Introduction to Computational Complexity

A 10-lectures Graduate Course

Martin Stigge, martin.stigge@it.uu.se

Uppsala University, Sweden

13.7. - 17.7.2009

Administrative Meta-Information

- 5-Day course, Monday (13.7.) to Friday (17.7.)
- Schedule:

Mon (13.7.):	10:00 - 12:00, 16:30 - 18:30
Tue (14.7.):	10:00 - 12:00, 14:00 - 16:00
Wed (15.7.):	10:00 - 12:00, 14:00 - 16:00
Thu (16.7.):	10:00 - 12:00, 14:00 - 16:00
Fri (17.7.):	10:00 - 12:00, 16:30 - 18:30

- Lecture notes avaiable at:

  `http://www.it.uu.se/katalog/marst984/cc-st09`

- Some small assignments at end of day
- Course credits: ??
- Course is interactive, so:

  Any questions so far?

What is Computational Complexity?

- Studies *intrinsic complexity* of computational tasks
- *Absolute Questions:*
  - How much time is needed to perform the task?
  - How much resources will be needed?
- *Relative Questions:*
  - More difficult than other tasks?
  - Are there "most difficult" tasks?
- (Surprisingly: Many *relative* answers, only few *absolute* ones..)
- *Rigorous* treatment:
  - Mathematical formalisms, minimize "hand-waving"
  - Precise definitions
  - Theorems have to be proved

After all: Complexity *Theory*

What is Computational Complexity? (Cont.)

- Basis: *Computability Theory*
  - Provides models of computation
  - Explores their strength (expressiveness)
  - Question: "What can be computed (at all)?"
- Then: *Complexity Theory*
  - Tries to find meaningful complexity measures
  - Tries to classify and relate problems
  - Tries to find upper and lower complexity bounds
  - Question: "What can efficiently be computed?"
- One core concern:
  - What does "efficiently" actually mean?
  - *Proving* vs. *Verifying* (P $\stackrel{?}{=}$ NP problem)

Course Outline

0. Introduction
1. Basic Computability Theory
   - Formal Languages
   - Model of Computation: Turing Machines
   - Decidability, Undecidability, Semi-Decidability
2. Complexity Classes
   - Landau Symbols: The $\mathcal{O}(\cdot)$ Notation
   - Time and Space Complexity
   - Relations between Complexity Classes
3. Feasible Computations: P vs. NP
   - Proving vs. Verifying
   - Reductions, Hardness, Completeness
   - Natural NP-complete problems
4. Advanced Complexity Concepts
   - Non-uniform Complexity
   - Probabilistic Complexity Classes
   - Interactive Proof Systems

This is a theoretical course – expect a lot of "math"!

Problems as Formal Languages

- Start with very high-level model of computation
- Assume a *machine* with *input* and *output*
- Formal notation for format:
  - $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite set of *symbols*
  - $w = (w_1, \ldots, w_l)$ is a *word over* Σ: $\forall i : w_i \in \Sigma$
    - ★ Write also just $w_1 w_2 \ldots w_l$
  - l is the *length* of w, also denoted $|w|$
  - ε is the *empty word*, i.e., $|\varepsilon| = 0$
  - Σ^k is the set of words of length k
  - $\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$ are all words over Σ
  - A *language* is a set $L \subseteq \Sigma^*$
  - Let $L_1, L_2 \subseteq \Sigma^*$, language operations:
    - ★ $L_1 \cup L_2$ (*union*), $L_1 \cap L_2$ (*intersection*), $L_1 - L_2$ (*difference*)
    - ★ $\overline{L} := \Sigma^* - L$ (*complement*)
    - ★ $L_1 L_2 := \{w \mid \exists w_1 \in L_1, w_2 \in L_2 : w = w_1 w_2\}$ (*concatenation*)

Problems as Formal Languages (Example)

Example: NAT

- Let $\Sigma := \{0, 1, \ldots, 9\}$
- Σ^* is all strings with digits
- Let $[n]_{10}$ denote *decimal representation* of $n \in \mathbb{N}$
- NAT $:= \{[n]_{10} \mid n \in \mathbb{N}\} \subsetneq \Sigma^*$ all representations of naturals
  - $010 \in \Sigma^* - \text{NAT}$
- Machine for calculating square:

  Input: $w = [n]_{10} \in \Sigma^*$

  Output: $v \in \Sigma^*$ with $v = [n^2]_{10}$

  (Plus error-handling for $w \notin$ NAT)

Problems as Formal Languages (2nd Example)

Example: PRIMES

- Let $\Sigma := \{0, 1, \ldots, 9\}$ and NAT as before
- PRIMES $:= \{[p]_{10} \mid p \text{ is a prime number}\}$
- Clearly: PRIMES $\subsetneq$ NAT
- Machine M for checking primality:

  Input: $w = [n]_{10} \in \Sigma^*$

  Output: 1 if n is prime, 0 otherwise
- This is a *decision problem*:
  - PRIMES are the *positive instances*,
  - Σ^* – PRIMES (everything else) the *negative instances*
  - M has to distinguish both (it *decides* PRIMES)

Important concept, will come back to that later!

Model of Computation

- Input/Output format defined. What else?
- Model of Computation should:
  - Define what we mean with "computation", "actions", ...
  - Be *simple* and *easy* to use
  - ... but yet powerful
- Models: Recursive Functions, Rewriting Systems, Turing Machines, ...
- All equally powerful:

Church's Thesis

All "solvable" problems can be solved by any of the above formalisms.

- We will focus on the *Turing Machine*.

Turing Machine

- Turing Machines are like simplified computers containing:
  - A *tape* to read/write on
    - ★ Contains squares with one symbol each
    - ★ Is used for input, output and temporary storage
    - ★ Unbounded
  - A read/write *head*
    - ★ Can change the symbol on the tape at current position
    - ★ Moves step by step in either direction
  - A *finite state machine*
    - ★ Including an initial state and final states
- Looks simple, but is very powerful
- Standard model for the rest of the course

Turing Machine: Definition

Definition (Turing Machine)

A Turing machine M is a five-tuple $M = (Q, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of *states*;
- Γ is the *tape alphabet* including the blank: $\square \in \Gamma$;
- q_0 is the *initial state*, $q_0 \in Q$;
- F is the *set of final states*, $F \subseteq Q$;
- δ is the *transition function*, $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{R, N, L\}$.

Operation:

- Start in state q_0, input w is on tape, head over its first symbol
- Each step:
  - Read current state q and symbol a at current position
  - Lookup $\delta(q, a) = (p, b, D)$
  - Change to state p, write b, move according to D
- Stop as soon as $q \in F$. Left on tape: Output

Turing Machine: Configuration

- *Configuration* (w, q, v) denotes status after each step:
  - Tape contains wv (with infinitely many $\Box$ around)
  - Head is over first symbol of v
  - Machine is in state q
- Start configuration: (ε, q_0, w) if input is w
- End configuration: (v, q, z) for a $q \in F$
  - Output is z, denoted by $M(w)$
  - In case machine *doesn't halt (!)*: $M(w) = \nearrow$

Turing Machine: Step Relation

- Step relation: Formalizes semantics of Turing machine

Definition (Step Relation)

Let $M = (Q, \Gamma, \delta, q_0, F)$, define $\vdash$ for all $w, v \in \Gamma^*$, $a, b \in \Gamma$ and $q \in Q$ as:

$$(wa, q, bv) \vdash \begin{cases} (wac, p, v) & \text{if } \delta(q, b) = (p, c, R), \\ (wa, p, cv) & \text{if } \delta(q, b) = (p, c, N), \\ (w, p, acv) & \text{if } \delta(q, b) = (p, c, L). \end{cases}$$

- α reaches β in 1 step: $\alpha \vdash \beta$
- α reaches β in k steps: $\alpha \vdash^k \beta$
- α reaches β in any number of steps: $\alpha \vdash^* \beta$

Turing Machine: The Universal Machine

- Turing machine model is quite simple
- Can be easily simulated by a human
  - Provided enough pencils, tape space and patience
- Important result: *Machines can simulate machines*
  - Turing machines are *finite* objects!
  - Effective encoding into words over an alphabet
  - Also configurations are *finite*! Encode them also
- Simulator machine U only needs to
  - Receive an encoded M as input
  - Input of M is w, give that also to U
  - U maintains encoded configurations of M and applies steps

Turing Machine: The Universal Machine (Cont.)

- Let $\langle M \rangle$ be encoding of machine M.

Theorem (The Universal Machine)

There exists a universal Turing machine U, such that for all Turing machines M and all words $w \in \Sigma^*$:

$$U(\langle M \rangle, w) = M(w)$$

In particular, U does not halt iff [1] M does not halt.

(Without proof.)

[1] "if and only if"

Turing Machine: Transducers and Acceptors

- Definition so far: Receive input, compute output
- We call this a *transducer*:
  - Interpret a TM M as a function $f : \Sigma^* \to \Sigma^*$
  - All such f are called *computable functions*
  - *Partial functions* may be undefined for some inputs w
    - ⋆ In case M does not halt for them ($M(w) = \nearrow$)
  - *Total functions* are defined for all inputs
- For decision problems L: Only want a positive or negative answer
- We call this an *acceptor*:
  - Interpret M as halting in
    - ⋆ Either state q_{yes} for positive instances $w \in L$
    - ⋆ Or in state q_{no} for negative instances $w \notin L$
  - Output does not matter, only final state
  - M *accepts* the language $L(M)$:

$$L(M) := \{w \in \Sigma^* \mid \exists y, z \in \Gamma^* : (\varepsilon, q_0, w) \vdash^* (y, q_{yes}, z)\}$$

- Rest of the course: Mostly acceptors

Turing Machine: Multiple Tapes

- Definition so far: Machine uses *one* tape
- More convenient to have k tapes (k is a constant)
  - As dedicated input/output tapes
  - To save intermediate results
  - To precisely measure used space (except input/output space)
- Define this as *k-tape Turing machines*
  - Still only one state, but k heads
  - Equivalent to 1-tape TM in terms of expressiveness
    (Encode a "column" into one square)
  - Could be more efficient, but not much
- Rest of the course: k-tape TM with dedicated input/output tapes

Turing Machine: Non-determinism

- Definition so far: Machine is *deterministic*
  - Exactly one next step possible
- Extension: Allow different possible steps

$$\delta : (Q - F) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{R, N, L\})$$

- Machine chooses *non-deterministically* which step to do
  - Useful to model uncertainty in a system
  - Imagine behaviour as a *computation tree*
  - Each path is one possible computation
  - Accepts w iff there is a path to q_{yes} (*accepting path*)
- Not a *real machine*, rather a *theoretical model*
- Will see another characterization later
- Expressiveness does *not* increase in general (see following Theorem)

Turing Machine: Non-determinism (Cont.)

Theorem

Given a non-deterministic TM N, one can construct a deterministic TM M with $L(M) = L(N)$.
Further, if $N(w)$ accepts after $t(w)$ steps, then there is c such that $M(w)$ accepts after at most $c^{t(w)}$ steps.

Remark

- *Exponential* blowup concerning speed
- Ignoring speed, expressiveness is the same
- Note that N might not terminate on certain inputs

Turing Machine: Non-determinism (Cont. 2)

Proof (Sketch).

- Given a non-deterministic N and an input w
- Search the computation tree of N
- Breadth-first technique: Visit all "early" configurations first
  - Since there may be infinite paths
  - For each $i \geq 0$, visit all configurations up to depth i
  - If N *accepts* w, we will find accepting configuration at a depth t and halt in q_{yes}
  - If N *rejects* w, we halt in q_{no} or don't terminate
- Let d be maximal degree of non-determinism (choices of δ)
- Above takes at most $\sum_{i=0}^{t} d^i$ steps
- Can be bounded from above by c^t with a suitable constant c

Summary (Turing Machine)

- *Simple* model of computation, but *powerful*
- Clearly defined syntax and semantics
- May *accept* languages or *compute* functions
- May use *multiple tapes*
- *Non-determinism* does not increase expressiveness
- A *Universal Machine* exists, simulating all other machines

Remark

The machines we use from now on

- are *deterministic*
- are *acceptors*,
- with *k tapes*

(except stated otherwise).

