi_2 \to \varphi_3$, where
$\varphi_1 = p(f_{u_1}(c), f_{v_1}(c)) \wedge \cdots \wedge p(f_{u_k}(c), f_{v_k}(c))$,
$\varphi_2 = \forall x \forall y (p(x,y) \to p(f_{u_1}(x), f_{v_1}(y)) \wedge \cdots \wedge p(f_{u_k}(x), f_{v_k}(y)))$,
$\varphi_3 = \exists z p(z,z)$.
Observe, that φ_D is a closed formula.
First, assume that φ_D is valid. Let I be the following interpretation. Let the domain of I be Σ^* and f_{a_i} be interpreted as F_{a_i}, where $F_{a_i}(u) := a_i u \, (1 \leqslant i \leqslant k, u \in \Sigma^*)$. Furthermore let c be interpreted as ε and p be interpreted as binary relation R, where $(u,v) \in R$ holds for some $u, v \in \Sigma^*$ if and only if

$$\text{there exist } m \geqslant 1 \text{ and } 1 \leqslant i_1, \ldots i_m \leqslant k, \text{ such that } u_{i_1} \cdots u_{i_m} = u \text{ and } v_{i_1} \cdots v_{i_m} = v. \qquad (*)$$

Observe, that (*) holds if and only if there exists a sequence of dominoes such that the concatenation of the top words is u and the concatenation of the bottom words is v.

Undecidable problems in first order logic

By induction on the length of u it is easy to check that $\mathcal{D}_I(f_u(c)) = u$ holds for every $u \in \Sigma^*$. So $v_I(p(f_{u_i}(c), f_{v_i}(c)))$ is true if and only if $(u_i, v_i) \in R$. Since $(u_i, v_i) \in R$ holds for every $1 \leqslant i \leqslant k$ we have $I \vDash \varphi_1$.

Assume, that $(u,v) \in R$ holds for some $u, v \in \Sigma^*$. Then by (*) we have an $m \geqslant 1$ and $1 \leqslant i_1, \ldots i_m \leqslant k$ with the properties $u_{i_1} \cdots u_{i_m} = u$ and $v_{i_1} \cdots v_{i_m} = v$.

For every $1 \leqslant i \leqslant k$ (*) holds for $u_i u$ and $v_i v$ as well (shown by the sequence of indices $i, i_1, \ldots i_m$) implying $(u_i u, v_i v) \in R$.

Since $\mathcal{D}_I(f_{u_i}(u)) = u_i u$ and $\mathcal{D}_I(f_{v_i}(v)) = v_i v$ holds for any $u, v \in \Sigma^*$ we have that $(u,v) \in R$ implies $(\mathcal{D}(f_{u_1}(u)), \mathcal{D}(f_{v_1}(v))) \in R, \ldots, (\mathcal{D}(f_{u_k}(u)), \mathcal{D}(f_{v_k}(v))) \in R$. So we have $I \vDash \varphi_2$.

φ_D is valid, so we have $I \vDash \varphi_D$. We also know, that $I \vDash \varphi_1$ and $I \vDash \varphi_2$ hold. These three imply $I \vDash \varphi_3$. φ_3 is true in I if and only if D has a solution.

Undecidable problems in first order logic

To prove the other direction suppose, that D has a solution and let I be an arbitrary interpretation of the first order logic with the above given $\mathcal{P}, \mathcal{F}, \mathcal{A}$. We have to prove that $I \vDash \varphi_D$.

If $I \nvDash \varphi_1 \wedge \varphi_2$, then $I \vDash \varphi_D$ holds according to the definition of $\to$. Therefore $I \vDash \varphi_1$ and $I \vDash \varphi_2$ can be assumed.

Let $m \geqslant 1$ and $1 \leqslant i_1, \ldots i_m \leqslant k$ with the property $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$ (such integers exist, since we assumed, that D has a solution).

$I \vDash \varphi_1$ implies $I \vDash p(f_{u_{i_m}}(c), f_{v_{i_m}}(c))$.

Using $I \vDash \varphi_2$ the following statements can be easily proved by induction.

$$\begin{gathered} I \vDash p(f_{u_{i_m}}(c), f_{v_{i_m}}(c)) \\ I \vDash p(f_{u_{i_{m-1}} u_{i_m}}(c), f_{v_{i_{m-1}} v_{i_m}}(c)) \\ \vdots \\ I \vDash p(f_{u_{i_1} \cdots u_{i_{m-1}} u_{i_m}}(c), f_{v_{i_1} \cdots v_{i_{m-1}} v_{i_m}}(c)) \end{gathered}$$

Undecidable problems in first order logic

Since $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$ we have that for $w = \mathcal{D}_I(f_{u_{i_1} \cdots u_{i_m}}(c))$ from the domain of I we have $(w,w) \in R$, where relation R is the interpretation of p in I.

So $I \vDash \varphi_1$ and $I \vDash \varphi_2$ implies $I \vDash \varphi_3$ proving $I \vDash \varphi_D$.

The mapping $D \mapsto \varphi_D$ is computable, so we have $L_{\text{PCP}} \leqslant \textsc{ValidityPred}$. The statement of the theorem follows from a previous theorem on reduction and from $L_{\text{PCP}} \notin \mathsf{R}$. $\square$.

Corollary

$\textsc{UnsatPred}, \textsc{SatPred}, \textsc{EquivPred}, \textsc{ConsPred} \notin \mathsf{R}$

Proof: φ is unsatisfiable $\Leftrightarrow \vDash \neg\varphi$. According to the previous theorem we have $\textsc{UnsatPred} \notin \mathsf{R}$.

The complement of an undecidable language is undecidable, so $\textsc{SatPred} \notin \mathsf{R}$.

φ is unsatisfiable $\Longleftrightarrow \varphi \sim \bot$, so $\textsc{EquivPred} \notin \mathsf{R}$.

$\varnothing \vDash \varphi \Longleftrightarrow$ φ is valid, so $\textsc{ConsPred} \notin \mathsf{R}$. $\square$

Undecidable problems in first order logic

Remark: There exists a partial algorithm halting for a first order formula φ with the answer 'yes' if and only if φ is unsatisfiable. (If φ is satisfiable the algorithm either halts with 'no' or does not halt.)

There exist first order resulution procedures. These algorithms are of this kind. We don't go into the details of these algorithms.

So the following statement holds (without proof).

Theorem

$\textsc{UnsatPred} \in \mathrm{RE}$.

Corollary

$\textsc{SatPred} \notin \mathrm{RE}$

Proof of the Corollary: According to a previous theorem $L, \bar{L} \in \mathrm{RE} \Rightarrow L \in \mathrm{R}$. Since $\overline{\textsc{UnsatPred}} = \textsc{SatPred}$ and $\textsc{UnsatPred} \in \mathrm{RE} \setminus \mathrm{R}$ we have $\textsc{SatPred} \notin \mathrm{RE}$. □

$\mathcal{L}_0$ and RE

Theorem

(1) For any grammar G there exists a NTM M with $L(M) = L(G)$.

(2) For any NTM M there exists a grammar G with $L(G) = L(M)$.

Proof: (1) Let M have 3 tapes. The 1st tape contains the input of M, the 2nd tape contains the rules of G. M does not overwrite these during its computations. On tape 3 there is a sequential form α of G. (Initially the start symbol of G.)

M chooses nondeterministically a rule $p \to q$ of G and a position k of α ($1 \leqslant k \leqslant |\alpha|$). If the subword of length $|p|$ of α begining with the kth letter of α equals to p then M replaces p by q in α (i.e., xpy is replaced by xqy on tape 3, where $|x| = k - 1$.)

Then M checks whether the contents of the 1st and 3rd tape are the same. In this case M halts in q_a, otherwise it continues with a new iteration. Computations of M correspond to derivations in G, so we have $L(M) = L(G)$. □

$\mathcal{L}_0$ and RE

(2) Let TM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle$ be a NTM. Then $G = \langle (\Gamma \setminus \Sigma) \cup Q \cup \{S, A, \rhd, \lhd\}, \Sigma, P, S \rangle$, where P is as follows.

1. $S \to \rhd A q_a A \lhd$
2. $A \to aA \,|\, \varepsilon$ $\qquad (\forall a \in \Gamma)$
3. $bq' \to qa$, if $\delta(q, a) = (q', b, R)$
4. $q'b \to qa$, if $\delta(q, a) = (q', b, S)$
5. $q'cb \to cqa$, if $\delta(q, a) = (q', b, L)$ $\qquad (\forall c \in \Gamma)$
6. $\sqcup \lhd \to \lhd$, $\quad \lhd \to \varepsilon$, $\quad \rhd \sqcup \to \rhd$, $\quad \rhd q_0 \to \varepsilon$

Sequential forms of G correspond to configurations of M, but G works in the opposite direction. First (1-2.), G generates an accepting configuration nondeterministically. Then it tries to derive a starting configuration of M (3-5.) from which the extra symbols can be easily eliminated (6.) to generate a word of $L(M)$.

More formally, by induction on the length (n) of a sequence of transitions we can show that

$\mathcal{L}_0$ and RE

$upv \vdash^* u'qv'$ ($p, q \in Q, u, u', v, v' \in \Gamma^*$) if and only if $\rhd \sqcup^{i'} u'qv' \sqcup^{j'} \lhd \Rightarrow^* \rhd \sqcup^{i} upv \sqcup^{j} \lhd$ holds for some $i, i', j, j' \in \mathbb{N}$.

For $n = 0$ the statement obviously holds. For n to $n + 1$ let's check it for right-moves. If $upv \vdash^* u'qav' \vdash u'brv''$ holds for some $a, b \in \Gamma, r \in Q$, where $v'' = v'$ if $v' \neq \varepsilon$, $v'' = \sqcup$ otherwise, then by induction we have $\rhd \sqcup^{i'} u'qav' \sqcup^{j'} \lhd \Rightarrow^* \rhd \sqcup^{i} upv \sqcup^{j} \lhd$ for some $i, i', j, j' \in \mathbb{N}$ and since $bs \to qa \in P$ we have $\rhd \sqcup^{i'} u'brv'' \sqcup^{j'} \lhd \Rightarrow \rhd \sqcup^{i'} u'qav' \sqcup^{j'} \lhd$. So $\rhd \sqcup^{i'} u'brv'' \sqcup^{j'} \lhd \Rightarrow^* \rhd \sqcup^{i} upv \sqcup^{j} \lhd$ holds, i.e., we proved the statement for a sequence of transitions of length $n + 1$ ending with a right-move. The proof is similar for stay- and left-moves, and for the other direction.

So $q_0 w \vdash^* \alpha q_a \beta$ holds for some $\alpha, \beta \in \Gamma^*$ if and only if $\rhd \sqcup^{i'} \alpha q_i \beta \sqcup^{j'} \lhd \Rightarrow^* \rhd \sqcup^{i} q_0 w \sqcup^{j}$. Since $S \Rightarrow^* \rhd \sqcup^{i'} \alpha q_i \beta \sqcup^{j'} \lhd$ and $\rhd \sqcup^{i} q_0 w \sqcup^{j} \lhd \Rightarrow^* w$ we have $q_0 w \vdash_M^* \alpha q_i \beta \;\Leftrightarrow\; S \Rightarrow_G^* w$. □

Corollary: According to a previous theorem, a deterministic TM accepting $L(G)$ can be given in (1) as well.