Deciding a Problem

- Recall: Turing Machines running with input w may
  - halt in state q_{yes},
  - halt in state q_{no}, or
  - run *without halting*.
- Given problem L and instance w, want to *decide* whether $w \in L$:
  - Using a machine M
  - If $w \in L$, M should halt in q_{yes}
  - If $w \notin L$, M should halt in q_{no}
- In particular: Always terminate! (Little use otherwise...)

Decidability and Undecidability

Definition

L is called *decidable*, if there exists a TM M with $L(M) = L$ that *halts on all inputs*.
REC is the set of all decidable languages.

- We can *decide* the status of w by just running $M(w)$.
- Termination guaranteed, we won't wait infinitely
- "M *decides* L"
- If $L \notin$ REC, then L is *undecidable*

Decidability and Undecidability: Example

Example (PRIMES $\in$ REC)

- Recall PRIMES $:= \{[p]_{10} \mid p \text{ is a prime number}\}$
- Can be decided:
  - Given $w = [n]_{10}$ for some n
  - Check for all $i \in (1, n)$ whether n is multiple of i
  - If an i found: Halt in q_{no}
  - Otherwise, if all i negative: Halt in q_{yes}
- Can be implemented with a Turing machine
- Always terminates (only finitely many i)
- Thus: PRIMES $\in$ REC

Semi-Decidability

Definition

L is called *semi-decidable*, if there exists a TM M with $L(M) = L$.
RE is the set of all semi-decidable languages.

- Note the missing "*halts on all inputs*" !
- We can only "half-decide" the status of a given w:
  - Run M, wait for answer
  - If $w \in L$, M will halt in q_{yes}
  - If $w \notin L$, M may not halt
  - We don't know: $w \notin L$ or too impatient?
- "M *semi-decides* L"

Class Differences

- Questions at this point:
  1. Are there undecidable problems?
  2. Can we at least semi-decide some of them?
  3. Are there any we can't even semi-decide?
- Formally: $\text{REC} \overset{?}{\subsetneq} \text{RE} \overset{?}{\subsetneq} \mathcal{P}(\Sigma^*)$
- Subtle difference between REC and RE: Termination guarantee

Properties of Complementation

Theorem

1. $L \in \text{REC} \iff \overline{L} \in \text{REC}$. *("closed under taking complements")*
2. $L \in \text{REC} \iff (L \in \text{RE} \wedge \overline{L} \in \text{RE})$.

Proof (First part).

- Direction "$\Longrightarrow$":
  - Assume M decides L and halts always
  - Construct M': Like M, but swap q_{yes} and q_{no}
  - M' decides $\overline{L}$ and halts always!
- Direction "$\Longleftarrow$":
  - Exact same thing.

Properties of Complementation

Theorem

1. $L \in \text{REC} \iff \overline{L} \in \text{REC}$. *("closed under taking complements")*
2. $L \in \text{REC} \iff (L \in \text{RE} \wedge \overline{L} \in \text{RE})$.

Proof (Second part).

- Direction "$\Longrightarrow$":
  - Follows from $\text{REC} \subseteq \text{RE}$ and first part
- Direction "$\Longleftarrow$":
  - Let M_1, M_2 with $L(M_1) = L$ and $L(M_2) = \overline{L}$
  - Given w, simulate $M_1(w)$ and $M_2(w)$ step by step, in turns
  - *Eventually one of them will halt in q_{yes}*
  - If it was M_1, halt in q_{yes}
  - It it was M_2, halt in q_{no}
  - Thus, we always halt (and decide L)!

The Halting Problem

- Approach our three questions:
  1. Are there undecidable problems?
  2. Can we at least semi-decide some of them?
  3. Are there any we can't even semi-decide?
- Classical problem: *Halting Problem*
  - Given a program M (Turing machine!) and an input w
  - Will $M(w)$ terminate?
  - Natural problem of great practical importance
- Formally: Let $\langle M \rangle$ be an encoding of M

Definition (Halting Problem)

H is the set of all Turing machine encodings $\langle M \rangle$ and words w, such that *M halts on input w*:

$$H := \{(\langle M \rangle, w) \mid M(w) \neq \nearrow\}$$

Undecidability of the Halting Problem

Theorem

$H \in \text{RE} - \text{REC}$

Proof (First part).

We show $H \in \text{RE}$:

- Need to show: There is a TM M', such that
  - Given M and w
  - If $M(w)$ halts, M' accepts (halts in q_{yes})
  - If $M(w)$ doesn't halt, M' halts in q_{no} or doesn't halt
- Construct M': *Just simulate $M(w)$*
  - If simulation halts, accept (i.e. halt in q_{yes})
  - If simulation doesn't halt, we also won't
- Thus: $L(M') = H$

Undecidability of the Halting Problem (Cont.)

Theorem

$H \in \text{RE} - \text{REC}$

Proof (Second part).

We show $H \notin \text{REC}$:

- Need to show: There is no TM M_H, such that
  - Given M and w
  - If $M(w)$ halts, M_H accepts (halts in q_{yes})
  - If $M(w)$ doesn't halt, M_H rejects (halts in q_{no})
  - *Note: M_H always halts!*
- We can't use simulation!
  - What if it doesn't halt?
- New approach: Indirect proof
  - Assume there is M_H with above properties
  - Show a contradiction

Undecidability of the Halting Problem (Cont. 2)

Theorem

$H \in \mathrm{RE} - \mathrm{REC}$

Proof (Second part, cont.)

We show $H \notin \mathrm{REC}$: Assume there is M_H that always halts

- Build another machine N:
  - On input w, simulate $M_H(w, w)$
  - If simulation halts in q_{yes}, enter infinite loop
  - If simulation halts in q_{no}, accept (i.e. halt in q_{yes})
- N is Turing machine and $\langle N \rangle$ its encoding. *Does $N(\langle N \rangle)$ halt?*
- Assume "yes, $N(\langle N \rangle)$ halts":
  - By construction of N, $M_H(\langle N \rangle, \langle N \rangle)$ halted in q_{no}
  - Definition of H: $N(\langle N \rangle)$ does not halt. *Contradiction!*
- Assume "no, $N(\langle N \rangle)$ doesn't halt":
  - By construction of N, $M_H(\langle N \rangle, \langle N \rangle)$ halted in q_{yes}
  - Definition of H: $N(\langle N \rangle)$ does halt. *Contradiction!*
- N can not exist! $\Longrightarrow$ M_H can not exist.

Class Differences: Results

- Know now: $H \in \text{RE} - \text{REC}$, thus: $\text{REC} \subsetneq \text{RE}$
- What about RE and $\mathcal{P}(\Sigma^*)$?
  - Is there an $L \subseteq \Sigma^*$ that's not even semi-decidable?
- Counting argument:
  - RE is *countably* infinite: Enumerate all Turing machines
  - $\mathcal{P}(\Sigma^*)$ is *uncountably* infinite: Σ^* is countably infinite

Corollary

$\text{REC} \subsetneq \text{RE} \subsetneq \mathcal{P}(\Sigma^*)$

Remark

- Actually, we even know one of those languages: $\overline{H} \notin \text{RE}$
- Otherwise, H would be decidable: $(H \in \text{RE} \wedge \overline{H} \in \text{RE}) \implies H \in \text{REC}$

Reductions

- We saw: Some problems are harder than others
- Possible to *compare* them directly?
- Concept for this: *Reductions*
  - Given problems A and B
  - Assume we know how to solve A using B
  - Then: Sufficient to find out how to solve B for solving A
  - We *reduced* A to B
  - Consequence: A is "easier" than B
- Different formal concept established
  - Differ in how B is used when solving A
  - We use *Many-one reductions*

Reductions: Definition

Definition (Many-one Reduction)

$A \subseteq \Sigma^*$ is *many-one reducible* to $B \subseteq \Sigma^*$ ($A \leq_m B$),
if there is $f : \Sigma^* \to \Sigma^*$ (computable and total), such that

$$\forall w \in \Sigma^* : w \in A \iff f(w) \in B$$

f the *reduction function*.

- f maps positive to positive instances, negative to negative
- Impact on decidability:
  - Given problems A and B with $A \leq_m B$
  - And given M_f calculating reduction f
  - And given M_B deciding B
  - Decide A by simulating M_f and on its output M_B

Reductions: Properties

Lemma

For all A, B and C the following hold:

1. $A \leq_m B \wedge B \in \mathrm{REC} \Longrightarrow A \in \mathrm{REC}$ (*Closedness of* REC *under* $\leq_m$)
2. $A \leq_m B \wedge B \in \mathrm{RE} \Longrightarrow A \in \mathrm{RE}$ (*Closedness of* RE *under* $\leq_m$)
3. $A \leq_m B \wedge B \leq_m C \Longrightarrow A \leq_m C$ (*Transitivity of* $\leq_m$)
4. $A \leq_m B \iff \overline{A} \leq_m \overline{B}$

Proof.

- First two: We just discussed this
- Second two: Easy exercise

□

Reductions: Example

Example (The Problems)

- Need to introduce two problems: REACH and REG-EMPTY
- First Problem: The *reachability problem*:

$$\text{REACH} := \{(G, u, v) \mid \text{there is a path from } u \text{ to } v \text{ in } G\}$$

  - G is a finite directed graph; u, v are nodes in G
  - Question: "Is v reachable from u?"
  - Easily solvable using standard breath first search: REACH $\in$ REC
- Second Problem: *Emptiness problem for regular languages*

$$\text{REG-EMPTY} := \{\langle D \rangle \mid L(D) = \emptyset\}$$

  - D encodes a Deterministic Finite Automaton
  - Question: "Is the language D accepts empty?"

Reductions: Example (Cont.)

Example (The Reduction)

- Will reduce REG-EMPTY to $\overline{\text{REACH}}$
- Idea: Interpret DFA D as a graph
  - Is a final state reachable from initial state?
  - Thus: Start node u is initial state
  - Problem: Want just one target node v, but many final states possible
  - Solution: Additional node v with edges from final states
- Result: f with $\langle D \rangle \mapsto (G, u, v)$
- $L(D)$ empty $\iff$ u can not reach v
- Thus: REG-EMPTY $\leq_m \overline{\text{REACH}}$
- Remark: Implies REG-EMPTY $\in$ REC
  (Closedness of REC under complement!)

A Second Example: Halting Problem with empty input

Lemma

The Halting Problem with empty input is undecidable, i.e.:

$$H_\varepsilon := \{\langle M \rangle \mid M(\varepsilon) \neq \nearrow\} \notin \text{REC}$$

Proof.

- Already know: $H \notin \text{REC}$
- Sufficient to find a reduction $H \leq_m H_\varepsilon$ (Closedness!)
- Given is $(\langle M \rangle, w)$: A machine M with input w
- Idea: Encode w into the states
- Construct a new machine M':
  1. Ignore input and write w on tape (is encoded in states of M')
  2. Simulate M
- $f : (\langle M \rangle, w) \mapsto \langle M' \rangle$ is computable: Simple syntactical manipulations!
- Reduction property by construction:
  - If $(\langle M \rangle, w) \in H$, then M' terminates with all inputs (also with empty input)
  - If $(\langle M \rangle, w) \notin H$, then M' doesn't ever terminate (also not with empty input)

□

Rice's Theorem: Introduction

- We know now: Halting is undecidable for Turing machines
- Even for just empty input!
- Are other properties undecidable?
- (Maybe halting is just a strange property..)
- Will see now: No "non-trivial" behavioural property is decidable!
  - For Turing machines
  - Simpler models behave better (DFA..)
  - Non-trivial: Some Turing machines have it, some don't
- High practical relevance:
  - Either have to restrict model (less expressive)
  - Or only approximate answers (less precise)
- Formally: *Rice's Theorem*

Rice's Theorem: Formal formulation

Theorem (Rice's Theorem)

Let $\mathcal{C}$ be a non-trivial class of semi-decidable languages, i.e., $\emptyset \subsetneq \mathcal{C} \subsetneq \mathrm{RE}$. Then the following $L_{\mathcal{C}}$ is undecidable:

$$L_{\mathcal{C}} := \{\langle M \rangle \mid L(M) \in \mathcal{C}\}$$

Proof (Overview).

- First assume $\emptyset \notin \mathcal{C}$
- Then there must be a non-empty $A \in \mathcal{C}$ (since $\mathcal{C}$ is non-empty)
- We will reduce H to $L_{\mathcal{C}}$
- Idea:
  - We are given M with input w
  - Simulate $M(w)$
  - If it halts, we will semi-decide A
  - If it doesn't halt, we will semi-decide $\emptyset$ (never accept)
  - This is the reduction!

Rice's Theorem: Formal formulation (Cont.)

Theorem (Rice's Theorem)

Let $\mathcal{C}$ be a non-trivial class of semi-decidable languages, i.e., $\emptyset \subsetneq \mathcal{C} \subsetneq \mathrm{RE}$. Then the following $L_{\mathcal{C}}$ is undecidable:

$$L_{\mathcal{C}} := \{\langle M \rangle \mid L(M) \in \mathcal{C}\}$$

Proof (Details).

- Recall: $\emptyset \notin \mathcal{C}$, $A \in \mathcal{C}$, let M_A be machine for A
- Construct a new machine M':
  1. Input y, first simulate $M(w)$ on second tape
  2. If $M(w)$ halts, simulate $M_A(y)$
- Reduction property by construction:
  - If $(\langle M \rangle, w) \in H$, then $L(M') = A$, thus $\langle M' \rangle \in L_{\mathcal{C}}$
  - If $(\langle M \rangle, w) \notin H$, then $L(M') = \emptyset$, thus $\langle M' \rangle \notin L_{\mathcal{C}}$
- What about the case $\emptyset \in \mathcal{C}$? Similar construction showing $H \leq_m \overline{L_{\mathcal{C}}}$

Rice's Theorem: Examples

Example

- The following language is *undecidable*:

$$L := \{\langle M \rangle \mid L(M) \text{ contains at most 5 words}\}$$

- Follows from Rice's Theorem since $\mathcal{C} \neq \emptyset$ and $\mathcal{C} \neq \text{RE}$
- Thus: For any k, can't decide if an M only accepts at most k inputs

Example

- The following language is *decidable*:

$$L := \{\langle M \rangle \mid M \text{ contains at most 5 states}\}$$

- Easy check by looking at encoding of M
- Not a *behavioural* property

Summary Computability Theory

- Defined a model of computation: *Turing machines*
- Explored properties:
  - Decidability and Undecidability
  - Semi-Decidability
  - Example: The Halting problem is undecidable
- Reductions as a *relative* concept
- Closedness allows using them for *absolute* results
- Rice's Theorem:
  All non-trivial behavioural properties of TM are undecidable.

Course Outline

Restricted Resources

- Previous Chapter: *Computability Theory*
  - "What can algorithms do?"
- Now: *Complexity Theory*
  - "What can algorithms do *with restricted resources*?"
  - Resources: Runtime and memory
- Assume the machines always halt in q_{yes} or q_{no}
  - But after how many steps?
  - How many tape positions were necessary?

Landau Symbols

- Resource bounds will depend on *input size*
- Described by functions $f : \mathbb{N} \to \mathbb{N}$
- Need ability to express "grows in the order of"
  - Consider $f_1(n) = n^2$ and $f_2(n) = 5 \cdot n^2 + 3$
  - Eventually, n^2 dominates for large n
  - Both express "quadratic growth"
  - Want to see all $c_1 \cdot n^2 + c_2$ equivalent
  - *Asymptotic behaviour*
- Formal notation for this: $\mathcal{O}(n^2)$
- Will provide a kind of *upper bound of asymptotic growth*

Landau Symbols: Definition

Definition

Let $g : \mathbb{N} \to \mathbb{N}$. $\mathcal{O}(g)$ denotes the set of all functions $f : \mathbb{N} \to \mathbb{N}$ such that there are n_0 and c with

$$\forall n \geq n_0 : f(n) \leq c \cdot g(n).$$

We also just write $f(n) = \mathcal{O}(g(n))$.

Lemma (Alternative characterization)

For $f, g : \mathbb{N} \to \mathbb{N}_{>0}$ the following holds:

$$f \in \mathcal{O}(g) \iff \exists c > 0 : \limsup_{n \to \infty} \frac{f(n)}{g(n)} \leq c$$

(Without proof.)

Landau Symbols: Examples

- We have $5 \cdot n^2 + 3 = \mathcal{O}(n^2)$
- One even writes $\mathcal{O}(n) = \mathcal{O}(n^2)$ (meaning "$\subseteq$")
- Both is abuse of notation! Not symmetric: $\mathcal{O}(n^2) \neq \mathcal{O}(n)$!

Examples

- $n \cdot \log(n) = \mathcal{O}(n^2)$
- $n^c = \mathcal{O}(2^n)$ for all constants c
- $\mathcal{O}(1)$ are the bounded functions
- $n^{\mathcal{O}(1)}$ are the functions bounded by a polynomial

- Other symbols exist for lower bounds (Ω), strict bounds (o, ω) and "grows equally" (Θ)

Proper complexity functions

- Landau-Symbols classify functions according to growth
- Which functions to consider for resource bounds?
- Only "proper" ones:

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function.

1. f is *time-constructible* if there exists a TM which on input 1^n stops after $\mathcal{O}(n + f(n))$ steps.
2. f is *space-constructible* if there exists a TM which on input 1^n outputs $1^{f(n)}$ and does not use more than $\mathcal{O}(f(n))$ space.

- This allows us to assume "stopwatches"
- All common "natural" functions have these properties

Resource measures

Definition

1. The *runtime* $\text{time}_M(w)$ of a TM M with input w is defined as:

$$\text{time}_M(w) := \max\{t \geq 0 \mid \exists y, z \in \Gamma^*, q \in F : (w, q_0, \varepsilon) \vdash^t (y, q, z)\}$$

2. If, for all inputs w and a $t : \mathbb{N} \to \mathbb{N}$ it holds that $\text{time}_M(w) \leq t(|w|)$, then M is *$t(n)$-time-bounded*. Further:

$$\text{DTIME}(t(n)) := \{L(M) \mid M \text{ is } t(n)\text{-time-bounded}\}$$

3. The *required space* $\text{space}_M(w)$ of a TM M with input w is defined as:

$$\text{space}_M(w) := \max\{n \geq 0 \mid \ M \text{ uses } n \text{ squares on a working tape}\}$$

4. If for all inputs w and an $s : \mathbb{N} \to \mathbb{N}$ it holds that $\text{space}_M(w) \leq s(|w|)$, then M is *$s(n)$-space-bounded*. Further:

$$\text{DSPACE}(s(n)) := \{L(M) \mid M \text{ is } s(n)\text{-space-bounded}\}$$

Resource measures (Cont.)

Definition

1. For functions, we have:

$$\text{FTIME}(t(n)) := \{f \mid \exists M \text{ being } t(n)\text{-time-bounded and computing } f\}$$

2. For *non-deterministic* M, time and space are as above, and we have:

$$\text{NTIME}(t(n)) := \{L(M) \mid M \text{ is non-det. and } t(n)\text{-time-bounded}\}$$
$$\text{NSPACE}(s(n)) := \{L(M) \mid M \text{ is non-det. and } s(n)\text{-space-bounded}\}$$

- Recall: *Non-deterministic* machines can choose different next steps
  - Can be imagined as a *computation tree*
  - Time and space bounds for *all paths* in the tree
- Note: $\text{space}_M(w)$ is for the *working tapes*
  - Only they "consume memory" during the computation
  - Input (read-only) and output (write-only) should not count
  - Allows notion of *sub-linear space*, e.g., $\log(|w|)$

Common Complexity Classes

- Deterministic *time complexity* classes:
  - Linear time:

$$\text{LINTIME} := \bigcup_{c \geq 1} \text{DTIME}(cn + c) = \text{DTIME}(\mathcal{O}(n))$$

  - Polynomial time:

$$\text{P} := \bigcup_{c \geq 1} \text{DTIME}(n^c + c) = \text{DTIME}(n^{\mathcal{O}(1)})$$

  - Polynomial time functions:

$$\text{FP} := \bigcup_{c \geq 1} \text{FTIME}(n^c + c) = \text{FTIME}(n^{\mathcal{O}(1)})$$

  - Exponential time

$$\text{EXP} := \bigcup_{c \geq 1} \text{DTIME}(2^{n^c + c}) = \text{DTIME}\left(2^{n^{\mathcal{O}(1)}}\right)$$

Common Complexity Classes (Cont.)

- Deterministic *space complexity* classes:
  - Logarithmic space:
    $$\mathsf{L} := \mathrm{DSPACE}(\mathcal{O}(\log(n)))$$
  - Polynomial space:
    $$\mathrm{PSPACE} := \mathrm{DSPACE}(n^{\mathcal{O}(1)})$$
  - Exponential space:
    $$\mathrm{EXPSPACE} := \mathrm{DSPACE}\left(2^{n^{\mathcal{O}(1)}}\right)$$
- Non-deterministic classes defined similarly:

  NLINTIME, NP, NEXP, NL, NPSPACE and NEXPSPACE

Common Complexity Classes: Example

Example (REACH)

- Consider again the *reachability problem*:

$$\text{REACH} := \{(G, u, v) \mid \text{there is a path from } u \text{ to } v \text{ in } G\}$$

- Decidable – but how much space is needed?
- *Non-deterministically*: $\text{REACH} \in \text{NL}$
  - Explore graph beginning with u
  - Choose next node non-deterministically, for at most n steps
  - If there is a path to v, it can be found that way
  - Space: For step counter and number of current node: $\mathcal{O}(\log(n))$
- *Deterministically*: $\text{REACH} \in \text{DSPACE}(\mathcal{O}(\log(n)^2))$
  - Sophisticated recursive algorithm
  - Split path p of length $\leq n$:
    - $p = p_1 p_2$ with p_1, p_2 of length $\leq n/2$
    - Iterate over all intermediate nodes
  - Space: Recursion stack depth $\log(n)$ and elements $\log(n)$: $\mathcal{O}(\log(n)^2)$

Complexity Class Relations

- Clear from definitions:

$$\text{LINTIME} \subseteq \text{P} \subseteq \text{EXP}$$

- Same relation for non-deterministic classes:

$$\text{NLINTIME} \subseteq \text{NP} \subseteq \text{NEXP}$$

- Only *inclusion*, no *separation* yet:
  - Know that $\text{LINTIME} \subseteq \text{P}$
  - But is there $L \in \text{P} - \text{LINTIME}$?
  - Such an L would *separate* LINTIME and P
- Will now see a very "fine-grained" separation result

Hierarchy Theorem

Theorem (Hierarchy Theorem)

- *Let* $f : \mathbb{N} \to \mathbb{N}$ *be time-constructible and* $g : \mathbb{N} \to \mathbb{N}$ *with*

$$\liminf_{n\to\infty} \frac{g(n) \cdot \log(g(n))}{f(n)} = 0.$$

Then there exists $L \in \mathrm{DTIME}(f(n)) - \mathrm{DTIME}(g(n))$.

- *Let* $f : \mathbb{N} \to \mathbb{N}$ *be space-constructible and* $g : \mathbb{N} \to \mathbb{N}$ *with*

$$\liminf_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

Then there exists $L \in \mathrm{DSPACE}(f(n)) - \mathrm{DSPACE}(g(n))$.

(Without proof.)