Linear bounded automaton

Definition

A **linear bounded automaton** (LBA) is a **nondeterministic** TM restricted as follows. Its input alphabet Σ contains two special symbols $\rhd$ (left endmarker) and $\lhd$ (right endmarker). Furthermore

- inputs are of $\rhd(\Sigma\backslash\{\rhd,\lhd\})^*\lhd$,
- $\rhd$ and $\lhd$ can not be overwritten,
- head can not be to the left from $\rhd$ and to the right from $\lhd$,
- the initial position of the head is the right neighbor of the cell with the content $\rhd$.

So an LBA is an NTM, where the head is restricted to stay in the space occupied by the input during computation.

Remark: The name comes from an equivalent model, where the size of available space is constant times (a linear function of) the length of the input. It can be shown, that such a constant factor does not increase the computational power of the model.

$\mathcal{L}_1$ and LBA

Theorem

(1) For every type 1 grammar G there exists an LBA A having $L(A) = L(G)$.

(2) For every LBA A there exists a type 1 grammar G having $L(G) = L(A)$.

Proof (sketch):

(1) In the proof of (1) of the previous theorem we have constructed a NTM M recognising $L(G)$, where G is a type 0 grammar. On the 3rd tape of M a sequential form of G was generated and was compared with the input at the end of the iteration steps.

If grammar G is of type 1, then the rules of G are length-nondecreasing. So, the sequential form on tape 3 does not need more space than $|u|$ in any accepting computation.

Therefore, the NTM constructed in the previous proof can be easily converted to an LBA.

$\mathcal{L}_1$ and LBA

(2) We would be ready if the rules in the proof the previous theorem were length-nondecreasing.
But length-decreasing rules can be avoided.

- We used $A \to \varepsilon$ to generate an accepting configuration. Since we are building a word of length at least 1 from a single starting symbol (of length 1), such a rule is not needed, we used it for simplicity only.
- The rules to eliminate $\rhd$, $\lhd$, q_0 are used only once per derivation. We can add symbols from $(N \cup T) \times (N \cup T)$ to simulate these by length-nondecreasing rules. (Example: length-decreasing rule $AB \to C$ can be replaced by $(A, B) \to C$),
- If M is linear bounded then the configurations of M can not be longer than the length of the starting configuration. So no extra $\sqcup$'s are needed to be generated by the production rules 1-2. and no production rules are needed to eliminate any $\sqcup$'s. □

$\mathcal{L}_1$ and R

Theorem

Let A be an LBA, then $L(A)$ is decidable.

Proof: Since A is an LBA there is an upper bound $m(u) = |Q| \cdot |u| \cdot |\Gamma|^{|u|}$, for the number of different configurations for an input u (Q is the set of states, Γ is the tape alphabet of A).

If A has an accepting computation, then there should be one with at most $m(u)$ steps. (Steps between same configurations can be left out.)

Let M be an NTM working exactly as A, but stopping all computation in its rejecting state after $m(u)$ steps. Then $L(M) = L(A)$ and all computations of M halts for every input. □

Corollary

$\mathcal{L}_1 \subseteq R$.

$\mathcal{L}_1$ and R

Theorem

$\mathcal{L}_1 \subset \mathrm{R}$.

Proof: By the previous corollary we have $\mathcal{L}_1 \subseteq \mathrm{R}$.

Let $L_{\text{LBA-diag}} = \{\langle M \rangle \mid M \text{ LBA and } \langle M \rangle \notin L(M)\}$.

- $L_{\text{LBA-diag}}$ is decidable. ($\Rightarrow L_{\text{LBA-diag}} \in$ R)

  Let the TM S go for an input LBA M to its q_a if $\langle M \rangle \notin L(M)$ and to its q_r if $\langle M \rangle \in L(M)$. Since $L(M)$ is decidable S halts for every input.
- $L_{\text{LBA-diag}}$ is unrecognisable by an LBA ($\Rightarrow L_{\text{LBA-diag}} \notin \mathcal{L}_1$)

  (By the diagonal method of Cantor)

  Suppose that $L_{\text{LBA-diag}}$ can be recognised by an LBA S.
  - if $\langle S \rangle \in L_{\text{LBA-diag}} = L(S)$ holds, then S recognises $\langle S \rangle$ so we have $\langle S \rangle \notin L_{\text{LBA-diag}}$, a contradiction,
  - if $\langle S \rangle \notin L_{\text{LBA-diag}} = L(S)$, then S does not recognise $\langle S \rangle$, so we have $\langle S \rangle \in L_{\text{LBA-diag}}$, a contradiction. □

R,RE and the Chomsky language classes

In the following table models of computation in the same cell has the same computational power and has a bigger computational power than the models being in above cells.

$\mathcal{L}_3$	type 3 (regular) grammars deterministic finite automaton nondeterministic finite automaton regular expressions
	deterministic pushdown automaton
$\mathcal{L}_2$	type 2 (CF) grammars (nondeterministic) pushdown automaton
$\mathcal{L}_1$	type 1 (CS) grammars linear bounded automaton
R	Turing machine halting for every input
RE = $\mathcal{L}_0$	Turing machine nondeterministic Turing machine type 0 grammar

Fundamentals of theory of computation 2

8th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

COMPLEXITY THEORY

Next, we consider problems of R. Since these problems are decidable the question is how efficiently these problems can be solved regarding a certian resource, typically time or space.

These problems can be classified according to how much resource the most economic solution uses.

Complexity theory deals with these time and space complexity classes and their relations to each other.

Time complexity classes, $P \stackrel{?}{=} NP$

Definition

- $\mathrm{TIME}(f(n)) =$
  $\{L \mid L$ is decidable by a $O(f(n))$ time deterministic TM$\}$
- $\mathrm{NTIME}(f(n)) =$
  $\{L \mid L$ is decidable by a $O(f(n))$ time nondeterministic TM$\}$
- $P = \bigcup_{k \geq 1} \mathrm{TIME}(n^k)$.
- $NP = \bigcup_{k \geq 1} \mathrm{NTIME}(n^k)$.

Example: Our earlier theorem: $\mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$.

Observation: $P \subseteq NP$, as deterministic TM's can be considered as special cases of nondeterministic TM's.

Conjecture: $P \neq NP$ (it is conjectured to be true, but we cannot prove it).

The Clay Mathematics Institute awards 1M$ for solving any of its 7 Millenium Prize Problems, proposed in 2000. One of them: $P \stackrel{?}{=} NP$.

NP

P can be considered as the set of efficiently solvable problems. (Not 100% true.)

What kind of problems are there in NP?

For a problem $L \in NP$ there is a NTM M, such that for all inputs w M "conjectures" (i.e., nondeterministically generates) a "witness" for the inclusion of w in L. Such a witness should be generated in polynomial time and should provide a polynomial time checkable proof for the inclusion.

Note, that this argument can be made formal, but we skip it.

Next, we examine the connection between P and NP.

Polynomial time reduction

Definition

$f : \Sigma^* \to \Delta^*$ is **computable in polynomial time**, if there is a TM, which computes it in polynomial time.

Definition

$L_1 \subseteq \Sigma^*$ is **reducable in polynomial time** to $L_2 \subseteq \Delta^*$ if there is a function $f : \Sigma^* \to \Delta^*$ computable in polynomial time satifying $w \in L_1 \Leftrightarrow f(w) \in L_2$. Notation: $L_1 \leqslant_p L_2$.

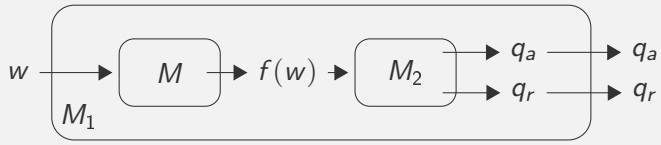
Another name for polynomial time reduction is Karp-reduction, named after Richard Karp.

Theorem

- If $L_1 \leqslant_p L_2$ és $L_2 \in$ P, then $L_1 \in$ P.
- If $L_1 \leqslant_p L_2$ és $L_2 \in$ NP, then $L_1 \in$ NP.

Polynomial time reduction

Proof:
Let $L_2 \in$ P and suppose, that $L_1 \leqslant_p L_2$.
Let M_2 be a TM deciding L_2 in polynomial time $p_2(n)$ and M be another TM computing the reduction in $p(n)$ time.
Construct M_1 as follows:



- M_1 decides L_1 (if $w \in L_1$, then $f(w) \in L_2$, so M_1 halts in q_a, otherwise $f(w) \notin L_2$, so M_1 it halts in q_r)
- if w is of length n, then $f(w)$ has a length of at most $p(n) + n$
- time complexity of M_1 is $p(n) + p_2(p(n) + n)$ which is a polynomial itself

$\mathcal{C}$-completness

Intuitively, if we reduce a language to another it means that it is at least as hard to decide the other language as the first one. In this sense the hardest problems with respect to a class of languages are those languages for which all languages can be reduced in the class.

Definition

Let $\mathcal{C}$ be a class of languages. A problem L is **$\mathcal{C}$-hard** (with respect to polynomial time reduction), if $L' \leqslant_p L$ holds for all $L' \in \mathcal{C}$.

Definition

A $\mathcal{C}$-hard problem L is **$\mathcal{C}$-complete**, if $L \in \mathcal{C}$.

Examples for $\mathcal{C}$ are P,NP,EXP (the class of languages decidable in exponential time), space complexity classes.

NP-completeness

So in the case of $\mathcal{C}$=NP

Definition

L is called **NP-complete** (with respect to polynomial time reduction), if

- $L \in$ NP
- L is NP-hard, i.e., $L' \leqslant_p L$ holds for all $L' \in$NP.

Theorem

Let L be an NP-complete language. If $L \in$ P, then P = NP.

Proof: Enough to show that NP $\subseteq$ P.
Let $L' \in$NP be an arbitrary problem.
Then $L' \leqslant_p L$ holds, since L is NP-complete.
As $L \in$P, $L' \in$P holds, too, by a previous theorem.
This holds for all $L' \in$NP, so NP $\subseteq$ P. □

NP-completeness

So NP-complete problems (if there are any) are the hardest problems of NP. No polynomial algorithm is known for them and expected to be invented in the future. This is due to the previous theorem, since a polynomial time algorithm for an NP-complete problem would mean P=NP.

Is there any NP-complete problems at all?

$\mathrm{SAT} = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable propositional formula in CNF}\}$

Theorem (Cook,Levin)

SAT is NP-complete.

Proof: $\mathrm{SAT} \in$ NP: Given a CNF φ. A computation of an NTM can produce an interpretation I in polynomial time. Then it can be checked in polynomial time whether $I \models \varphi$.

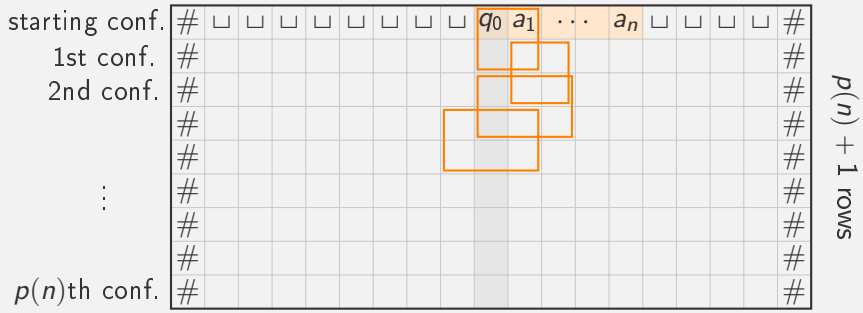
Proof of Cook-Levin Theorem

SAT is NP-hard: Let $L \in$ NP be arbitrary, we need to prove, that $L \leqslant_p \mathrm{SAT}$.

- Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r\rangle$ be a NTM deciding L in $p(n)$ polynomial time (Assume $p(n) \geqslant n$.)
- Let $w = a_1 \cdots a_n \in \Sigma^*$ be a word.
- We construct a CNF φ_w in polynomial time satisfying $w \in L \Leftrightarrow \langle\varphi_w\rangle \in \mathrm{SAT}$.
- A computation of M on w can be described by a table T where
  - the 1st row is $\# \sqcup^{p(n)} C_0 \sqcup^{p(n)-n} \#$, where $C_0 = q_0 w$ is the starting configuration of M for w
  - Consecutive rows of T are consecutive configurations of M (extended by enough $\sqcup$'s and #'s at the begining and at the end). The rows are of length $2p(n) + 3$.

Proof of Cook-Levin Theorem

– there are $p(n) + 1$ rows. An accepting configuration is repeated till the last row if it is reached earlier.



– by the definition of the one step transition relation difference between two consecutive rows can be covered by a 2x3 "window "
– T is wide enough to contain all computations of length $\leqslant p(n)$. The number of $\sqcup$'s (i.e., the width of T) is big enough to ensure these 2x3 "windows" do not "fall off" at the sides.

Proof of Cook-Levin Theorem

– Atoms of φ_w are of the form $p_{i,j,s}$, where $1 \leqslant i \leqslant p(n) + 1$, $1 \leqslant j \leqslant 2p(n) + 3$ and $s \in \Delta$ for $\Delta = Q \cup \Gamma \cup \{\#\}$.

– φ_w describes all possible computations of M for w of length $\leqslant p(n)$. φ_w is of the form $\varphi_w = \varphi_0 \wedge \varphi_{\text{start}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}}$.

– φ_0 is true if and only if all cells contain exactly one symbol

$$\varphi_0 := \bigwedge_{\substack{1\leqslant i\leqslant p(n)+1\\ 1\leqslant j\leqslant 2p(n)+3}} \left(\left(\bigvee_{s\in\Delta} p_{i,j,s} \right) \wedge \bigwedge_{s,t\in\Delta, s\neq t} (\neg p_{i,j,s} \vee \neg p_{i,j,t}) \right)$$

– So interpretations satisfying φ_0 correspond to filling T and $p_{i,j,s}$ is true if and only if $T(i,j) = s$.

– φ_{start} is true if and only if the first row of T is the starting configuration extended by enough $\sqcup$'s and #'s.

$$\varphi_{\text{start}} := p_{1,1,\#} \wedge p_{1,2,\sqcup} \wedge \cdots \wedge p_{1,2p(n)+2,\sqcup} \wedge p_{1,2p(n)+3,\#}$$

Proof of Cook-Levin Theorem

– φ_{move} is true if all "windows" are legal according to δ:

$$\varphi_{\text{move}} := \bigwedge_{\substack{1 \leqslant i \leqslant p(n) \\ 2 \leqslant j \leqslant 2p(n)+2}} \psi_{i,j},$$

where $\psi_{i,j} \sim \bigvee_{\substack{(b_1,\ldots,b_6) \\ \text{legal window}}} p_{i,j-1,b_1} \wedge p_{i,j,b_2} \wedge p_{i,j+1,b_3} \wedge p_{i+1,j-1,b_4} \wedge p_{i+1,j,b_5} \wedge p_{i+1,j+1,b_6}$

b_1	b_2	b_3
b_4	b_5	b_6

Trouble: $\psi_{i,j}$ is not a disjunction of literals. Let's change it to

$\psi_{i,j} := \bigwedge_{\substack{(b_1,\ldots,b_6) \\ \text{illegal window}}} \neg p_{i,j-1,b_1} \vee \neg p_{i,j,b_2} \vee \neg p_{i,j+1,b_3} \vee \neg p_{i+1,j-1,b_4} \vee \neg p_{i+1,j,b_5} \vee \neg p_{i+1,j+1,b_6}$

Proof of Cook-Levin Theorem

– finally: φ_{accept} is true if and only if there is a q_a in the last row

$$\varphi_{\text{accept}} = \bigvee_{j=2}^{2p(n)+2} p_{p(n)+1,j,q_a}$$

.

– $w \in L \Leftrightarrow$ NTM M has a computation accepting $w \Leftrightarrow T$ can be filled so that φ_w is true $\Leftrightarrow \varphi_w$ is satisfiable $\Leftrightarrow \langle \varphi_w \rangle \in$ SAT,

– how many literals are there in φ_w? Let $k := |\Delta|$.

φ_0 : $(p(n)+1)(2p(n)+3)(k+k(k-1)) = O(p^2(n))$,

φ_{start} : $2p(n)+3 = O(p(n))$,

φ_{move} : $\leqslant p(n)(2p(n)+1)k^6 \cdot 6 = O(p^2(n))$,

φ_{accept} : $2p(n)+1 = O(p(n))$,

so φ_w has size $O(p^2(n))$, that can be constructed in polynomial time

– so $w \mapsto \langle \varphi_w \rangle$ is a pol. time reduction, i.e., $L \leqslant_p$ SAT.

– This holds for all $L \in$NP. So SAT is NP-hard. Being in NP means it is NP-complete.

Transitivity of polynomial time reduction

Theorem

$L_1 \leqslant_p L_2,\ L_2 \leqslant_p L_3 \Rightarrow L_1 \leqslant_p L_3$.

Proof:

Let the first reduction be realized by function f and let M_1 be a TM computing f in $p_1(n)$ time. Let the second reduction be realized by function g and let M_2 be a TM computing g in $p_2(n)$ time. We can assume, that $p_1(n)$ and $p_2(n)$ are polynomials.

$w \in L_1 \Leftrightarrow f(w) \in L_2 \Leftrightarrow g(f(w)) \in L_3$, so $g \circ f$ is a reduction of L_1 to L_3.

$|f(w)| \leqslant n + p_1(n)$, if $|w| = n$, since M_1 takes at most $p_1(n)$ steps increasing the length by at most 1 per step. So $M_2 \circ M_1$ computes the reduction $g \circ f$ in at most $h(n) := p_1(n) + p_2(n + p_1(n))$ time.

So $g \circ f$ reduces L_1 to L_3 in $h(n)$ time, which is a polynomial. Therefore $L_1 \leqslant_p L_3$ holds. □

Further NP-complete problems

So polynomial time reduction can be used to prove NP-completeness of other languages.

Theorem

If L is NP-complete, $L \leqslant_p L'$ and $L' \in$ NP, then L' is NP-complete as well.

Proof: Let $L'' \in$ NP be arbitrary. Since L is NP-complete $L'' \leqslant_p L$ holds. By the conditions $L \leqslant_p L'$ holds, therefore by the transitivity of polynomial time reductions L' is NP-hard. The statement follows from this fact and the 3rd condition of the theorem. □

kSAT

Definition

Let $k \geqslant 1$. A CNF, where each term consists of exactly k literals of pairwise different atoms is called a kCNF.

Examples:

4CNF:
$(\neg x_1 \vee x_3 \vee x_5 \vee \neg x_6) \wedge (\neg x_1 \vee \neg x_3 \vee x_4 \vee \neg x_6) \wedge (x_1 \vee x_2 \vee \neg x_4 \vee \neg x_6)$.

2CNF: $(\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$.

Definition

kSAT=$\{\langle\varphi\rangle \mid \varphi$ is a satisfiable kCNF formula in propositional logic$\}$

NP-completeness of 3SAT

Theorem

3SAT is NP-complete.

Proof:

- 3SAT is in NP, see SAT
- SAT$\leqslant_p$ 3SAT
  We need $f : \varphi \mapsto \varphi'$, φ CNF, φ' 3CNF, φ' is satisfiable $\Leftrightarrow$ φ satisfiable, f is a polynomial time reduction.

$\varphi \mapsto \varphi'$:

ℓ	$\ell \vee x \vee y,\ \ell \vee x \vee \neg y,\ \ell \vee \neg x \vee y,\ \ell \vee \neg x \vee \neg y$
$\ell_1 \vee \ell_2$	$\ell_1 \vee \ell_2 \vee x,\ \ \ell_1 \vee \ell_2 \vee \neg x$
$\ell_1 \vee \ell_2 \vee \ell_3$	$\ell_1 \vee \ell_2 \vee \ell_3$
$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$	$\ell_1 \vee \ell_2 \vee x,\ \ \neg x \vee \ell_3 \vee \ell_4$
$\ell_1 \vee \cdots \vee \ell_n$ $(n \geqslant 5)$	$\ell_1 \vee \ell_2 \vee x_1, \neg x_1 \vee \ell_3 \vee x_2, \ldots, \neg x_{n-3} \vee \ell_{n-1} \vee \ell_n$

$x, x, x_1, \ldots, x_{n-2}$ are new atoms.

φ' is the conjunction of these terms.

NP-completeness of 3SAT

ℓ	$\ell \vee x \vee y,\ \ell \vee x \vee \neg y,\ \ell \vee \neg x \vee y,\ \ell \vee \neg x \vee \neg y$
$\ell_1 \vee \ell_2$	$\ell_1 \vee \ell_2 \vee x,\ \ \ell_1 \vee \ell_2 \vee \neg x$
$\ell_1 \vee \ell_2 \vee \ell_3$	$\ell_1 \vee \ell_2 \vee \ell_3$
$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$	$\ell_1 \vee \ell_2 \vee x,\ \ \neg x \vee \ell_3 \vee \ell_4$
$\ell_1 \vee \cdots \vee \ell_n$ $(n \geqslant 5)$	$\ell_1 \vee \ell_2 \vee x_1, \neg x_1 \vee \ell_3 \vee x_2, \ldots, \neg x_{n-3} \vee \ell_{n-1} \vee \ell_n$

We prove that for any interpretation I satisfying φ I can be extended to an interpretation I' for the new atoms in such a way that I' satisfies φ'.

On the other hand, given an interpretation I' for φ', there is a restriction I of I' for the original atoms satisfying φ.

We prove it for the different lengths of the terms. Consider a term of φ of n literals.

$n = 3$: nothing to prove

$n = 2$: ($\Rightarrow$): if one of the literals is true, then this makes both new terms true ($\Leftarrow$): one of x and $\neg x$ is false, so if both new ters are true, then either ℓ_1 or ℓ_2 should be true.

NP-completeness of 3SAT

ℓ	$\ell \vee x \vee y,\ \ell \vee x \vee \neg y,\ \ell \vee \neg x \vee y,\ \ell \vee \neg x \vee \neg y$
$\ell_1 \vee \ell_2$	$\ell_1 \vee \ell_2 \vee x,\ \ \ell_1 \vee \ell_2 \vee \neg x$
$\ell_1 \vee \ell_2 \vee \ell_3$	$\ell_1 \vee \ell_2 \vee \ell_3$
$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$	$\ell_1 \vee \ell_2 \vee x,\ \ \neg x \vee \ell_3 \vee \ell_4$
$\ell_1 \vee \cdots \vee \ell_n$ $(n \geqslant 5)$	$\ell_1 \vee \ell_2 \vee x_1, \neg x_1 \vee \ell_3 \vee x_2, \ldots, \neg x_{n-3} \vee \ell_{n-1} \vee \ell_n$

$n = 1$: ($\Rightarrow$): if ℓ is true, then all the new terms are true ($\Leftarrow$): Removing "$\ell \vee$" from the 4 terms at least one of the remainders is false in any interpretations, therefore ℓ is true in any satisfying interpretation od φ'.

$n = 4$: ($\Rightarrow$): if one of the literals ℓ_i $(i = 1, 2, 3, 4)$ is true, then it makes one of the new terms true. Truth value of x can be set to make the other term true. ($\Leftarrow$): one of x and $\neg x$ is false, so if both new terms are true, then one of ℓ_i $(i = 1, 2, 3, 4)$ should be true.

$n \geqslant 5$: ($\Rightarrow$): Suppose, that ℓ_i is true. Then let $x_1, \ldots, x_{i-2}$ be true $x_{i-1}, \ldots, x_{n-3}$ be false. It can be checked that this interpretation makes all new terms true.

NP-completeness of 3SAT

ℓ	$\ell \vee x \vee y,\ \ell \vee x \vee \neg y,\ \ell \vee \neg x \vee y,\ \ell \vee \neg x \vee \neg y$
$\ell_1 \vee \ell_2$	$\ell_1 \vee \ell_2 \vee x,\ \ell_1 \vee \ell_2 \vee \neg x$
$\ell_1 \vee \ell_2 \vee \ell_3$	$\ell_1 \vee \ell_2 \vee \ell_3$
$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$	$\ell_1 \vee \ell_2 \vee x,\ \neg x \vee \ell_3 \vee \ell_4$
$\ell_1 \vee \cdots \vee \ell_n$ $(n \geqslant 5)$	$\ell_1 \vee \ell_2 \vee x_1, \neg x_1 \vee \ell_3 \vee x_2, \ldots, \neg x_{n-3} \vee \ell_{n-1} \vee \ell_n$

$n \geqslant 5$: ($\Leftarrow$): Suppose on the contrary, that all new terms are true but $\ell_1, \ldots, \ell_n$ are false. This implies x_1 to be true, which implies x_2 to be true, etc. x_{n-3} true. But then the last new term is false, a contradiction.

So φ is satisfiable $\Leftrightarrow$ φ' is satisfiable. φ' can be computed from φ in polynomial time, therefore SAT$\leqslant_p$ 3SAT. $\square$

2SAT is in P

Theorem

2SAT $\in$ P.

Proof: Let φ be a 2CNF of atoms $x_1, \ldots, x_n$ and m terms.

We construct a directed graph G_φ of $2n$ vertices as follows. Let the vertices of G_φ be the $2n$ literals from $x_1, \ldots, x_n$ and add for each term $\ell_i \vee \ell_j$ the edges $(\neg\ell_i, \ell_j)$ and $(\neg\ell_j, \ell_i)$ to the edge set of G_φ. Motivation: $\ell_i \vee \ell_j \equiv \neg\ell_i \to \ell_j \equiv \neg\ell_j \to \ell_i$.

We shall prove the following statement.

Claim: φ is satisfiable if and only if none of the strongly connected components of G_φ contains a complementary pair of literals.

(Remember, a directed graph is called strongly connected, if there is a directed path between each pair of vertices in both direction. The vertices of a directed graph can be partitioned into strongly connected components.)

2SAT is in P

The claim implies the statement of the theorem since strongly connected components of a graph $G = (V, E)$ can be determined in $O(|V| + |E|)$ time (see e.g., Algorithms and Data Structures II). Here, $|V| = 2n, |E| = 2m$, so satisfiability of φ can be determined in $O(\max\{n, m\})$ time.

Proof of the claim: Consider an interpretation I of φ. Observe, that any true literal ℓ, i.e., a vertex in G_φ implies that all the out-neighbors of ℓ are true. Moreover it implies that all literals in the same strongly connected component are true.

Therefore all literals in the same strongly connected component of G_φ should have the same truth value. So it can not contain a complementary pair of literals as they have different truth values.

2SAT is in P

Suppose, that none of the strongly connected components of G_φ contain a complementary pair of literals. We have to give an interpretation satisfying φ.

Let x_i be an atom. Then there is no directed path either form x_i to $\neg x_i$ or from $\neg x_i$ to x_i. If none of these paths exist add $e = (x_i, \neg x_i)$ to the set of edges of G_φ.

This does not change the partition, since if x_j and $\neg x_j$ were be in unified component after the change, then this component would contain e and consequently x_i and $\neg x_i$ as well. But this is impossible, since there is no path from $\neg x_i$ to x_i avoiding e.

Continue adding edges in this manner till there is a path in G_φ between complementary pairs of literals in exactly one of the directions.

2SAT is in P

Let

$$I(x_i) := \begin{cases} T, & \text{if there is a path from } \neg x_i \text{ to } x_i \text{ in } G_\varphi \\ F & \text{if there is a path from } x_i \text{ to } \neg x_i \text{ in } G_\varphi \end{cases}$$

for every $1 \leqslant i \leqslant n$. So there is a path from every false literal to its complementary pair.

I makes all terms true.

Suppose on the contrary, that term $\ell_i \vee \ell_j$ of φ is false. Then both ℓ_i and ℓ_j are false. According to the remark after the definition of I

(1) there is a directed path from ℓ_i to $\neg\ell_i$ and from ℓ_j to $\neg\ell_j$.

On the other hand by the defintion of G_φ we have

(2) $(\neg\ell_i, \ell_j)$ and $(\neg\ell_j, \ell_i)$ are edges of G_φ.

(1) and (2) implies ℓ_i and $\neg\ell_i$ being in the same strongly connected component of G_φ, a contradiction proving the claim and the theorem. □

HORNSAT is in P

Definition

A CNF is called a **Horn formula** if each term contains at most one positive (unnegated) literal.

Example:

$(\neg x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4 \vee \neg x_6) \wedge (\neg x_2 \vee \neg x_4 \vee \neg x_6)$

Definition

HORNSAT=$\{\langle\varphi\rangle \mid \varphi$ is a satisfiable Horn formula$\}$

HORNSAT is in P

Theorem

HORNSAT $\in$ P.

Proof: Let φ be a Horn formula of atoms $x_1, \ldots, x_n$ and m terms. We can assume that all terms contain literals of pairwise different atoms. There can be 3 types of terms

1. $x_k \quad (1 \leqslant k \leqslant n)$,
2. $x_k \vee \neg x_{i_1} \vee \cdots \vee \neg x_{i_j} \quad (j \geqslant 1, 1 \leqslant k, i_1 \ldots, i_j \leqslant n)$,
3. $\neg x_{i_1} \vee \cdots \vee \neg x_{i_j} \quad (j \geqslant 1, 1 \leqslant i_1 \ldots, i_j \leqslant n)$.

We defne an interpretation $I_{\min}$ by the following algorithm.

- Initialize the truth value of each atom for false.
- Switch the truth vale of atoms in terms of type 1 to true.
- While there is a term $x_k \vee \neg x_{i_1} \vee \cdots \vee \neg x_{i_j}$ of type 2 with the property $I_{\min}(x_k) =$F and $I_{\min}(x_\ell) =$T for all $\ell \in \{i_1, \ldots, i_j\}$ do: Switch the truth value of x_k from false to true.

HORNSAT is in P

Claim: φ is satisfiable $\Leftrightarrow I_{\min} \models \varphi$.

It can be seen by induction, that after each switch in the above algorithm the current set of true atoms should be true in every satisfying interpretation.

Therefore every satisfying interpretation of φ should evaluate the true atoms of $I_{\min}$ for true.

So $I_{\min}$ is an interpretation evaluating the minimum number of type 3 terms for false.

If N is the length of the formula, then $I_{\min}$ can be computed in $O(mN) = O(N^2)$ time. After computing $I_{\min}$ it can be checked in $O(N)$ time whether all terms (of type 3) are satisfied in $I_{\min}$ or not. □

Fundamentals of theory of computation 2

9th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

k-colorable graphs

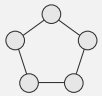
Definition

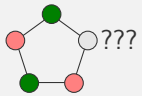
Let $k \geqslant 1$. An undirected graph is called k-colorable, if its nodes can be colored by k colors, so that there are no two adjacent nodes with the same color.

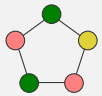
Formally: $G = (V, E)$ is k-colorable, if there exists a mapping $f : V \to \{1, \ldots, k\}$ with the property $\forall x, y \in V : f(x) = f(y) \Rightarrow \{x, y\} \notin E$.

Example 1: Let us denote the complete graph on n vertices by K_n. Then K_n is k-colorable for every $k \geqslant n$. On the other hand it is not $(n-1)$-colorable. (no two nodes can have the same color).

Example 2: A 5-cycle is 3-colorable, but not 2-colorable.







3-Coloring is NP-complete

Definition

k-Coloring:=$\{\langle G \rangle \mid G$ is k-colorable$\}$

$\langle G \rangle$ is the code of G over $\{0, 1\}$, e.g., the concatenation of the rows of its adjacency matrix.

Theorem

3-Coloring is NP-complete.

- k-Coloring is in NP: let different computations of a NTM correspond to different not neccessarily sound k-colorings of the vertices. Such a coloring can be produced in linear time. Soundness of a given coloring can be checked in polynomial time. Let sound colorings correspond to accepting computations. For a graph being k-colorable it needs only one sound k-coloring, to accept a word by a NTM it needs only one accepting computation.

3-Coloring is NP-complete

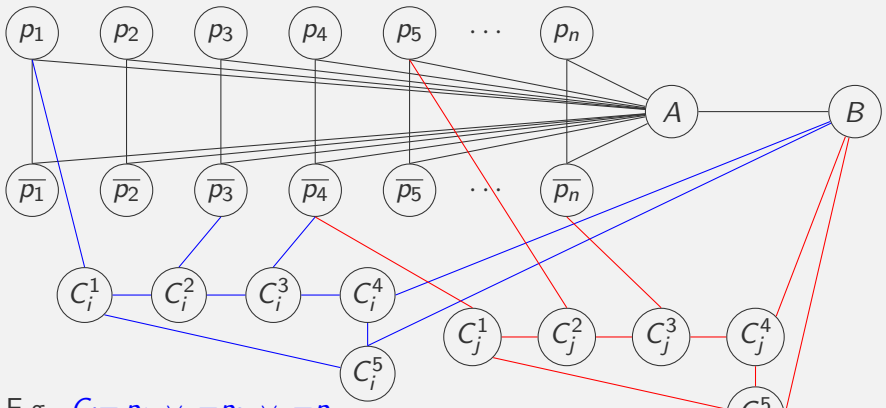
- 3SAT$\leqslant_p$ 3-Coloring

  Enough to construct a graph G_φ for every 3CNF formula φ in polynomial time with the property φ is satisfiable $\Leftrightarrow$ G_φ is 3 colorable.

  Let $p_1, \ldots, p_n$ be the atoms appearing in φ. Let $\varphi = C_1 \wedge \ldots \wedge C_m$, where the terms $C_1, \ldots C_m$ of φ are disjunctions of exacly 3 literals of different atoms.

  Construction of G_φ:

3-Coloring is NP-complete



E.g., $C_i = p_1 \vee \neg p_3 \vee \neg p_4$

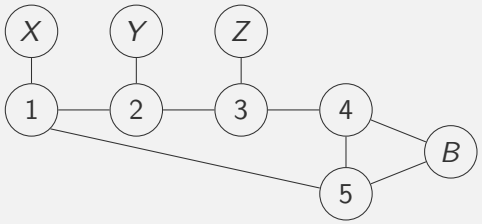
E.g., $C_j = \neg p_4 \vee p_5 \vee \neg p_n$

There is a pentagon corresponding to each term of φ as shown on the figure.