Hierarchy Theorem: Examples

Example

- Let $\mathcal{C}_k := \text{DTIME}(\mathcal{O}(n^k))$
- Using time hierarchy theorem:

$$\mathcal{C}_1 \subsetneq \mathcal{C}_2 \subsetneq \mathcal{C}_3 \subsetneq \ldots \qquad \text{(Infinite hierarchy)}$$

- Means: Let $p(n)$ and $q(n)$ be polynomials, $\deg p < \deg q$
- Then there is L such that:
  - Decidable in $\mathcal{O}(q(n))$ time
  - *Not* decidable in $\mathcal{O}(p(n))$ time

- Remark: Theorem states "more time means more power"
- Also the case with REC $\subsetneq$ RE:
  - REC: Time bounded: Always halt
  - RE: May not halt, "infinite time"

Determinism vs. Non-determinism

Theorem

For each space-constructible function $f : \mathbb{N} \to \mathbb{N}$, the following holds:

$$\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f)$$

Proof (Overview).

- First and third clear: Determinism is special case
- Now show $\mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$
- Time bounded by $f(n)$ implies space bounded by $f(n)$
- Still need to remove non-determinism
- Key idea:
  - Time bound $f(n)$: At most $f(n)$ non-deterministic choices
  - Computation tree at most $f(n)$ deep
  - Represent paths by strings of size $f(n)$
  - Simulate all paths by enumerating the strings

Determinism vs. Non-determinism (Cont.)

Theorem

For each space-constructible function $f : \mathbb{N} \to \mathbb{N}$, the following holds:

$$\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f)$$

Proof (Details).

- Want to show $\mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$
- Let $L \in \mathrm{NTIME}(f)$ and N corresponding machine
- Let d be maximal degree of non-determinism
- Build new machine M:
  1. Systematically generate words $c \in \{1, \ldots, d\}^{f(n)}$
  2. Simulate N with non-deterministic choices c
  3. Repeat until all words generated (overwrite c each time)
- Simulation is deterministic and needs only $\mathcal{O}(f(n))$ space
  - (But takes exponentially long!)

Deterministic vs. Non-deterministic Space

- Theorem implies:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{NPSPACE}$$

- Thus, in context of polynomial bounds:
  - Non-determinism "beats" determinism
  - Space "beats" time
- But are these inclusions *strict*?
- Will now see: $\mathrm{PSPACE} = \mathrm{NPSPACE}$
- Recall: $\mathrm{REACH} \in \mathrm{DSPACE}(\mathcal{O}(\log(n)^2))$

Deterministic vs. Non-deterministic Space (Cont.)

Theorem (Savitch)

For each space-constructible function $f : \mathbb{N} \to \mathbb{N}$, the following holds:

$$\mathrm{NSPACE}(f) \subseteq \mathrm{DSPACE}(f^2)$$

Proof (Sketch).

- Let $L \in \mathrm{NSPACE}(f)$ and M_L corresponding non-deterministic TM
- Consider *configuration graph* of M_L for an input w
  - Each node is a configuration
  - Edges are given by step relation $\vdash$
  - M_L space bounded, thus only $c^{f(|w|)}$ configurations
- Assume just one final accepting configuration
- Question: "Is there a path from initial to final configuration?"
- *Reachability problem!*
- Solve it with $\mathcal{O}\left(\log\left(c^{f(n)}\right)^2\right) = \mathcal{O}(f(n)^2)$ space

□

Polynomial Complexity Classes

Corollary

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$$

- Previous theorem implies NPSPACE $\subseteq$ PSPACE
- First two inclusions: Difficult, *next chapter!*
- Following concept will be of use:

Definition

Let $\mathcal{C} \subseteq \mathcal{P}(\Sigma^*)$ be a class of languages. We define:

$$\text{co-}\mathcal{C} := \{\overline{L} \mid L \in \mathcal{C}\}$$

- For deterministic $\mathcal{C} \subseteq$ REC: $\mathcal{C} = \text{co-}\mathcal{C}$

Complementary Classes: Asymmetries

- Consider RE and co-RE:
  - For RE the TM always halts on the *positive inputs*
    - ★ "For $x \in L$ there is a finite path to q_{yes}"
  - For co-RE it always halts on the *negative inputs*
    - ★ "For $x \notin L$ there is a finite path to q_{no}"
  - RE $\neq$ co-RE (Halting Problem, ..)
  - REC = co-REC and REC = RE $\cap$ co-RE
- Consider NPSPACE and co-NPSPACE:
  - We know PSPACE = NPSPACE and PSPACE = co-PSPACE
  - Thus NPSPACE = co-NPSPACE
- What about P, NP and co-NP?
  - Looks like RE situation:
    - ★ NP: "For $x \in L$ there is a *bounded* path to q_{yes}"
    - ★ co-NP: "For $x \notin L$ there is a *bounded* path to q_{no}"
  - Surprisingly: Relationship not known!

Course Outline

0. Introduction
1. Basic Computability Theory
   - Formal Languages
   - Model of Computation: Turing Machines
   - Decidability, Undecidability, Semi-Decidability
2. Complexity Classes
   - Landau Symbols: The $\mathcal{O}(\cdot)$ Notation
   - Time and Space Complexity
   - Relations between Complexity Classes
3. Feasible Computations: P vs. NP
   - Proving vs. Verifying
   - Reductions, Hardness, Completeness
   - Natural NP-complete problems
4. Advanced Complexity Concepts
   - Non-uniform Complexity
   - Probabilistic Complexity Classes
   - Interactive Proof Systems

Course Outline

Feasible Computations

- Will now focus on classes P and NP
- *Polynomial time bounds* as "feasible", "tractable", "efficient"
  - Polynomials grow only "moderately"
  - Many practical problems polynomial
  - Often with small degrees (n^2 or n^3)

Recall P and NP

- Introduced P and NP via Turing machines:
  - Polynomial time bounds
  - Deterministic vs. non-deterministic operation
- Recall P: For $L_1 \in \mathrm{P}$
  - Existence of a *deterministic* TM M
  - Existence of a polynomial $p_M(n)$
  - For each input $x \in \Sigma^*$ runtime $\leq p_M(|x|)$
- Recall NP: For $L_2 \in \mathrm{NP}$
  - Existence of a *non-deterministic* TM N
  - Existence of a polynomial $p_N(n)$
  - For each input $x \in \Sigma^*$ runtime $\leq p_N(|x|)$
  - For *all* computation paths
- Theoretical model – practical significance?
- Introduce now a *new characterization* of NP

A new NP characterization

Definition

Let $R \in \Sigma^* \times \Sigma^*$ (binary relation). R is *polynomially bounded*, if there exists a polynomial $p(n)$, such that:

$$\forall (x, y) \in R : |y| \leq p(|x|)$$

Lemma

NP *is the class of all* L *such that there exists a polynomially bounded* $R_L \in \Sigma^* \times \Sigma^*$ *satisfying:*

- $R_L \in \mathrm{P}$, *and*
- $x \in L \iff \exists w : (x, w) \in R_L$.

We call w *a witness (or proof) for* $x \in L$ *and* R_L *the witness relation.*

Proving vs. Verifying

- For $L \in$ P:
  - Machine must decide membership of x in polynomial time
  - Interpret as "*finding a proof*" for $x \in L$
- For $L \in$ NP: (new characterization)
  - Machine is provided a witness w
  - Interpret as "*verifying the proof*" for $x \in L$
- *Efficient* proving and verifying procedures:
  - For P, runtime is bounded
  - For NP, also witness size is bounded
- Write $L \in$ NP as:

$$L = \{x \in \Sigma^* \mid \exists w \in \Sigma^* : (x, w) \in R_L\}$$

Proving vs. Verifying (Cont.)

- P-problems: solutions can be *efficiently found*
- NP-problems: solutions can be *efficiently checked*
- Checking certainly a prerequisite for finding (thus $P \subseteq NP$)
- But is finding *more* difficult?
  - Intuition says: "Yes!"
  - Theory says: "We don't know." (yet?)
- Formal formulation:

$$P \stackrel{?}{=} NP$$

- One of the most important questions of computer science!
  - Many proofs for either "$=$" or "$\neq$"
  - None correct so far
  - Clay Mathematics Institute offers $1.000.000 prize

A new NP characterization (Cont.)

Lemma (Revisited)

NP *is the class of all* L *such that there exists a polynomially bounded* $R_L \in \Sigma^* \times \Sigma^*$ *satisfying:*

- $R_L \in \mathrm{P}$, *and*
- $x \in L \iff \exists w : (x, w) \in R_L$. *($w$ is a witness)*

Proof (First part).

- First, let $L \in \mathrm{NP}$. Let N be the machine with $p_N(n)$ time bound.
- Want to show: R_L as above exists
- Idea:
  - On input x, all computations do $\leq p_N(|x|)$ steps
  - $x \in L$ iff an *accepting* computation exists
  - Encode computation (non-deterministic choices) into w
  - All such pairs (x, w) define R_L
- R_L has all above properties

A new NP characterization (Cont. 2)

Lemma (Revisited)

NP *is the class of all* L *such that there exists a polynomially bounded* $R_L \in \Sigma^* \times \Sigma^*$ *satisfying:*

- $R_L \in \mathrm{P}$, *and*
- $x \in L \iff \exists w : (x, w) \in R_L$. *($w$ is a witness)*

Proof (Second part).

- Now, let L as above, using R_L bounded by $p(n)$
- Want to show: Non-deterministic N exists, polynomially bounded
- Idea to construct N:
  - R_L bounds length of w by $p(|x|)$
  - $R_L \in P$: There is a M for checking R_L
  - N can "guess" w first
  - Then simulate M for checking $(x, w) \in R_L$
  - Accepting path exists iff $\exists w : (x, w) \in R_L$
- N is polynomially time bounded

□

A new co-NP characterization

Remark

- Recall: All $L \in$ NP can now be written as:

$$L = \{x \in \Sigma^* \mid \exists w \in \Sigma^* : (x, w) \in R_L\}$$

- Read this as:
  - *Witness relation* R_L
  - For each *positive* instance, there is a proof w
  - For no negative instance, there is a proof w
  - The proof is efficiently checkable
- Similar characterization for all $L' \in$ co-NP:

$$L' = \{x \in \Sigma^* \mid \forall w \in \Sigma^* : (x, w) \notin R_{L'}\}$$

- Read this as:
  - *Disproof relation* $R_{L'}$
  - For each *negative* instance, there is a *disproof* w
  - For no positive instance, there is a disproof w
  - The disproof is efficiently checkable

Boolean Formulas

Definition

Let $X = \{x_1, \ldots, x_N\}$ be a set of variable names.

- Define *boolean formulas* BOOL inductively:
  - $\forall i : x_i \in \text{BOOL}$.
  - $\varphi_1, \varphi_2 \in \text{BOOL} \implies (\varphi_1 \wedge \varphi_2), (\neg\varphi_1) \in \text{BOOL}$ (*conjunction* and *negation*)
- A *truth assignment* for the variables in X is a word $\underbrace{\alpha_1 \ldots \alpha_N}_{\alpha} \in \{0,1\}^N$.
- The *value* $\varphi(\alpha)$ of φ under α is defined inductively:

φ :	x_i	$\neg\psi$	$\psi_1 \wedge \psi_2$
$\varphi(\alpha)$:	α_i	$1 - \psi(\alpha)$	$\psi_1(\alpha) \cdot \psi_2(\alpha)$

- Shorthand notations:
  - $\varphi_1 \vee \varphi_2$ (disjunction) for $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$,
  - $\varphi_1 \rightarrow \varphi_2$ (implication) for $\neg\varphi_1 \vee \varphi_2$
  - $\varphi_1 \leftrightarrow \varphi_2$ (equivalence) for $(\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

Example: XOR Function

Example

- Consider the *exclusive or* XOR with m arguments:

$$\mathrm{XOR}(z_1, \ldots, z_m) := \bigvee_{i=1}^{m} z_i \wedge \bigwedge_{1 \le i < j \le m} \neg(z_i \wedge z_j)$$

- $\mathrm{XOR}(z_1, \ldots, z_m) = 1 \iff z_j = 1$ for exactly one j
- Can also be used as shorthand notation.