3-Coloring is NP-complete

Lemma: Let G_0 be a graph, as below, where X, Y, Z, B are colored with 2 colors. There exists an extension of this partial coloring for G_0 if and only if X, Y, Z, B are not monochromatic.

G_0



Proof of the lemma:

- If X, Y, Z, B are monochromatic, then there is no extension, since a pentagon can not be colored by 2 colors.
- Otherwise consider the following coloring.

  1st phase: we use only 2 colors, we color 1,2,3,4,5 to the opposite color of its neighbor from $\{X, Y, Z, B\}$.

3-Coloring is NP-complete

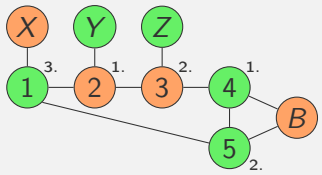
This coloring is not sound yet, as there can be neighbors among 1,2,3,4,5 with the same color.

2nd phase: Consider 1,2,3,4,5 in the clockwise cyclical order. If there are some consecutive nodes with the same color, then recolor every second of them for the 3rd color.

Example:

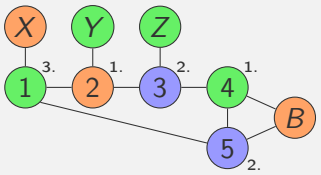
Coloring after phase 1:

G_0



Coloring after phase 2:

G_0



3-Coloring is NP-complete

- Suppose that φ is satisfiable and let I be an interpratation satisfying φ. Let's color G_φ with colors red, green and blue. If p_i is true in I, then let the node p_i be green and $\overline{p_i}$ be red. If p_i is false we color these vertices the opposite way. Let A be blue and B be red. Since all clauses are true all the pentagons have both a green (a true literal) and a red neighbor (B), so by the Lemma this coloring can be extended for all pentagons.
- Suppose that G_φ is colored by 3 colors. Wlog. let A be blue. $p_1, \ldots, p_n, \overline{p_1}, \ldots, \overline{p_n}$ are all neighbors of A so none of them can be blue. Furthermore the pairs $(p_i, \overline{p_i})$ are connected, so each pair have 1 green and 1 red vertex. Wlog. B is red (the green case is similar). Since all pentagons are colored there should be a green neighbor for all of them by the lemma. Interpretation "p_i:=true $\Leftrightarrow$ node p_i is "green" satisfies φ.

2-Coloring is in P

So $\varphi \mapsto G_\varphi$ is a reduction. As G_φ can be built from φ in polynomial time, this is a polynomial time reduction.

Since 3SAT is NP-complete our earlier theorem implies NP-completeness of 3-Coloring. □

The 2-colorable graphs are the bipartite graphs. (The 2 parts of the set of vertices correspond to the color classes.) Bipartiteness of a graph can be checked in linear time as follows.

Theorem

2-Coloring $\in$ P

Proof:
Start a breadth first search from any vertex x of $G(V, E)$. This will classify the vertices in the connected component of x according to the distance (length of shortest path) from x. If G is not connected do it for every component. Time complexity of this algorithm is $O(|V| + |E|)$.

2-Coloring is in P

Claim: G is 2-colorable $\Leftrightarrow$ a breadth first search of G does not find any edge between vertices on the same level.

Proof of the claim:

($\Leftarrow$) Let the vertices on even levels be blue, and vertices on odd levels be red. According to the condition the only possibility of an edge between 2 vertices of the same color is that they are in the same component and their level is at distance $\geqslant 2$. But BFS has no edge between levels of distance $\geqslant 2$.

($\Rightarrow$) If there is an edge between x and y of the same level then let z be their first common ancestor being k level higher in the BFS tree. Then there is a cycle of length $2k + 1$ containing x, y, z as due to z being the first common ancestor z is the only common vertex of the paths $z \rightsquigarrow x$ and $z \rightsquigarrow y$. Cycles of odd length are not 2-colorable, so G is not 2-colorable.

The property in the claim can be checked during the BFS algorithm. □

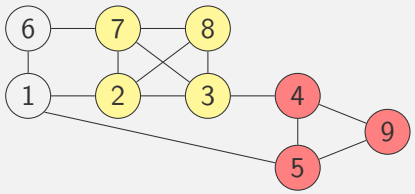
Clique

Definition

A complete subgraph of a simple, undirected graph G is called a **clique**.

Clique:$= \{\langle G, k\rangle \mid G \text{ has a clique of size } k\}$

Example:



$\{2, 3, 7, 8\}$ and $\{4, 5, 9\}$ are cliques. $\{1, 2, 6, 7\}$ is not a clique.

Observation: If G has a clique of size k, then it has a clique of any smaller size.

Independent set

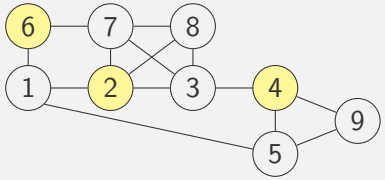
Definition

Let G be a simple, undirected graph. An empty (induced) subgraph of G is called an **independent set**.

Remark: Other names for independent set: *stable set*, *anticlique*.

Independent set:$=\{\langle G, k\rangle \mid G \text{ has an independent set of size } k\}$

Example:



$\{2, 6, 4\}$ is an independent set. $\{1, 7, 3, 9\}$ is not an independent set due to the edge $\{3, 7\}$.

Observation: If G has an independent set of size k then it has an independent set of any smaller k.

Vertex cover

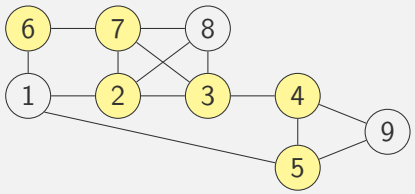
Definition

Let $S \subseteq V(G)$ and $e \in E(G)$. S **covers** the edge e if $S \cap e \neq \emptyset$. S is called a **vertex cover** if S covers all $e \in E(G)$.

Note: A vertex cover is sometimes called a *blocking set*.

VERTEX COVER$:= \{\langle G, k\rangle \mid G$ has a vertex cover of size $k\}$

Example:



$\{2, 3, 4, 5, 6, 7\}$ is a vertex cover.

Observation: If G has a vertex cover of size k, then it has a vertex cover for any $k \leqslant k' \leqslant |V(G)|$.

Further NP-complete graph problems

Theorem

INDEPENDENT SET, CLIQUE, VERTEX COVER are NP-complete.

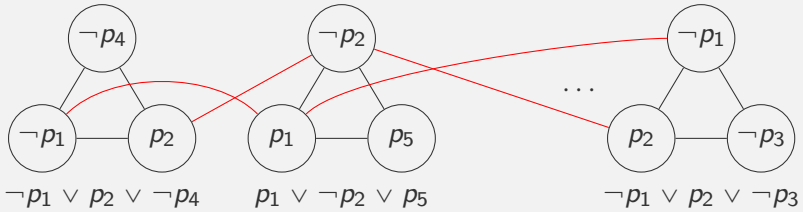
- A NTM can check a k-element subset on one of its branches. For all of these 3 languages both constructing k-element subset and checking whether the set is a clique/independent set/vertex cover can be done in polynomial time.
- 3SAT$\leqslant_p$ INDEPENDENT SET

  We need a function $f : \varphi \mapsto (G_\varphi, k)$ computable in polynomial time, where φ is a 3CNF and G_φ has an independent set of size k if and only if φ is satisfiable.

  Construction of (G_φ, k): If φ has m terms then let G_φ consist of m disjoint triangles, one for each term. Map the 3 literals of a term injectively to the 3 nodes of the term's triangle as labels. Add further edges between those pairs of nodes that have labels which are complementary pairs of literals. $k := m$.

NP-completeness of INDEPENDENT SET

An example:



* If φ is satisfiable, then all terms have at least one true literal in a satisfying interpretation I, choose one for each term, the corresponding vertices form an independent set of size m.

* If G_φ has m independent vertices, than it must contain exactly one per triangle. Consider one such set, there are no compementary pairs among these literals (they are connected), so this set corresponds to a partial interpretation which evaluates all the clauses for true. Complete this interpretation arbitrarily.

NP-completeness of CLIQUE and VERTEX COVER

- INDEPENDENT SET $\leqslant_p$ CLIQUE

  $f : (G, k) \mapsto (\bar{G}, k)$

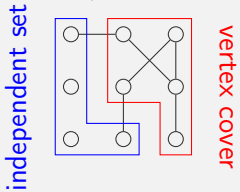
  A clique in G is an independent set in $\bar{G}$ and vice versa.

- INDEPENDENT SET $\leqslant_p$ VERTEX COVER

  $f : (G, k) \mapsto (G, |V(G)| - k)$

  If G has an independent set F of size k then there is vertex cover of size $|V(G)| - k$ (the complement of F).

  If G has a vertex cover L of size $|V(G)| - k$ then there is an independent set of size k (the complement of L).



Both reductions are of polynomial time.

Vertex cover in hypergraphs

Definition

$\mathcal{S}$ is a **hypergraph** (or a family of sets), if $\mathcal{S} = \{A_1, \ldots, A_n\}$, where $A_i \subseteq U$, $(1 \leqslant i \leqslant n)$ holds for some set U. $H \subseteq U$ is called a **vertex cover of hypergraph** $\mathcal{S}$, if $\forall 1 \leqslant i \leqslant n : H \cap A_i \neq \varnothing$ holds.

HYPERGRAPH VERTEX COVER:=
$\{\langle \mathcal{S}, k\rangle \mid \mathcal{S}$ is a hypergraph having a vertex cover of size $k\}$.

Theorem

HYPERGRAPH VERTEX COVER is NP complete.

Proof: It is in NP, since a subset H of U can be produced in polynomial time and it can be checked in polynomial time whether H is a vertex cover.

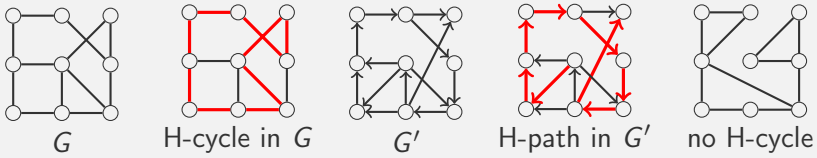
VERTEX COVER $\leqslant_p$ HYPERGRAPH VERTEX COVER. Every graph $G = (V_G, E_G)$ can be considered as a hypergraph with $U = V_G$, $\mathcal{S} = E_G$ giving a polynomial time reduction with k being the same. □

(Un)directed Hamiltonian path/cycle

Definition

Given a graph G. A **Hamiltonian path** is a path visiting all vertices of G. A cycle visiting all the vertices is called a **Hamiltonian cycle**. If G is directed then the Hamiltonian path/cycle should follow the direction.

Abbreviation: H-path, H-cycle for Hamiltonian path/cycle.



HP=$\{\langle G, s, t\rangle \mid G$ is a directed graph having an H-path form s to $t\}$.

UHP=$\{\langle G, s, t\rangle \mid G$ is an undirected graph having an H-path between s and $t\}$.

UHC=$\{\langle G\rangle \mid G$ is an undirected graph having an H-cycle$\}$

NP-completeness of directed $s \rightsquigarrow t$ H-path

Theorem

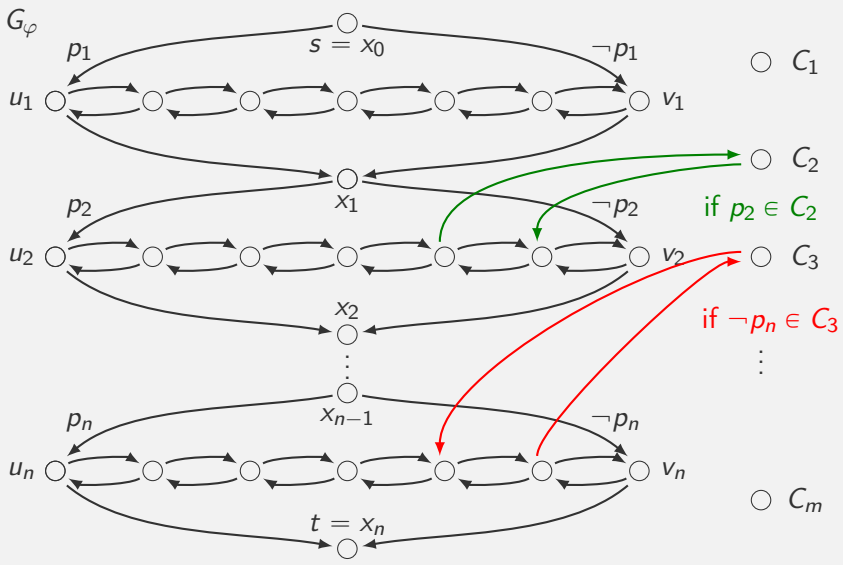
HP is NP-complete

Proof: It's in NP, since a list P of size the number of vertices can be constructed in polynomial time. Being P a permutation of the vertices and then being an H-path are verifiable in polynomial time.

SAT$\leqslant_p$ HP. For an arbitrary CNF φ it is enough to construct (G_φ, s, t) with the property of φ satisfiable $\Leftrightarrow$ there is an H-path in G_φ from s to t.

Let $p_1, \ldots p_n$ be the atoms appearing in φ and $C_1, \ldots C_m$ be the terms of φ.

NP-completeness of directed $s \rightsquigarrow t$ H-path



NP-completeness of directed $s \rightsquigarrow t$ H-path

the construction of (G_φ, s, t) :

- $\forall 1 \leqslant i \leqslant n : (x_{i-1}, u_i), (x_{i-1}, v_i), (u_i, x_i), (v_i, x_i) :\in E(G_\varphi)$
- $s := x_0, t := x_n$
- $\forall 1 \leqslant i \leqslant n$ there is path in both direction between u_i and v_i with $3m-1$ inner vertices $w_{i,1}, \ldots, w_{i,3m-1}$.
- All $w_{i,k}$'s are connected by at most one C_j and connected by none of the if k is divisible by 3.
- If $p_i \in C_j$ then let $(w_{i,3j-2}, C_j)$ and $(C_j, w_{i,3j-1}) :\in E(G_\varphi)$. (positive visit)
- If $\neg p_i \in C_j$ then let $(w_{i,3j-1}, C_j)$ and $(C_j, w_{i,3j-2}) :\in E(G_\varphi)$. (negative visit)

Positive traversal of $u_i v_i$: $u_i \rightsquigarrow v_i$.
Negative traversal of $u_i v_i$: $u_i \leftsquigarrow v_i$.

- An $s \rightsquigarrow t$ H-path contains (x_{i-1}, u_i) or (x_{i-1}, v_i) but not both $(\forall 1 \leqslant i \leqslant n)$. In the first case, the path should be continued by a positive traversal of $u_i v_i$. In the second case it should be continued by a negative one.

NP-completeness of directed $s \rightsquigarrow t$ H-path

- A visit of C_j from $w_{i,3j-1}$ on a positive traversal of $u_i v_i$ would make $w_{i,3j}$ (or v_i for $j = m$) unvisitable. If $(w_{i,3j-2}, C_j)$ is not followed by $(C_j, w_{i,3j-1})$ on an H-path then $w_{i,3j-1}$ is unvisitable. So, to have C_j on an H-path we have 2 options. Either a positive visit from a positive traversal or a negative visit from a negative traversal of $u_i v_i$ for some $1 \leqslant i \leqslant n$.
- Let p_i be true in interpretation I if an $s \rightsquigarrow t$ H-path P includes the edge (x_{i-1}, u_i), false otherwise. Then a visit of the node C_j $(1 \leqslant j \leqslant m)$ implies a true literal in the term C_j. P visits all C_j's implying $I \models \varphi$.
- If φ is satisfiable, then choose an interpretation I and a true literal for each term. Choose the positive traversal of $u_i v_i$ if p_i is true, otherwise choose the negative one. Visit each C_j $(1 \leqslant j \leqslant m)$ on the $u_i v_i$ traversal corresponding to the chosen literal. We get an H-path from s to t.

G_φ can be constructed in polynomial time. So SAT$\leqslant_p$ HP holds. HP is in NP, implying the NP-completeness of HP. □

NP-completeness of undirected $s \rightsquigarrow t$ H-path

Observation: UHP and UHC are in NP by the same reason as HP.

Theorem

UHP is NP-complete.

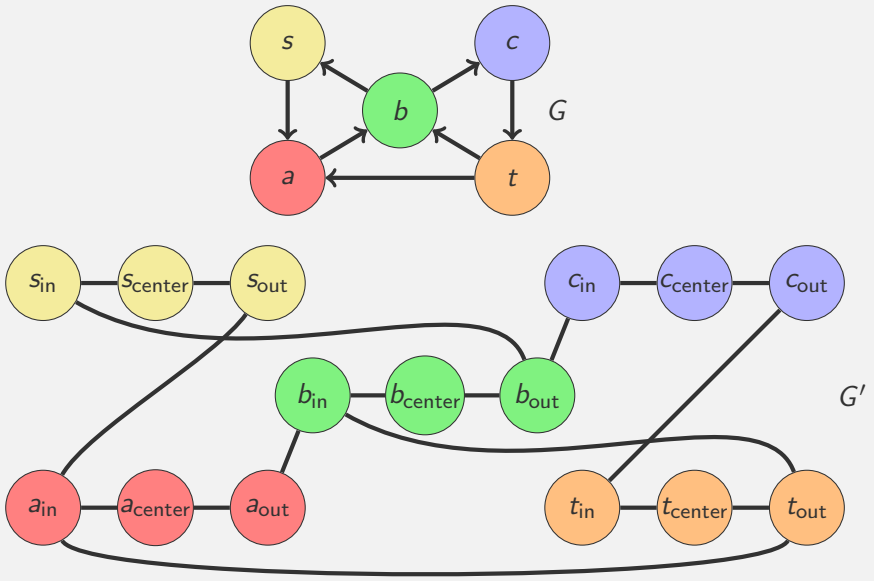
Proof: It is enough to prove, that HP$\leqslant_p$ UHP.

A polynomial time reduction $f : \langle G, s, t \rangle \to \langle G', s', t' \rangle$ is needed with the property G has a directed H-path from s to t iff G' has an undirected H-path between s' and t'. (G is a directed, G' is an undirected graph. s, t are vertices of G. s', t' are vertices of G'.)

So let G be a directed graph and s, t be two of its vertices. We construct (G', s', t') as follows.

Let 3 nodes correspond to each vertex v of G in G': v_{in}, v_{center} and v_{out} and add the edges $\{v_{\text{in}}, v_{\text{center}}\}$ and $\{v_{\text{center}}, v_{\text{out}}\}$ to the edge set $E(G')$ of G'. Furthermore, add the edge $\{u_{\text{out}}, v_{\text{in}}\}$ to $E(G')$ for every edge (u, v) of G. Let $s' := s_{\text{in}}, t' := t_{\text{out}}$.

NP-completeness of undirected $s \rightsquigarrow t$ H-path



NP-completeness of undirected $s \rightsquigarrow t$ H-path

Easy to check that this is a polynomial time reduction.

- (G', s', t') can be constructed from (G, s, t) in polynomial time.
- if G has a directed H-path from s to t, then we get a H-path between s' and t' by visiting the corresponding vertices in the order v_{in}, v_{center}, v_{out}. According to the construction of G' if the H-path in G uses edge (u, v), then there is an edge $\{u_{\text{out}}, v_{\text{in}}\}$ for the corresponding H-path in G' to use.
- if G' has an H-path between s' and t' then for each vertex v of G the vertices v_{in}, v_{center}, v_{out} of G' should be visited and since v_{center} has degree 2, they should be visited in this consecutive manner. All further edges are of type $\{u_{\text{out}}, v_{\text{in}}\}$ corresponds to an edge (u, v) in G, so by replacing the triples v_{in}, v_{center}, v_{out} in the H-path of G' between s' and t' by v we get an H-path of G from s to t. □

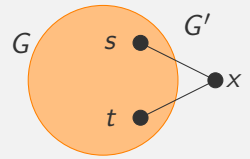
NP-completeness of undirected H-cycle

Theorem

UHC is NP-complete

Proof: UHP$\leqslant_p$ UHC. For a given (G, s, t) we construct G' as follows. Add a new vertex x to G and add two new edges $\{s, x\}$ and $\{t, x\}$ to the edge set.

- G' can be constructed from (G, s, t) in polynomial time.
- If G has an H-path $P : s \rightsquigarrow t$, then G' has an H-cycle. Just add edges $\{s, x\}$ and $\{t, x\}$ to P.



- If G' has an H-cycle, then it contains $\{s, x\}$ and $\{t, x\}$, since x has degree 2 and the only way for a degree 2 vertex to be included in a cycle is that the two incident edges are included as well. Leaving $\{s, x\}$, $\{t, x\}$ and x from the H-cycle we get an H-path between s and t in G. □

NP-completeness of Travelling Salesman Problem

Travelling Salesman Problem (TSP): What is the minimum cost a travelling salesman can visit all of his/her custumers.

More formally:

Function problem: Given an undirected graph G with nonnegative weights on its edges. Determine an H-cycle with the smallest possible total weights (if there's any at all).

Decision problem:
TSP=$\{\langle G, k\rangle \mid$ weighted, undirected graph G has an H-cycle with total weights $\leqslant k\}$.

Theorem

TSP is NP complete.

Proof: TSP$\in$ NP, due to similar reasons as HP. (The total weight condition is verifiable in polynomial time, too.)
UHC$\leqslant_p$ TSP. $G' := G$, all weights are equal to 1 and let $k := |V|$. Easy to see, that G has an H-cycle $\Leftrightarrow$ G' has an H-cycle of total weight $\leqslant k$. □

Fundamentals of theory of computation 2

10th lecture

lecturer: Tichler Krisztián
ktichler@inf.elte.hu

System of linear diophantic inequalites

SLDI= $\{\langle \mathbf{A}, \mathbf{b}\rangle |$ system of inequalities $\mathbf{Ax} \leqslant \mathbf{b}$ with integer coefficients have an integer solution$\}$.

Theorem

SLDI is NP-hard.

Proof: 3SAT can be reduced to SLDI in polynomial time. Let φ be a 3CNF using the variables $x_1, \ldots, x_n$. Add the inequalities $0 \leqslant x_i \leqslant 1$ to the system. Furthermore, if $L_1 \vee L_2 \vee L_3$ φ is a term of φ, then add the inequalities $t_1 + t_2 + t_3 \geqslant 1$ where $t_i = x_j$ if $L_i = x_j$ and $t_i = 1 - x_j$ if $L_i = \neg x_j$ $(i = 1, 2, 3)$.