Example for NP: The Satisfiability Problem

Example

- Consider $\psi_1 = (x_1 \vee \neg x_2) \wedge x_3$ and $\psi_2 = (x_1 \wedge \neg x_1)$:
  - $\psi_1(\alpha) = 1$ for $\alpha = 011$
  - $\psi_2(\alpha) = 0$ for all α
- $\varphi \in$ BOOL is called *satisfiable*, if $\exists \alpha : \varphi(\alpha) = 1$
- Can encode boolean formula into words over fixed alphabet Σ
- Language of all satisfiable formulas, the *satisfiability problem*:

$$\text{SAT} := \{\langle\varphi\rangle \mid \varphi \in \text{BOOL} \text{ is satisfiable}\}$$

- Obviously, SAT $\in$ NP:
  - Witness for positive instance $\langle\varphi\rangle$ is α with $\varphi(\alpha) = 1$
  - Size of witness: linearly bounded in $|\langle\varphi\rangle|$
  - Validity check efficient
- Unknown, whether SAT $\in$ P!

Bounded Reductions

- Comparing P and NP by directly comparing problems
- Assume $A, B \in \mathrm{NP}$ and $C \in \mathrm{P}$
  - How do A and B relate?
  - Is C "easier" than A and B?
  - Maybe we just didn't find good algorithms for A or B?
- Recall: *Reductions*
  - Given problems A and B
  - Solve A by reducing it to B and solving B
  - Tool for that: Reduction function f
  - Consequence: A is "easier" than B
- Used *many-one reductions* in unbounded setting
- Now: *Bounded setting*, so f should be also bounded!
  - Introduce "Cook reductions"

Polynomial Reduction (Cook Reduction)

Definition

$A \subseteq \Sigma^*$ is *polynomially reducible* to $B \subseteq \Sigma^*$ (written $A \leq_m^p B$), if there $f \in \mathrm{FP}$, such that

$$\forall w \in \Sigma^* : w \in A \iff f(w) \in B$$

Lemma

For all A, B and C the following hold:

1. $A \leq_m^p B \wedge B \in \mathrm{P} \Longrightarrow A \in \mathrm{P}$ (*Closedness of* P *under* $\leq_m^p$)
2. $A \leq_m^p B \wedge B \in \mathrm{NP} \Longrightarrow A \in \mathrm{NP}$ (*Closedness of* NP *under* $\leq_m^p$)
3. $A \leq_m^p B \wedge B \leq_m^p C \Longrightarrow A \leq_m^p C$ (*Transitivity of* $\leq_m^p$)
4. $A \leq_m^p B \iff \overline{A} \leq_m^p \overline{B}$

Hardness, Completeness

- Can *compare* problems now
- Introduce now "hard" problems for a class $\mathcal{C}$:
  - Can solve whole $\mathcal{C}$ if just one of them
  - Are more difficult then everything in $\mathcal{C}$

Definition

- A is called *$\mathcal{C}$-hard*, if: $\forall L \in \mathcal{C} : L \leq_m^p A$
- If A is $\mathcal{C}$-hard and $A \in \mathcal{C}$, then A is called *$\mathcal{C}$-complete*
- NPC is the class of all NP-*complete* languages

- NPC: "Most difficult" problems in NP
- Solve one of them, solve whole NP
- Solve one of them *efficiently*, solve whole NP *efficiently*

Hardness, Completeness: Properties

Lemma

1. *A is $\mathcal{C}$-complete if and only if $\overline{A}$ is co-$\mathcal{C}$-complete.*
2. $\mathrm{P} \cap \mathrm{NPC} \neq \emptyset \Longrightarrow \mathrm{P} = \mathrm{NP}$
3. $A \in \mathrm{NPC} \wedge A \leq_m^p B \wedge B \in \mathrm{NP} \Longrightarrow B \in \mathrm{NPC}$

Proof (First part).

- Let A be $\mathcal{C}$-complete, and $L \in$ co-$\mathcal{C}$
- Want to show: $L \leq_m^p \overline{A}$
- Indeed: $L \in$ co-$\mathcal{C} \iff \overline{L} \in \mathcal{C} \implies \overline{L} \leq_m^p A \iff L \leq_m^p \overline{A}$
- Other direction similar (symmetry)

Hardness, Completeness: Properties (Cont.)

Lemma

1. *A is $\mathcal{C}$-complete if and only if $\overline{A}$ is co-$\mathcal{C}$-complete.*
2. $\mathrm{P} \cap \mathrm{NPC} \neq \emptyset \Longrightarrow \mathrm{P} = \mathrm{NP}$
3. $A \in \mathrm{NPC} \wedge A \leq_m^p B \wedge \ B \in \mathrm{NP} \Longrightarrow B \in \mathrm{NPC}$

Proof (Second part).

- Assume $A \in \mathrm{P} \cap \mathrm{NPC}$ and let $L \in \mathrm{NP}$
- Want to show: $L \in \mathrm{P}$ (since then $\mathrm{P} = \mathrm{NP}$)
- $L \in \mathrm{NP} \implies L \leq_m^p A$ since $A \in \mathrm{NPC}$
- $L \leq_m^p A \implies L \in \mathrm{P}$ since $A \in \mathrm{P}$

Hardness, Completeness: Properties (Cont. 2)

Lemma

1. *A is $\mathcal{C}$-complete if and only if $\overline{A}$ is* co-$\mathcal{C}$*-complete.*
2. $\mathrm{P} \cap \mathrm{NPC} \neq \emptyset \Longrightarrow \mathrm{P} = \mathrm{NP}$
3. $A \in \mathrm{NPC} \wedge A \leq_m^p B \wedge \ B \in \mathrm{NP} \Longrightarrow B \in \mathrm{NPC}$

Proof (Third part).

- Assume $A \in \mathrm{NPC}$, $B \in \mathrm{NP}$, $A \leq_m^p B$ and $L \in \mathrm{NP}$
- Want to show: $L \leq_m^p B$ (since then, B is NP-complete)
- $L \in \mathrm{NP} \implies L \leq_m^p A$ since $A \in \mathrm{NPC}$
- $L \leq_m^p A \implies L \leq_m^p B$ since $A \leq_m^p B$ (transitivity!)

A first NP-complete Problem

- Do NP-complete problems actually exist? Indeed:

Lemma

The following language is NP*-complete:*

$$\text{NPCOMP} := \{(\langle M \rangle, x, 1^n) \mid M \text{ is NTM and accepts } x \text{ after} \leq n \text{ steps}\}$$

("NTM" means "non-deterministic Turing machine".)

- How to prove a problem A is NP-complete? 2 parts:
  1. Membership: *Show* $A \in$ NP
     (Directly or via $A \leq_m^p B$ for a $B \in$ NP)
  2. Hardness: *Show* $L \leq_m^p A$ *for all* $L \in$ NP
     (Directly or via $C \leq_m^p A$ for a C which is NP-hard)

A first NP-complete Problem (Cont.)

Lemma

The following language is NP*-complete:*

$$\text{NPCOMP} := \{(\langle M \rangle, x, 1^n) \mid M \text{ is NTM and accepts } x \text{ after} \leq n \text{ steps}\}$$

Proof (First part).

- Want to show: NPCOMP $\in$ NP
- Given $(\langle M \rangle, x, 1^n)$
- If M accepts x in $\leq n$ steps, then at most n non-deterministic choices
- For each x, these choices are witness w!
  - Exactly the positive instances x have one w
  - $|w|$ is bounded by n
  - Efficient check by simulating that path
- All (x, w) are witness relation R_L, so NPCOMP $\in$ NP

A first NP-complete Problem (Cont. 2)

Lemma

The following language is NP*-complete:*

$$\text{NPCOMP} := \{(\langle M \rangle, x, 1^n) \mid M \text{ is NTM and accepts } x \text{ after} \leq n \text{ steps}\}$$

Proof (Second part).

- Want to show now: NPCOMP is NP-hard
- Let $L \in \text{NP}$, decided by M_L, bound $p(n)$
- Show $L \leq_m^p \text{NPCOMP}$ with reduction function:

$$f : x \mapsto (\langle M_L \rangle, x, 1^{p(|x|)})$$

  - $f \in \text{FP}$
  - If $x \in L$, then M_L accepts x within $p(|x|)$ steps
  - If $x \notin L$, then M_L never accepts x
  - Thus: $x \in L \iff f(x) \in \text{NPCOMP}$

□

NP-completeness of SAT

- Know now: There is an NP-complete set
- Practical relevance?
- Are there "natural" NP-complete problems?
- Recall the *satisfiability problem*:

$$\text{SAT} := \{\langle\varphi\rangle \mid \varphi \in \text{BOOL} \text{ is satisfiable}\}$$

- We saw that SAT $\in$ NP:
  - A satisfying truth assignment α is witness
- Even more, it's one of the most difficult NP-problems:

Theorem (Cook, Levin)

SAT *is* NP*-complete.*

NP-completeness of SAT: Proof ideas

- We will show NPCOMP $\leq_m^p$ SAT
- Need *reduction function* $f \in$ FP such that:
  - *Input* $(\langle M \rangle, x, 1^n)$: Machine M, word x, runtime bound n
  - *Output* ψ: Boolean formula such that

$$(\langle M \rangle, x, 1^n) \in \text{NPCOMP} \iff \psi \in \text{SAT}\,.$$

- Assume M has just one tape
- If M accepts x, then *within n steps*
- *Only $2n+1$ tape positions reached*!
- Central idea:
  - Imagine a configuration as a line, $\mathcal{O}(n)$ symbols
  - Whole computation as a *matrix* with n lines
  - Encode matrix into formula ψ
  - *ψ satisfiable iff computation reaches q_{yes}*
  - Formula size = Matrix size = $\mathcal{O}(n^2)$

NP-completeness of SAT: Proof ideas (Cont.)

- Note: M is *non-deterministic*
  - Different computations possible for each x
  - Different paths in computation tree
- Matrix represents one path to q_{yes}
- If $x \in L(M)$ then there is *at least one path* to q_{yes}
  - Each path described by one matrix
  - Thus, at least one matrix!
- If $x \notin L(M)$ then there *no path* to q_{yes}
  - Thus, there is no matrix!
- Formula ψ describes a matrix which
  - Represents a computation path
  - Of length at most n
  - To q_{yes}
- Thus: ψ satisfiable iff accepting computation path exists!