Easy to check that this system of linear inequalites has an integer solution if and only if φ is satisfiable. (1 corresponds to 'true', 0 corresponds to 'false'.) □

System of linear diophantic inequalites

Remark 1 SLDI is NP-complete as well. To prove that SLDI is in NP we need an upper bound on the size of the solution. A polynomial bound can be given, but it is not easy prove this bound, this is due to the possibly negative coeffients.

Remark 2 Solving a system of general (not neccessarily linear) diophantic inequalities is undecidable. Even solving a general diophantic equation (Hilbert's 10th problem) is undecidable (Jurij Matyijaszevich, 1970). This is not surprising, since this class of problems incude Fermat's last 'theorem' ($a^n + b^n = c^n$ has no solution if $n > 2$). Fermat's last 'theorem' was a long standing open problem eventually solved by Andrew Wiles in 1995.

Subset sum problem

SUBSET SUM:= $\{\langle S, K\rangle | S$ is a multiset of integers, $K \in \mathbb{Z}$, and there is a submultiset S' of S, such that sum of the numbers in S' is exactly $K\}$.

Example: $S = \{5, 8, 9, 13, 17, 17\}, K = 27$
Then $\langle S, K\rangle \in$ SUBSET SUM, since 5+9+13=27.

Theorem

SUBSET SUM is NP-complete.

Proof: A submultiset S' can be produced in polynomial time. For a specific submultiset S' it can be checked in polynomial time whether the sum of the numbers in S' equal to K or not. So SUBSET SUM$\in$ NP.

We prove that 3SAT$\leqslant_p$SUBSET SUM.
Let φ be 3CNF with n atoms and m clauses.

S contains $3m + 2n$ pieces of $n + m$ digit integers. jth digit of a number means jth digit from right to left in this proof.

Subset sum problem

The first m digits correspond to the clauses, the last n digits correspond to the n atoms.

- for the ith atom we assign 2 integers. In both of them the $(m+i)$th bit is 1. Furthermore in the first one exactly those digits at position of at most m are equal to 1, where the corresponding clause contain x_i, while the second have a 1 in exactly those positions, where the corresponding clause have a $\neg x_i$. Let the other digits be 0.
- For the jth clause we assign 3 numbers. Each of them has a single non-0 digit. The only exception is the jth digit, let it be 5,6 and 7 in the 3 numbers.

Example: $\varphi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4)$.

The numbers:

000101 001000 010000 100001
000100 001010 010011 100010
000005 000006 000007
000050 000060 000070

Subset sum problem

Observe, that in the sum of all numbers the first m digit add up to exacly 21, while the last n digits add up to exactly 2.

Let the base of the number system be a number greater than 21, say 32. Let K be a number having an 8 for the first m digits and 1 for the last n digits (from right to left). Since 21<32 the only way for a sum of the numbers to be K is that the sum equals to 8 for the first m digits and equals to 1 for the last n digit.

If φ is satisfiable, then there exists an interpretation I evaluating φ for TRUE. Select those n numbers from the first $2n$ ones corresponding to the true literals of I. The sum of these numbers equals to 1 for the digits $(m+1)$ $(m+n)$ while it is equal to 1, 2 or 3 for digits 1 to m according to the the number of true literals in the corresponding clause. The sum for these digits can be complemented to 8 by the remaining $3m$ numbers as there is a 7, 6 and a 5.

Subset sum problem

Now, consider a subset S' having a sum equal to K. The only way this is possible is that we have chosen exactly n numbers from the first $2n$, exactly one out of the two possibilities for each digit. Let x_i be true if and only if the first option was chosen the for the ith variable. S' contains exactly one out of the three possibilities for jth digit ($1 \leqslant j \leqslant m$), since 0 would give a sum 0, 1, 2 or 3, while 2 or more would give a sum at least 11. In order to get 8 we need 1, 2 or 3 1's for the jth digit. This corresponds to having 1,2 or 3 true literals for every clause implying φ being true in this interpretation. So φ is satisfiable.

Since S consists of $O(n+m)$ pieces of $O(n+m)$ digit numbers and K being a $O(n+m)$ digit number as well the mapping $\varphi \mapsto (S, K)$ is computable in polynomial time. So 3SAT is reducible to SUBSET SUM in polynomial time implying the NP-completeness of SUBSET SUM. □

Knapsack problem

The language KNAPSACK consists of $(2n+2)$-tuples of non-negative numbers $a_1, \ldots, a_n, b, p_1, \ldots p_n, k$ such that there exist a subset of the indices $I \subseteq \{1, \ldots n\}$ satisfying the inequalities $\sum_{i \in I} a_i \leqslant b$ and $\sum_{i \in I} p_i \geqslant k$.

Story: Given a cave of n treasures. The ith treasure has a weight (or volume) a_i and has a value (or profit) p_i. Can we bring out a sum of values k or more if the capacity of our knapsack (backpack) is b? (We assume, that a set of treasures can packed in the knapsack whenever the sum of their weights do not exceed the capacity.)

Theorem

KNAPSACK is NP-complete.

Proof: KNAPSACK is in NP, since we can produce a subset I of the items in polynomial time. Checking the 2 inequalities can be done in polynomial time, as well.

Knapsack problem

SUBSET SUM $\leqslant_p$ KNAPSACK:
Let (S, K) be an input of SUBSET SUM, where $S = \{s_1, \ldots, s_n\}$.

Let $a_i := s_i, p_i := s_i, b := K, k := K$ for every $1 \leqslant i \leqslant n$-re.

If $\sum_{i \in I} s_i = K$ holds for some $I \subseteq \{1, \ldots n\}$, then $\sum_{i \in I} a_i = K = b \leqslant b$ and $\sum_{i \in I} p_i = K = k \geqslant k$ hold.

If we have $\sum_{i \in I} a_i \leqslant b = K$ and $\sum_{i \in I} p_i \geqslant k = K$ for some $I \subseteq \{1, \ldots n\}$, then $\sum_{i \in I} s_i = K$ holds since $a_i = p_i = s_i$ holds for every $1 \leqslant i \leqslant n$.

$a_1, \ldots, a_n, b, p_1, \ldots p_n, k$ can be computed for (S, K) in polynomial time, so SUBSET SUM $\leqslant_p$ KNAPSACK, it follows that KNAPSACK is NP-hard, due to the inclusion in NP it is NP-complete as well. □

Partition problem

PARTITION:=$\{\langle B \rangle \mid B$ is a multiset of numbers, that can be partioned into 2 equal parts$\}$.

Example: The entry 2,2,2,3,3,4 is in PARTITION, since 2+2+4=2+3+3.

Partition problem

Theorem

PARTITION is NP-complete.

Proof: PARTITION is in NP since producing a subset and summing the weights for the 2 parts can be done in polynomial time.

We show that SUBSET SUM $\leqslant_p$ PARTITION. Let $S = \{s_1, \ldots, s_m\}$ and b be an entry for SUBSET SUM. We can assume, that $b \leqslant s = \sum_{i=1}^{m} s_i$. $B := \{s_1, \ldots, s_m, s + 1 - b, b + 1\}$. Then $\langle B \rangle \in$ PARTITION $\Longleftrightarrow \langle S, b \rangle \in$ SUBSET SUM.

Observe, that sum of all weights in B is $2s + 2$. Sum of the last two weights is $s + 2$, more than half of the total sum. Therefore the last 2 weights can not be in the same part of the partition. So if B can be halved if and only if there is a subset of $\{s_1, \ldots, s_m\}$ having a sum of b.

B can be constructed from (S, b) in polynomial time so PARTITION is NP-complete. □

Bin packing

As a computational (optimization) problem: How many bins of unit capacity is needed to pack all items of a set if the weights in the set are $s_1, \ldots, s_n (\leqslant 1)$? (Sum of the weights in each bin can not exceed 1.)

Decision problem: Can we pack items of weights $s_1, \ldots, s_n$ in k bins of unit capacity?

BIN PACKING:=$\{\langle s_1, \ldots, s_n, k \rangle \mid$ the weights $s_i \in \mathbb{Q}^+ (1 \leqslant i \leqslant n)$ can be partitioned into $k \in \mathbb{N}^+$ parts, such that, for every part the sum of the weights is $\leqslant 1\}$.

Example: For the items of weights 0,34; 0,44; 0,54; 0,64 putting the first 2 in the same bin results 3 bins, but it is possible to place the items in just 2 bins.

Bin packing

Theorem

BIN PACKING is NP-complete.

Proof: BIN PACKING is in NP, since a NTM can produce all possible partitions on the branches of a nondeterministic configuration tree in polynomial time and checking for all bins whether the items selected for the bin fit in can be done in polynomial time as well.

PARTITION $\leqslant_p$ BIN PACKING:

Let $b_1, \ldots, b_n$ be the elements of the multset B and let $b = \sum_{i=1}^{n} b_i$. Let elements of a multiset L be $2b_1/b, \ldots, 2b_n/b$. It is easy to see, that $\langle B \rangle \in$ PARTITION $\iff \langle L, 2 \rangle \in$ BIN PACKING.

The reduction can be computed in polynomial time. □

Possible structures of the NP class

Definition

L is called NP-intermediate, if $L \in$ NP, $L \notin$ P and L is not NP-complete.

Ladner's Theorem

If P $\neq$ NP, then NP-intermediate languages exist.

(without proof)

Since we do not know the answer for P $\stackrel{?}{=}$ NP no NP-intermediate language is guaranteed to exist. Most probably they exist, as we conjecture, that P $\neq$ NP.

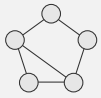
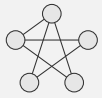
There are some languages for which neither its inclusion in P, nor its NP-completeness is proved despite of intense research. Such languages are strong NP-intermediate candidates.

NP-intermediate candidates – graph isomorphism

Definition

Undirected graphs $G_i = (V_i, E_i)$ $(i = 1, 2)$ are called isomorphic, if there is a bijection $f : V_1 \to V_2$ with the property $\{u, v\} \in E_1 \Leftrightarrow \{f(u), f(v)\} \in E_2$ for every $u, v \in V_1$.

GRAPH ISOMORPHISM$=\{\langle G_1, G_2 \rangle |$ G_1 and G_2 are undirected isomorphic graphs$\}$.

Example:  and  are isomorphic graphs.

Remark: Graph isomorphism is in P for several practical special cases. E.g., for

- trees
- planar graphs
- graphs of bounded degree

NP-intermediate candidates – graph isomorphism

A result from 2017 due to Babai.

Theorem: GRAPH ISOMORPHISM $\in$ QP, where

$$\text{QP} = \bigcup_{c \in \mathbb{N}} \text{TIME}(2^{(\log n)^c})$$

is the complexity class of problems decidable in "quasipolynomial" time.

The following generalization of graph isomorphism is NP-complete.

SUBGRAPH ISOMORPHISM $= \{\langle G_1, G_2 \rangle |$ undirected graph G_1 is isomorphic with a subgraph of undirected graph $G_2\}$.

SUBGRAPH ISOMORPHISM is NP-complete

Remark: Here, subgraph is not an induced subgraph. E.g., $G_1 = (\{a, b, c\}, \{\{a, b\}, \{b, c\}\})$, a path of 2 edges is a subgraph of G_2=$(\{A, B, C, D\}, \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\})$, the complete graph on 4 vertices. G_1 is isomorphic with $G_3 = (\{A, B, C\}, \{\{A, B\}, \{B, C\}\})$ which is considered to be a subgraph of G_2 despite $\{A, C\} \in E(G_2)$.

Theorem

SUBGRAPH ISOMORPHISM is NP-complete.

Proof: It is in NP, since an NTM can produce an injective mapping f from $V(G_1)$ to $V(G_2)$, then it can check for all edges e of G_1 whether $f(e)$ is an edge of G_2 as well in polynomial time.

UHC $\leqslant_p$ SUBGRAPH ISOMORPHISM. Let G be an arbitrary undirected graph. Let G_1 be a cycle of $|V(G)|$ vertices and $G_2 = G$. Then G has a Hamiltonian cycle if and only if G_2 has a subgraph isomorphic to G_1. The reduction is polynomial in the size of G. □

NP-intermediate candidates – prime factorization

Function problem:

Prime factorization: Give the prime factors of natural number.

Decision problem version:

PRIME FACTORIZATION =
$$\{\langle n, k\rangle \mid n \text{ has a prime factor of size at most } k\}$$

An application:

RSA is a public-key cryptosystem that is widely used for secure data transmission introduced by Ron Rivest, Adi Shamir and Len Adleman in 1976. It is based on the following idea. If two huge primes are multiplied there is no efficient (polynomial) algorithm exists which determines the prime factors.

We can not prove that PRIME FACTORIZATION $\notin$ P. But still the procedure is widly used for secure data transmission. This shows how strong is our belief that P $\neq$ NP.

co$\mathcal{C}$

Definition

If $\mathcal{C}$ is a complexity class, then let co$\mathcal{C} = \{L \mid \overline{L} \in \mathcal{C}\}$.

Definition

$\mathcal{C}$ is **closed under polynomial time reduction** if whenever $L_2 \in \mathcal{C}$ and $L_1 \leqslant_p L_2$ holds then $L_1 \in \mathcal{C}$ holds, too.

Earlier: P and NP are closed under polynomial time reduction.

Theorem

If $\mathcal{C}$ is closed under polynomial time reduction then so is co$\mathcal{C}$.

Proof: Let $L_2 \in$co$\mathcal{C}$ and L_1 be languages satisfying $L_1 \leqslant_p L_2$.

$L_1 \leqslant_p L_2$ implies $\overline{L}_1 \leqslant_p \overline{L}_2$ (the same reduction is good).

Since $\overline{L}_2 \in \mathcal{C}$, we get $\overline{L}_1 \in \mathcal{C}$ by $\mathcal{C}$ being closed under polynomial time reduction. I.e., $L_1 \in$co$\mathcal{C}$. □

Corollary: coNP is closed under polynomial time reduction.

co$\mathcal{C}$

Is it true that P=coP? Yes. (Exchanging q_a and q_r in a TM deciding L we get a TM deciding $\overline{L}$ in polynomial time.)

Is it true, that NP=coNP? The above construction does not work, it may not decide $\overline{L}$.

Theorem

L is $\mathcal{C}$-complete $\iff$ $\overline{L}$ is co$\mathcal{C}$-complete.

Proof: If $L \in \mathcal{C}$, then $\overline{L} \in$ co$\mathcal{C}$.

Let L' be in $\mathcal{C}$ satisfying $L' \leqslant_p L$. Then $\overline{L'} \leqslant_p \overline{L}$ holds, too. (The same reduction is good.)

$L' \mapsto \overline{L'}$ is a bijection between $\mathcal{C}$ and co$\mathcal{C}$. So all languages of co$\mathcal{C}$ is reducible to $\overline{L}$ in polynomial time.

So both $\overline{L} \in$ co$\mathcal{C}$ and $\overline{L}$ being co$\mathcal{C}$-hard holds, therefore $\overline{L}$ is co$\mathcal{C}$-complete. □

Examples for coNP-complete languages

UNSAT := $\{\langle\varphi\rangle \mid \varphi$ is an unsatisfiable propositional formula$\}$.

VALID := $\{\langle\varphi\rangle \mid$ a φ is a valid propositional formula $\}$.

Theorem

UNSAT and VALID are coNP-complete.

Proof: Easy to see, that GENSAT $= \{\langle\varphi\rangle \mid \varphi$ is a satisfiable propositional formula$\}$ is NP-complete. $\overline{\text{GENSAT}} =$ UNSAT. So, by the previous Theorem UNSAT is coNP-complete.

UNSAT $\leqslant_p$ VALID, since $\varphi \mapsto \neg\varphi$ is a polynomial time reduction.□

Informally: coNP contains the problems for which there is a polynomial time *refutation* for the "no" instances.

E.g., an interpretation satisfying φ is a refutation of φ being unsatisfiable. An interpretation evaluating φ for FALSE refutes φ being valid. An interpretation can be produced in polynomial time, and a formula can be evaluated in polynomial time in a given interpretation.

A property of coNP-complete languages

Theorem

If L is coNP-complete and $L \in$ NP, then NP = coNP.

Proof: Let $L' \in$ coNP be arbitrary. Since L is coNP-complete we have $L' \leqslant_p L$. But L is in NP, so by NP being closed under polynomial time reduction we have $L' \in$ NP. Therefore coNP $\subseteq$ NP.

Let $L \in$ NP be arbitrary, then by the definition of coNP we have $\bar{L} \in$ coNP. Due to coNP $\subseteq$ NP we have $\bar{L} \in$ NP. By the definition of coNP we get $L \in$ coNP. So NP $\subseteq$ coNP holds as well. □

Conjecture: NP $\neq$ coNP.

If the conjecture is true UNSAT and VALID are not even in NP. At least there is no polynomial witness is known for the membership of these languages.

NP ∩ coNP

If NP $\neq$ coNP then the class NP ∩ coNP is of interest. Obviously P $\subseteq$ NP ∩ coNP. Let us consider some languages from NP ∩ coNP.

Example 1

CONNECTIVITY:= $\{\langle G\rangle \mid G =$ (V, E) is undirected and connected$\}$

CONNECTIVITY is in NP-beli, since $\binom{n}{2}$ paths between the pairs of the vertices is proof for connectedness. Each paths have length n. An NTM can produce $\binom{n}{2}$ paths of length n, on one each of its computational branches in polynomial time. Checking whether they are paths in fact can be produced in polynomial time, as well.

CONNECTIVITY is in coNP, since a partition into 2 parts, X and $V\backslash X$ can be produced in polynomial time. Checking whether there's no edge between X and $V\backslash X$ can be done in polynomial time. G in unconnected if such a cut exists. ó.

CONNECTIVITY is in P, connectivity can be checked by a breadth first search algorithm.

NP ∩ coNP

Example 2

Definition

Let $G = (A, B, E)$ be an undirected bipartite graph. A subset $M \subseteq E$ of the edges is called a **perfect matching**, if the degree of each vertex in the graph $(A \cup B, M)$ equals to 1. (If it is at most one, then M is called a (partial) matching.)

Remark: "Perfect matching" is also called "complete matching" in the literature.

PERFECT MATCHING:= $\{\langle G\rangle \mid$ bipartite graph $G =$ (A, B, E) has a perfect matching$\}$

PERFECT MATCHING is in NP, since a list of pairs $(a, b) \in A \times B$ of size $|A|$ can be produced in polynomial time and also it can be checked whether it is a perfect matching.

PERFECT MATCHING is in coNP due to the following theorem of Frobenius.

NP ∩ coNP

Theorem (Frobenius): Bipartite graph $G = (A, B, E)$ has a complete matacing if and only if $|A| = |B|$ and every subset $X \subseteq A$ the neighborhood of X in B has a size at least $|X|$. (Neighborhood of X: union of the set of neighbours for all elements of X.)

A subset X of A can be produced in polynomial time and the property of the theorem can be checked in polynomial time.

In fact, PERFECT MATCHING is in P. There is a polynomial time algorithm called **Hungarian method** developed by Harold Kuhn based on the works of Hungarian mathematicians Dénes Kőnig and Jenő Egerváry for finding a maximal matching in a bipartite graph.

Idea: take independent edges till you can. Then try to find alternating augmenting paths (exactly every second edge is from the matching) from outside of the current matching, say from A to another vertex outside the matching, from part B. Exchange the even edges with the odd ones.

NP ∩ coNP

Example 3 Prime testing

PRIMES:= $\{p \mid p \text{ is a prime}\}$.

Observe, that length of $p \in$ PRIMES is the number of digits in p, i.e. $\Theta(\log p)$.

It is easy to see, that PRIMES is in coNP, since a divisor of n of size at most $\sqrt{n}$ refutes n being a prime. Size of the refutation is $O(\log n)$, result of division with remainder can be calculated in $O(\log^3 n)$ time (without proof).

Remark: A deterministic brute force method till $\sqrt{n}$ is too slow, it gives only an exponential algorithm in $\log n$.

Membership of PRIMES in NP is not that easy. We need a fast prime test.

NP ∩ coNP

Theorem (Lucas's prime test) n is a prime $\Leftrightarrow$ there exist $1 \leqslant x \leqslant n-1$ having the property $x^{n-1} \equiv 1 \pmod{n}$, but $x^{(n-1)/p} \not\equiv 1 \pmod{n}$ holds for every prime divisor p of $(n-1)$.

Using this we can construct a recursive nondeterministic algorithm of time complexity polynomial in $(\log n)$:

Nondeterministic prime test(n)
if $n = 2$ **then return** 'yes';
if $n = 1$ or $n > 2$ and n even **then return** 'no';
if $n > 2$ is odd **then**
let $1 < x < n$;
check whether $x^{n-1} \equiv 1 \pmod{n}$ holds
guess the prime factors of $n-1$: $p_1, \dots p_k$
check, whether $n - 1 = p_1 \cdots p_k$ holds
check whether p_i is a prime for every $1 \leqslant i \leqslant k$
and check whether $x^{(n-1)/p_i} \not\equiv 1 \pmod{n}$
return 'yes', if all checks are passed.

NP ∩ coNP

A potential prime factorization consists of $O(\log n)$ primes, each of them of length $O(\log n)$. Calculating the powers mod n can be done in $O(\log^3 n)$. That adds up to a $O(\log^4 n)$ asymptotic upper bound.

So the following recursion holds for the time complexity for an input of $m = \lceil \log_2 n \rceil$ digits.

$T(m) \leqslant cm^4 + \sum_{i=1}^{k} T(m_i)$, where m_i is the number of digits of p_i $(1 \leqslant i \leqslant k)$.

Then $\sum_{i=1}^{k} m_i \leqslant m$ and $m_i \leqslant m-1$ for every $1 \leqslant i \leqslant k$.

We can finish the calculation by induction. We prove that $T(m) \leqslant cm^5$:

$T(m) \leqslant cm^4 + \sum_{i=1}^{k} cm_i^5 \leqslant cm^4 + c(m-1)^4 \sum_{i=1}^{k} m_i = cm(m^3 + (m-1)^4) \leqslant cm^5$

So our nondeterministic algorithm is $O(\log^5 n)$ time bounded proving PRIMES $\in$ NP.

NP ∩ coNP

It was a long standing open question to find a deterministic prime test of polynomial time complexity, i.e., it was not known whether PRIMES is in P or not.

AKS prime test: In 2002 M. Agrawal, N. Kayal, N. Saxena of India found a deterministic prime test of $O(\log^{12} n \log^{k} \log n)$ time complexity (k is a constant) proving PRIMES$\in$P. Their work was honored by the Gödel and the Fulkerson prize in 2006. Later, there was an improvement on the the time complexity to $O(\log^{6} n \log^{k} \log n)$.

These examples may suggest that every problem in NP ∩ coNP is in P in fact. Probably, this is not the case, it is conjectured by most computer scientists, that this inclusion is proper.

Conjecture: P $\subset$ NP ∩ coNP.