NP-completeness of SAT: Proof details

- Describe now the formula ψ
- Given is M, states $Q = \{q_0, \ldots, q_k\}$, tape alphabet $\Gamma = \{a_1, \ldots, a_l\}$
- Final state $q_{yes} \in Q$
- Used boolean variables:
  - $Q_{t,q}$ for all $t \in [0, n]$ and $q \in Q$.
    Interpretation: After step t, the machine is in state q.
  - $H_{t,i}$ for all $t \in [0, n]$ and $i \in [-n, n]$.
    Interpretation: After step t, the tape head is at position i.
  - $T_{t,i,a}$ for all $t \in [0, n]$, $i \in [-n, n]$ and $a \in \Gamma$.
    Interpretation: After step t, the tape contains symbol a at position i.
- Number of variables: $\mathcal{O}(n^2)$
- Structure of ψ:

$$\psi := \mathit{Conf} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{End}$$

$\psi := \mathit{Conf} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{End}$

- Part *Conf* of ψ:
  - Ensures: *Satisfying* truth assignments describe *valid computations*
- Again, 3 parts:

$$\mathit{Conf} := \mathit{Conf}_Q \wedge \mathit{Conf}_H \wedge \mathit{Conf}_T$$

$$\mathit{Conf}_Q := \bigwedge_{t=0}^{n} \mathrm{XOR}(Q_{t,q_0}, \ldots, Q_{t,q_k})$$

$$\mathit{Conf}_H := \bigwedge_{t=0}^{n} \mathrm{XOR}(H_{t,-n}, \ldots, H_{t,n})$$

$$\mathit{Conf}_T := \bigwedge_{t=0}^{n} \bigwedge_{i=-n}^{n} \mathrm{XOR}(T_{t,i,a_1}, \ldots, T_{t,i,a_l})$$

$\psi := \mathit{Conf} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{End}$

- Part *Start* of ψ:
  - Ensures: *At* $t = 0$, machine is in *start configuration*
- One single formula:

$$\mathit{Start} := Q_{0,q_0} \wedge H_{0,0} \wedge \bigwedge_{i=-n}^{-1} T_{0,i,\square} \wedge \bigwedge_{i=0}^{|x|-1} T_{0,i,x_{i+1}} \wedge \bigwedge_{i=|x|}^{n} T_{0,i,\square}$$

$\psi := \mathit{Conf} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{End}$

- Part *Step* of ψ:
  - Ensures: *At each step*, machine executes a *legal action*
  - Only one tape field changed; head moves by one position
  - Consistency with δ

$$\mathit{Step} := \mathit{Step}_1 \wedge \mathit{Step}_2$$

$$\mathit{Step}_1 := \bigwedge_{t=0}^{n-1} \bigwedge_{i=-n}^{n} \bigwedge_{a \in \Gamma} ((\neg H_{t,i} \wedge T_{t,i,a}) \rightarrow T_{t+1,i,a})$$

$$\mathit{Step}_2 := \bigwedge_{t=0}^{n-1} \bigwedge_{i=-n}^{n} \bigwedge_{a \in \Gamma} \bigwedge_{p \in Q} \Big((Q_{t,p} \wedge H_{t,i} \wedge T_{t,i,a}) \rightarrow \bigvee_{(q,b,D) \in \delta(p,a)} (Q_{t+1,q} \wedge H_{t+1,i+D} \wedge T_{t+1,i,b}) \Big)$$

$\psi := \mathit{Conf} \wedge \mathit{Start} \wedge \mathit{Step} \wedge \mathit{End}$

- Part *End* of ψ:
  - Ensures: *Eventually*, machine reaches an *accepting configuration*
- One single formula:

$$\mathit{End} := \bigvee_{t=0}^{n} Q_{t,q_{yes}}$$

- Completes proof:
  - By construction, $\psi \in \mathrm{SAT} \iff (\langle M \rangle, x, 1^n) \in \mathrm{NPCOMP}$
  - Construction is efficient

□

co- NP-completeness of UNSAT

Remark

- SAT is NP-complete
- Consider its complement:

$$\text{UNSAT} := \{\langle\varphi\rangle \mid \varphi \in \text{BOOL} \text{ is } \textit{not} \text{ satisfiable}\} = \overline{\text{SAT}}$$

- Clearly, UNSAT $\in$ co- NP:
  - Disproof for $\langle\varphi\rangle$ is α with $\varphi(\alpha) = 1$
  - Can be checked efficiently, like for SAT
  - Follows from SAT $\in$ NP anyway
- SAT is NP-complete $\iff$ UNSAT is co- NP-complete

- Will now study some more NP-complete problems!

CIRSAT: Satisfiability of Boolean Circuits

Definition (Boolean Circuit)

Let $X = \{x_1, \ldots, x_N\}$ be a set of variable names.

- A *boolean circuit* over X is a sequence $c = (g_1, \ldots, g_m)$ of *gates*:

$$g_i \in \{\bot, \top, x_1, \ldots, x_N, (\neg, j), (\wedge, j, k)\}_{1 \leq j,k < i}$$

- Each g_i *represents a boolean function* $f_c^{(i)}$ with N inputs $\alpha \in \{0,1\}^N$:

$g_i(\alpha)$:	$\bot$	$\top$	x_i	$(\neg, j)$	$(\wedge, j, k)$
$f_c^{(i)}(\alpha)$:	0	1	α_i	$1 - f_c^{(j)}(\alpha)$	$f_c^{(j)}(\alpha) \cdot f_c^{(k)}(a)$

- Use $a \vee b$ as shorthand for $\neg(\neg a \wedge \neg b)$
- Whole circuit c represents boolean function $f_c(\alpha) := f_c^{(m)}(\alpha)$.
- c is *satisfiable* if $\exists \alpha \in \{0,1\}^N$ such that $f_c(\alpha) = 1$.

CIRSAT: Satisfiability of Boolean Circuits (Cont.)

- Practical question: "Is circuit ever 1?"
  - Find unused parts of circuits (like dead code)
- Formally:
  - (Assume again some fixed encoding $\langle c \rangle$ of circuit c)

Definition

The *circuit satisfiability problem* is defined as:

$$\text{CIRSAT} := \{\langle c \rangle \mid c \text{ is a satisfiable circuit}\}$$

CIRSAT: Satisfiability of Boolean Circuits (Cont. 2)

Lemma

CIRSAT *is* NP*-complete.*

Proof.

- CIRSAT $\in$ NP: Satisfying input is witness w
  - Size N for N variables
  - Verifying: Evaluating all gates is efficient
- SAT $\leq_m^p$ CIRSAT: Transform formula φ to circuit c

□

- Remark: Transformation circuit to equivalent formula *not efficient*
  - Circuit can "reuse" intermediate results
  - CIRSAT $\leq_m^p$ SAT anyway (SAT is NP-complete!)
  - Transformation produces *satisfiability equivalent* formula

CNF: Restricted Structure of Boolean Formulas

Definition (CNF)

Let $X = \{x_1, \ldots, x_N\}$ be a set of variable names.

- A *literal* l is either x_i (variable) or $\neg x_i$ (negated variable, also $\overline{x_i}$)
- A *clause* is a disjunction $C = l_1 \vee \ldots \vee l_k$ of literals
- A boolean formula in *conjunctive normal form* (CNF) is a conjunction of clauses $\varphi = C_1 \wedge \ldots \wedge C_m$
- Set of all CNF formulas:

$$\text{CNFBOOL} := \left\{ \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k(i)} \sigma_{i,j} \mid \sigma_{i,j} \text{ are literals} \right\}$$

- CNF formulas where the clauses contain only k literals: *k-CNF*

$$k\text{-SAT} := \{\langle \varphi \rangle \mid \varphi \in k\text{-CNFBOOL is satisfiable}\}$$

k-SAT: NP-complete for $k \geq 3$

Lemma

1. 1-SAT, 2-SAT $\in$ P
2. 3-SAT *is* NP-*complete.*

Proof (Overview).

- First part: Exercise
- Second part:
  - 3-SAT $\in$ NP clear: 3-SAT $\leq_m^p$ SAT (special case)
  - Then show CIRSAT $\leq_m^p$ 3-SAT
  - Given a circuit $c = (g_1, \ldots, g_m)$, construct a 3-CNF formula ψ_c
  - Variables in formula: One for each input and each gate
  - $x_1, \ldots, x_N$ for inputs of circuit
  - $y_1, \ldots, y_m$ for gates
  - Clauses (size 3) enforce values of gates

NP-completeness of 3-SAT

Lemma

2. 3-SAT *is* NP-*complete.*

Proof (Details).

Gate g_i	Clause	Semantics
$\bot$	$\{\overline{y_i}\}$	$y_i = 0$
$\top$	$\{y_i\}$	$y_i = 1$
x_j	$\{\overline{y_i}, x_j\}, \{\overline{x_j}, y_i\}$	$y_i \leftrightarrow x_j$
$(\neg, j)$	$\{\overline{y_i}, \overline{y_j}\}, \{y_i, y_j\}$	$y_i \leftrightarrow \overline{y_j}$
$(\wedge, j, k)$	$\{\overline{y_i}, y_j\}, \{\overline{y_i}, y_k\}, \{\overline{y_j}, \overline{y_k}, y_i\}$	$y_i \leftrightarrow (y_j \wedge y_k)$

- Finally, add $\{y_m\}$
- All clauses together form ψ_c

NP-completeness of 3-SAT (Cont.)

Lemma

2. 3-SAT *is* NP-*complete.*

Proof (Details, Cont.).

- If c is satisfiable, then also ψ_c:
  - Use assignment α of c for $x_1, \ldots, x_N$
  - Value $f_c^{(j)}(\alpha)$ at gate g_j as value for y_j
  - By construction, all clauses true, thus ψ_c satisfied
- If c not satisfiable, then neither ψ_c:
  - $f_c(\alpha) = 0$ for all α
  - Thus, $f_c^{(m)}(\alpha)$ always 0 ("top level gate")
  - If all clauses satisfied, $y_m = f_c^{(m)}(\alpha)$, but then $\{y_m\}$ not satisfied
- Thus, $c \in$ CIRSAT $\iff$ $\psi_c \in$ 3-SAT

Graph problems

- So far: *Satisfiability problems*
  - For boolean formulas (SAT, 3-SAT)
  - For boolean circuits (CIRSAT)
- Now: *Graph problems*
  - Undirected graph: $G = (V, E)$
  - V are the *nodes*
  - $E \subseteq \binom{V}{2}$ are the *edges*
  - Efficient encoding possible (adjacency matrix or list)
- Problems consider different properties of graphs

Independent Set Problem

- First problem: *Independent set problem*
  - Given: Undirected graph $G = (V, E)$ and number k
  - Question: Is there $I \subseteq V$ such that
    1. $\|I\| = k$, and
    2. No two nodes in I are connected?
- Formally:

Definition

The *independent set problem* is defined as:

$$\text{INDEPSET} := \left\{ (G, k) \mid \exists I \subseteq V(G) : \|I\| = k \wedge \binom{I}{2} \cap E(G) = \emptyset \right\}$$

- Turns out: Very difficult (i.e. NP-complete)

INDEPSET is NP-complete

Lemma

INDEPSET *is* NP*-complete.*

Proof (Sketch).

- INDEPSET $\in$ NP: Set I is the witness
- NP-completeness via 3-SAT $\leq_m^p$ INDEPSET:
  - Given φ with k clauses, construct G
  - Each literal is a node
  - Connect literals *from same clause* (triangles)
  - Connect *complementary* literals
- If φ satisfiable:
  Choose one satisfied literal in each clause for I
- If G has k-independent set:
  Represents a satisfying truth assignment

CLIQUE is NP-complete

- *Clique problem*
  - Given: Undirected graph $G = (V, E)$ and number k
  - Question: Is there $C \subseteq V$ such that
    1. $\|C\| = k$?
    2. *All nodes* in C are pairwise connected (a "k-clique")
- Formally:

Definition (Clique Problem)

The *clique problem* is defined as:

$$\text{CLIQUE} := \left\{ (G, k) \mid \exists C \subseteq V(G) : \|C\| = k \wedge \binom{C}{2} \subseteq E \right\}$$

- NP-complete! (See exercises)

NODECOVER is NP-complete

- *Node cover problem*
  - Given: Undirected graph $G = (V, E)$ and number k
  - Question: Is there $N \subseteq V$ such that
    1. $\|N\| = k$?
    2. *Each edge* in E contains a node from N
- Formally:

Definition (Node Cover Problem)

The *node cover problem* is defined as:

$$\text{NODECOVER} := \{(G, k) \mid \exists N \subseteq V(G) : \|N\| = k \wedge \forall e \in E(G) : e \cap N \neq \emptyset\}$$

- NP-complete! (See exercises)

HAMILTONPATH is NP-complete

- *Hamilton path problem*
  - Given: Undirected graph $G = (V, E)$
  - Question: Is there a path $p = (p_0, \ldots, p_k)$ in G such that:
    - ⋆ All nodes p_i are *pairwise different*? ("Hamilton path")
- Formally:

Definition (Hamilton Path Problem)

The *Hamilton Path Problem* is defined as:

$$\text{HAMILTONPATH} := \{G \mid \exists p : p \text{ is Hamilton path in } G\}$$

- NP-complete!

HITTINGSET is NP-complete

- *Hitting set problem*
  - Given:
    - A set A
    - A collection $C = (C_1, \ldots C_m)$ of subsets of A: $\forall i : C_i \subseteq A$
    - A number k
  - Question: Is there a set $H \subseteq A$ such that:
    1. $\|H\| = k$
    2. H contains an element from each $C_i \in C$ ("k-hitting set")
- Not a graph problem, but related
- Formally:

Definition (Hitting Set Problem)

The *hitting set problem* is defined as:

$$\text{HITTINGSET} := \{(A, C, k) \mid \exists H \subseteq A : \|H\| = k \wedge \forall C_i \in C : H \cap C_i \neq \emptyset\}$$

- NP-complete! (See exercises)

TSP is NP-complete

- *Travelling Salesman Problem*
  - Given:
    - n cities with a distance matrix $D \in \mathbb{N}^{n \times n}$
    - A number k
  - Question: Is there a tour through all cities such that
    1. Each city is visited exactly once, and
    2. The distance sum of the tour is at most k?
- Formally:

Definition (Travelling Salesman Problem)

The *Travelling salesman problem* is defined as:

$$\text{TSP} := \left\{ (D, k) \mid \exists \pi : \sum_{i=1}^{n} D[\pi(i), \pi(i+1)] \leq k \right\}$$

- NP-complete!

KNAPSACK is NP-complete

- *Knapsack Problem*
  - Given:
    - n items with values $V = (v_1, \ldots, v_n)$ and weights $W = (w_1, \ldots, w_n)$
    - A lower value limit l and an upper weight limit m
  - Question: Is there a selection $S \subseteq [1, n]$ of the items, such that
    1. The sum of the values is at least l, and
    2. The sum of the weights is at most m?
- Formally:

Definition (Knapsack problem)

The *Knapsack problem* is defined as:

$$\text{KNAPSACK} := \left\{ (V, W, l, m) \mid \exists S \subseteq [1, n] : \sum_{i \in S} w_i \leq l \wedge \sum_{i \in S} v_i \geq m \right\}$$

- NP-complete!

ILP is NP-complete

- *Integer linear programming problem*
  - Given: n linear inequalities in n variables with integer coefficients
  - Question: Is there an integer solution to that system?
- Formally:

Definition (Integer Linear Programming)

The *Integer linear programming problem* is defined as:

$$\text{ILP} := \{(A, b) \mid \exists x \in \mathbb{Z}^n : Ax \leq b\}$$

- NP-complete!
- Remark: *Linear programming* (allowing rationals) is in P!

BINPACK is NP-complete

- *Bin packing problem*
  - Given:
    - n items with sizes $A = (a_1, \ldots, a_n)$
    - b bins with capacity c each
  - Question: Is it possible to pack the items into the bins?
- Formally:

Definition (Bin Packing)

The *bin packing problem* is defined as:

$$\text{BINPACK} := \left\{ (A, b, c) \mid \exists \text{ partition } S_1, \ldots, S_b \text{ of } [1, n] \text{ s.t. } \forall i : \sum_{j \in S_i} a_j \leq c \right\}$$

- NP-complete!

Between P and NPC

- Many problems are NP-*complete*
- Many problems are *in* P
- Assume $P \neq NP$:
  - Are all problems either P or NP-complete?
  - No!

Lemma

If $P \neq NP$, *then there is a language* $L \in NP - (P \cup NPC)$.

(Without proof.)

- "Too easy" for NPC, "too difficult" for P
- Will see a candidate later

Pseudo-polynomial complexity

- Precise problem formulation may make a difference
  - Example: *Integer* linear programming
  - In P without restriction to integers
- *Representation* of problem instances may also matter:
  - KNAPSACK is NP-complete
  - *But:* Given n items and weight limit I, solve it in time $\mathcal{O}(n \cdot I)$
  - Still not polynomial in input: Input size is $\mathcal{O}(n \cdot \log(I))$
  - I is represented binary (or other k-ary)
- If polynomial in input *values* (not size/length): *Pseudo-polynomial*
- *Strong* NP*-completeness*:
  - NP-complete even if input values polynomially in input size
  - Equivalent: Input values are given in unary
  - Examples: All we saw except KNAPSACK
  - Don't have pseudo-polynomial algorithms (unless P = NP)

Unknown Relations

- NP- and co- NP-complete problems: Regarded as "difficult"
- How do they relate *to each other*?
  - Unknown: $\mathrm{NP} \stackrel{?}{=} \text{co-}\mathrm{NP}$
  - $\mathrm{NP} \neq \text{co-}\mathrm{NP} \implies \mathrm{P} \neq \text{co-}\mathrm{NP}$, thus $\mathrm{NP} \neq \text{co-}\mathrm{NP}$ *stronger*
  - Intuition: Efficiently verifiable proofs, *no* efficiently verifiable disproofs
- What about $\mathrm{NP} \cap \text{co-}\mathrm{NP}$?
  - Unknown: $\mathrm{P} \stackrel{?}{\subsetneq} \mathrm{NP} \cap \text{co-}\mathrm{NP}$
  - Intuition: Efficiently verifiable proofs *and* disproofs, not efficiently provable
- "Upper end": Also $\mathrm{NP} \stackrel{?}{\subsetneq} \mathrm{PSPACE}$ unknown
  - Intuition: Provable in polynomial space, but no (time-)efficiently verifiable proofs
  - Even $\mathrm{P} \stackrel{?}{\subsetneq} \mathrm{PSPACE}$ unknown

The Problems PRIMES and GI

- $P \neq NP$ not known, thus no language proven in $NP - (P \cup NPC)$
- Former candidate: PRIMES
  - Deciding primality of a number
  - Efficient *probabilistic* methods were known
  - Shown in 2002: PRIMES $\in$ P
- Another candidate: GI (graph isomorphism)
  - Given: Two graphs G_1 and G_2
  - Question: Are they *isomorphic*?

$$GI := \{(G_1, G_2) \mid \exists \pi : (e \in E(G_1) \iff \pi(e) \in E(G_2))\}$$

  - Isomorphism: Graphs "look the same" (same structure)
  - High practical relevance
  - Many approximations, but no exact complexity known
  - Own complexity class GI: Everything reducible to GI

Course Outline

0. Introduction
1. Basic Computability Theory
   - Formal Languages
   - Model of Computation: Turing Machines
   - Decidability, Undecidability, Semi-Decidability
2. Complexity Classes
   - Landau Symbols: The $\mathcal{O}(\cdot)$ Notation
   - Time and Space Complexity
   - Relations between Complexity Classes
3. Feasible Computations: P vs. NP
   - Proving vs. Verifying
   - Reductions, Hardness, Completeness
   - Natural NP-complete problems
4. Advanced Complexity Concepts
   - Non-uniform Complexity
   - Probabilistic Complexity Classes
   - Interactive Proof Systems

Course Outline

Uniform vs. Non-uniform Models

- Turing Machine: *One fixed* (finite) machine for *all* input sizes
- "One size fits it all" approach, *uniform* model
- Some situations: More hardwired information when size grows
  - Cryptography: Precomputed tables for different key sizes
  - Want to model such attackers
- Model this *non-uniform* notion using *advice*:
  - Machine gets advice string a_n in addition to input
  - One fixed string a_n for each input size n

Turing Machine with Advice

Definition (Turing Machine with Advice)

- A *Turing machine with advice* is a 6-tuple $M = (Q, \Gamma, \delta, q_0, F, A)$ with $Q, \Gamma, \delta, q_0, F$ as before and $A = \{a_n\}_{n \geq 0}$
- The set a A is called the *advice*.
- Language accepted by M:

$$L(M) := \{x \in \Sigma^* \mid \exists y, z \in \Gamma^* : (\varepsilon, q_0, x\#a_{|x|}) \vdash^* (y, q_{yes}, z)$$

(Separation symbol $\# \in \Gamma$)

Turing Machine with Advice: Remarks

- Classical Turing machine: Finite object
- Advice: *Infinite* object, external information to machine
- Difference to witness from NP:
  - *Witness* was different for *each input*, a specific proof
  - Only for the positive instances $x \in L(M)$
  - *Advice* is for *each input size*, a static computational "aid"
  - Fixed, the same a_n for each x of length n
- Very powerful without restrictions:
  - *Any* language L (even undecidable!) could be decided
  - Encode into a_n a large table with all words Σ^n
  - For each word x a bit: $x \in L$ or $x \notin L$
  - Machine can look up in table

Restricting the Advice: P/poly

- Restrict now advice polynomially

Definition

P/poly is the set of all languages L such that:

1. L is decided by a TM M with advice A exists, and
2. $\forall n : |a_n| \leq p(n)$ for some polynomial $p(n)$.

- Clear: P $\subseteq$ P/poly (just ignore the advice)
- Thus: If $\exists L \in$ NP with $L \notin$ P/poly then P $\neq$ NP!
- Difficulty: P/poly is powerful, contains still undecidable languages

P/poly and Undecidability

Lemma

P/poly *contains undecidable problems.*

Proof.

- First: There are unary undecidable languages
  - Encode H using one-symbol alphabet
- Second: All unary languages are in P/poly
  - For each size n, there is only one instance x_n
  - Set $a_n = 1$ if it is positive ($x_n \in L$)
  - Set $a_n = 0$ if it is negative ($x_n \notin L$)

□

- However, under reasonable assumptions: $\mathrm{NP} - \mathrm{P/poly} = \emptyset$

Circuit Characterization of Non-uniformity

- Defined P/poly using advice
- Alternative: Use *circuits*
- Recall: Circuit C_n
  - Gets input string $\alpha \in \{0,1\}^n$, and
  - Produces output bit $f_c(\alpha)$
  - Defines an *accepted language*:

$$L(C_n) := \{\alpha \in \{0,1\}^n \mid f_{C_n}(\alpha) = 1\}$$

- Works only with fixed instance size
  - Thus extend to *circuit family* $C = \{C_n\}_{n \geq 0}$:

$$L(C) := \{\alpha \in L(C_{|\alpha|})\}$$

- Non-uniformity: New device for each input size

Circuit Size

- Circuit family may have a non-finite representation (like advice)
- Very powerful without restrictions (like advice)
- *Resource bound*: $\text{size}(C_n)$
  - Defined via number of gates
  - Creates language class:

$$\text{DSIZE}(s(n)) := \{L(C) \mid \forall n \geq 0 : \text{size}(C_n) \leq s(n)\}$$

- TM can be simulated with quadratic circuit size:

Lemma

$\text{DTIME}(t(n)) \subseteq \text{DSIZE}(\mathcal{O}(t(n)^2))$ *(Without proof.)*

Circuit Characterization of P/poly

Lemma

$\mathrm{P/poly} = \mathrm{DSIZE}(n^{O(1)})$

Proof.

- First: Let $L \in \mathrm{P/poly}$ by a TM M with advice A
  - For each n, construct circuit C_n simulating M on Σ^n
  - a_n is only polynomially big, can be hardwired
  - Thus: $L \in \mathrm{DSIZE}(n^{O(1)})$
- Second: Let $L \in \mathrm{DSIZE}(n^{O(1)})$ via circuit family C
  - Use a TM for evaluating circuits
  - Advice A is the (encoded) circuit family C
  - Thus: $L \in \mathrm{P/poly}$

Probabilistic Computation

- Models so far: Had *exactly one* outcome
  - Classical deterministic Turing machine
  - Non-determinism with witness characterization
  - Circuits
- Relax this requirement: Introduce "coin-tosses"
  - At each step, machine can choose different next steps
  - Does so with a certain *probability*
  - Introduces uncertainty: Same input, different answers
- Produces still useful results if *low error probability*
  - Cryptography: Attacker may be happy to succeed 1% of the time
- Produces still useful results if *low error probability*
- Syntactically the same as non-deterministic machine, but:
  - Non-deterministic model: *Theoretic* model, implicit "$\exists$" quantification
  - Randomized model: *Practical* model, may be executed directly

Notation

- For $L \subseteq \Sigma^*$ let $\chi_L : \Sigma^* \to \{0,1\}$:

$$\chi_L(x) := \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

  χ_L is the *characteristic function* of L
- Assume from now on:
  - M outputs 1 for "accept"
  - M outputs 0 for "reject"
  - Thus, only q_{halt} instead of q_{yes} and q_{no}
- Allows expressions like "$M(x) = \chi_L(x)$"
- "Coin tosses" $r \in \{0,1\}^*$ as a second argument: $M(x,r)$
  - Notation of probability: $\text{Prob}_r[M(x,r) = 1]$
  - If M probabilistic, then r implicit: $\text{Prob}[M(x) = 1]$

One-sided Error: The class RP

- Didn't define accepted languages yet
- First type: *One-sided error*
  - Positive instances may be judged wrong (output 0 or 1)
  - Negative instances are always correct (output 0)

Definition

RP is the class of all L for which a polynomially time-bounded, probabilistic TM M exists, such that:

$$x \in L \implies \text{Prob}[M(x) = 1] \geq 1/2$$
$$x \notin L \implies \text{Prob}[M(x) = 1] = 0$$

- Symmetric behaviour: co- RP (always correct for $x \in L$)

P and NP vs. RP

- Clear: P $\subseteq$ RP (Machine is always correct, no coin tosses)
- RP $\subseteq$ NP:
  - RP: For $x \in L$, *at least half* the paths lead to 1
  - NP: For $x \in L$, *at least one* path leads to 1
  - Both produce 0 for all runs on $x \notin L$
- Or, using the witness-based characterization of NP:

	NP	RP
$x \in L$:	$\exists w : (x, w) \in R_L$	$\text{Prob}_r[(x, r) \in R_L] \geq 1/2$
$x \notin L$:	$\forall w : (x, w) \notin R_L$	$\forall r : (x, r) \notin R_L$

Error Bound Robustness

- Positive instances: Error probability $1/2$
- Assume: Error probability $2/3$
- Can be *reduced* to $1/2$:
  - Run the machine twice
  - Accept, if one of the runs accepted
  - For $x \notin L$: Still never acceptance
  - For $x \in L$: Error probability $(2/3)^2 = 4/9 < 1/2$
- In general, using more runs:

Lemma

If there is a polynomially time-bounded probabilistic TM M for L and a polynomial $p(n)$ such that:

$$x \in L \implies \text{Prob}[M(x) = 1] \geq 1/p(|x|)$$
$$x \notin L \implies \text{Prob}[M(x) = 1] = 0$$

Then $L \in$ RP. *(Without proof.)*

RP: Error Bound Robustness

- Seen: Very low acceptance ratio is "boosted" to $1/2$
- Can "boost" even further, *very close to* 1 ("almost always"):

Lemma

For each $L \in \mathrm{RP}$ and each polynomial $p(n)$,
there is a polynomially time-bounded probabilistic TM M such that:

$$x \in L \implies \mathrm{Prob}[M(x) = 1] \geq 1 - 2^{-p(|x|)}$$
$$x \notin L \implies \mathrm{Prob}[M(x) = 1] = 0$$

(Without proof.)

- Thus, two equivalent characterizations:
  1. Very *weak* ($1/p(|x|)$ bound) for proof *obligations*
  2. Very *strong* ($1 - 2^{-p(|x|)}$ bound) for proof *assumptions*

Two-sided Error: The class BPP

- Now allow errors for positive *and* negative instances

Definition

BPP is the class of all L for which a polynomially time-bounded, probabilistic TM M exists, such that:

$$\forall x \in L : \text{Prob}[M(x) = \chi_L(x)] \geq 2/3$$

- Notation from before:

$$x \in L \implies \text{Prob}[M(x) = 1] \geq 2/3$$
$$x \notin L \implies \text{Prob}[M(x) = 1] < 1/3$$

- *Symmetric definition*: BPP = co-BPP
- Definition is again quite robust, regarding the $2/3$

BPP: Error Bound Robustness

Lemma (Weak characterization of BPP)

If there is a polynomially time-bounded probabilistic TM M for L, a polynomial $p(n)$ and a computable function $f(n)$ such that:

$$x \in L \implies \text{Prob}[M(x) = 1] \geq f(|x|) + 1/p(|x|)$$
$$x \notin L \implies \text{Prob}[M(x) = 1] < f(|x|) - 1/p(|x|)$$

Then $L \in$ BPP. *(Without proof.)*

Lemma (Strong characterization of BPP)

For each $L \in$ BPP and each polynomial $p(n)$, there is a polynomially time-bounded probabilistic TM M such that:

$$\forall x \in L : \text{Prob}[M(x) = \chi_L(x)] \geq 1 - 2^{-p(|x|)}$$

(Without proof.)

Relations of BPP

- Clearly RP $\subseteq$ BPP (no error for $x \notin L$ with RP)
- Unknown relation between BPP and NP
- Note: Error rate can be made *exponentially small*
- "Efficient computation" nowadays often characterized with BPP
  - In fact, P $=$ BPP is a popular conjecture

Monte Carlo vs. Las Vegas

- Described machines always answer, sometimes wrong
  - *Monte Carlo Algorithms*
- Contrast: Always answer right, but sometimes with "I don't know"
  - *Las Vegas Algorithms*
- Denote "I don't know" with "$\perp$"

Definition

ZPP is the class of all L for which a polynomially time-bounded, probabilistic TM M exists, such that:

- $\forall x \in L : \text{Prob}[M(x) = \perp] \leq 1/2$, and
- $\forall x \in L, r : M(x, r) \neq \perp \implies M(x, r) = \chi_L(x)$

More Class Relationships

Lemma

1. $P \subseteq ZPP \subseteq RP \subseteq BPP$
2. $ZPP = RP \cap \text{co-}RP$
3. $BPP \subseteq P/poly$
4. $BPP = P$ *if pseudo random number generators exist. (Efficient derandomization)*

(Without proof.)

- Pseudo random number generators (PRNGs) output *looks* random
  - No observer can tell the difference
  - No observer can predict the next bit
  - PRNGs exist under certain (sophisticated) assumptions

Intuitive Notion of a Proof

- Proof: *Prover* and *Verifier*
- Prover convinces verifier of validity of some assertion
- In mathematics:
  - Prover writes down a list of steps
  - Verifier checks each step
- In general:
  - *Interaction* between the parties
  - Verifier asks questions (possibly adaptively)
  - Prover answers them
- Careful verifier is only convinced of valid assertions

Formalizing the Notion of a Proof

- Interpret NP as *non-interactive proofs*:
  - Supplied witness is the proof
  - Machine checking it is the verifier, working *efficiently*
  - Only true assertions ("$x \in L$") have a proof
  - "NP proof system"
- General notion with similar properties:
  1. *Efficiency* of the verifier
  2. *Correctness* requirement:

     Completeness: Each true assertion has a convincing proof strategy

     Soundness: No false assertion has a convincing proof strategy
- Will use *Interactive Turing Machines* for that

Interactive Turing Machine (ITM)

- Like ordinary Turing machine, but with
  - Communication tape and
  - Two communication states $q_?$ and $q_!$
- Operation of two *composed machines* $\langle M_1, M_2 \rangle$:
  - M_1 starts in q_0, M_2 waits in $q_?$
  - M_1 runs, writes message on *shared communication* tape
  - M_1 switches to $q_?$, M_2 switches to $q_!$
  - M_2 runs, M_1 waits
  - .. And so on, back and forth ..
  - Finally, M_2 stops
- Period between control switches: A *round*
- *Output*: Tape contents of M_2 after halting

Interactive Proof System

Definition (Interactive Proof System)

An *interactive proof system* for L is a pair $\langle P, V \rangle$ of ITMs with:

1. V is probabilistic and polynomially time-bounded.
2. Correctness requirement:

   Completeness: $\forall x \in L : \text{Prob}[\langle P, V \rangle(x) = 1] \geq 2/3$

   Soundness: $\forall x \notin L : \forall P^* : \text{Prob}[\langle P^*, V \rangle(x) = 1] < 1/3$

- Correctness is probabilistic
- Bounds:
  - Verifier is bounded
  - Prover is *not* bounded
- Soundness is against *all* provers (incl. very bad ones)
- Useful model for cryptographic protocols

IP hierarchy

Definition (IP hierarchy)

Let $r : \mathbb{N} \to \mathbb{N}$.

- IP($r(n)$) contains all L for with interactive proof systems $\langle P, V \rangle$ such that on common input x, at most $r(|x|)$ rounds are used.
- IP contains all L having interactive proof systems:

$$\mathrm{IP} := \bigcup_r \mathrm{IP}(r(n))$$

Properties of IP

- Clearly NP $\subseteq$ IP:
  - Just one round
  - Prover just writes witness
  - Verifier checks it (even deterministically)
  - Thus, *interaction* necessary to gain expressiveness
- Also *randomness* necessary:
  - Let $\langle P, V \rangle$ without random choices
  - P always knows exactly answer of V
  - Thus: Doesn't need to ask, can calculate answer itself
  - Therefore, only one final message needed
  - This is an NP *proof system*!
- Further: IP $=$ IP$(n^{\mathcal{O}(1)})$
  - V can only do polynomially many steps
  - Thus: Number of rounds *polynomially bounded*

IP: Error Bound Robustness

- As for BPP, bounds can be minimized:

Lemma

For each $L \in \mathsf{IP}$ and each polynomial $p(n)$, there is an interactive proof system $\langle P, V \rangle$ with strong correctness properties:

Completeness: $\forall x \in L : \mathrm{Prob}[\langle P, V \rangle(x) = 1] \geq 1 - 2^{-p(|x|)}$

Soundness: $\forall x \notin L : \forall P^* : \mathrm{Prob}[\langle P^*, V \rangle(x) = 1] < 2^{-p(|x|)}$

(Without proof.)

- Price for that:
  - Increased number of rounds (serial repetitions), or
  - Increased message sizes (parallel repetitions)

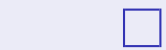
Example: Graph Non-Isomorphism

Theorem

$\mathsf{GNI} := \overline{\mathsf{GI}} \in \mathsf{IP}$.

Proof (Sketch).

- Given G_1, G_2, show they are not isomorphic
- Idea:
  - "Shuffle" one of them
  - Only possible to find which one it was, if G_1, G_2 not isomorphic
- Thus, protocol:
  - V chooses $i \in \{1, 2\}$ and permutation π
  - V sends $H := \pi(G_i)$
  - P finds j such that G_j is isomorphic to H, sends j
  - V checks whether $i = j$
- If $(G_1, G_2) \in \mathsf{GNI}$, P can find correct j (P is unbounded!)
- If $(G_1, G_2) \notin \mathsf{GNI}$, P can only guess and fails with 50% chance



IP: More properties

- Note: Protocol was just two rounds, GNI $\in$ IP(2)
- GNI $\in$ co- NP, not known whether in P or NP
- Indeed, IP is quite strong:

Theorem

1. $(\text{NP} \cup \text{co- NP}) \subseteq \text{IP}$
2. $\text{IP} = \text{PSPACE}$ *(Without proof.)*

- Interesting extension: *Zero knowledge proof systems*
  - Verifier does not gain knowledge
  - Needs definition of "knowledge":
    Information that can not be efficiently computed
  - Used for *secrecy properties* in cryptography

Course Outline

0. Introduction
1. Basic Computability Theory
   - Formal Languages
   - Model of Computation: Turing Machines
   - Decidability, Undecidability, Semi-Decidability
2. Complexity Classes
   - Landau Symbols: The $\mathcal{O}(\cdot)$ Notation
   - Time and Space Complexity
   - Relations between Complexity Classes
3. Feasible Computations: P vs. NP
   - Proving vs. Verifying
   - Reductions, Hardness, Completeness
   - Natural NP-complete problems
4. Advanced Complexity Concepts
   - Non-uniform Complexity
   - Probabilistic Complexity Classes
   - Interactive Proof Systems