Review session: Exam question 4
P, NP, and Cook's theorem.

- Lecture 15
  - Formal Languages, Turing machines, and the class P.
  - Miltersen: P, NP, and NPC, Sections 1-2.
- Lecture 16 (one hour)
  - The class NP. Reductions and the class NPC.
  - Miltersen: P, NP, and NPC, Sections 3-4.
- Lecture 17
  - Representations of Boolean functions. Truth tables, formulas, and Circuits. SAT and Circuit SAT.
  - Miltersen: P, NP, and NPC, pages 14-17 and 22-23.
- Lecture 18 (one hour)
  - Cook's Theorem
  - Miltersen: P, NP, and NPC, pages 16-22.

The theory of NP-completeness

- A mathematical tool for predicting that certain problems (”NP-hard problems”) ***cannot*** be solved by ”efficient algorithms”: ***algorithms running in polynomial time in the worst case***.
  - Main motivation for having such a tool: Saves (human) time!

- P: formalization of ”efficient computation”.
- NP: formalization capturing problems ”solvable by combinatorial exhaustive search”.

- Analogous theory: Undecidability (dBerLog)
  - Unlike in the case of undecidability theory, our theory needs to make an unproven assumption (P ≠ NP) to make the desired negative conclusions.
  - Like in the case of undecidability theory, the ”meat” of the theory consists of *reductions* between problems.

NP-completeness

”NP-complete problems”

… ⟷ **TSP** ⟷ **Mixed Integer Linear Programming**

Exponential

No reduction, unless P=NP

Efficient
by **Local Search**

Linear Programming

Min Cost Flow

Max Flow

Maximum matching

Shortest paths

↑ = reduction

Models of Computation

- **Model 1:** Our computer holds exact real numbers. The size of the input is the number of real numbers in the input. The time complexity of an algorithm is the number of arithmetic operations performed.

- **Model 2:** Our computer holds bits and bytes. The size of the input is the number of bits in the input. The time complexity of an algorithm is the number of bit-operations performed.

- We know an efficient algorithm for linear programming in Model 2 but not model 1.

- ***The NP-completeness theory is intended to capture Model 2 and not Model 1.***

Mathematical Foundation

- Our choices:
  - We model everything as formal languages over the binary alphabet, $\{0,1\}$.
  - We use the one-tape sequential Turing machine as our model of computation.
  - We study worst-case running time.
  - We equate “efficiency” with “polynomial” (in contrast to, say, “exponential”)

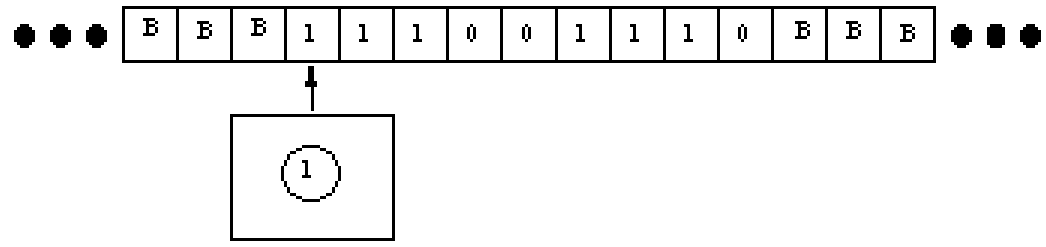
- Canonical decision problem for language L:
  - Given x, is $x \in L$?

Polynomial Church Turing thesis

- A decision problem can be solved in polynomial time by using a reasonable sequential model of computation if and only if it can be solved in polynomial time by a Turing machine.

Input: integer $n > 1$.

1. If ($n = a^b$ for $a \in \mathcal{N}$ and $b > 1$), output COMPOSITE.
2. Find the smallest r such that $o_r(n) > \log^2 n$.
3. If $1 < (a, n) < n$ for some $a \leq r$, output COMPOSITE.
4. If $n \leq r$, output PRIME.[1]
5. For $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ do
   if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$, output COMPOSITE;
6. Output PRIME.



Formal Languages

- We encode combinatorial objects and numbers in a standard way.
  - Strings over Σ: Encode as string over $\{0,1\}$ by concatenating encodings of Σ.
  - Pairs, tuples: $\langle x_1 x_2 \cdots x_n, y_1 y_2 \cdots y_m \rangle \coloneqq x_1 0 x_2 0 \cdots x_n 011 y_1 0 y_2 0 \cdots y_m 0$, etc.
  - Integers: Binary encoding.
  - Rationals: Pair of numerator/denominator.
  - Graphs: Adjacency matrix or Edge list.
  - …

- A representation of objects (say graphs, numbers) as strings is ***good*** if the language of valid representations is in **P**.
  - The will allow us to “forget” about malformed instances. We lump these together with negative instances of decision problems.

Stand-in languages for functions and optimization problems

- For a function $f\colon \{0,1\}^* \to \{0,1\}^*$, define
$$L_f = \{\langle x, b(j), y\rangle \mid f(x)_j = y\}$$

- For an optimization problem OPT: Given description of feasible set F, and objective function f, find $x \in F$ maximizing (minimizing) $f(x)$, define
$$L_{\mathrm{OPT}} = \{\langle \mathrm{desc}(F), \mathrm{desc}(f), K\rangle \mid \text{there exits } x \in F \text{ with } f(x) \geq K\}$$
(in case of minimization, we consider $f(x) \leq B$ instead)

- L_{OPT} may be easy to solve even though OPT is hard to solve, so L_{OPT} is not a perfect stand-in.
- However, an algorithm for OPT immediately yields an algorithm for L_{OPT}, so L_{OPT} can still be used to argue that OPT does not have a certain type of algorithms.

Polynomial time computable maps

- A map f: $\{0,1\}^* \to \{0,1\}^*$ is called polynomial time computable if for some polynomial *p*,
  - For all *x*, $|f(x)| \leq\ p(|x|)$.
  - $L_f \in\ \boldsymbol{P}$.

- Two different representations of objects are called ***polynomially equivalent*** if we may translate between them using polynomial time computable maps.

Proposition

- Given two good, polynomially equivalent representations of the instances of a decision problem, resulting in languages L_1 and L_2 we have $L_1 \in \mathbf{P}$ if and only if $L_2 \in \mathbf{P}$

P and NP

- **P** := the class of decision problems (languages) decided by a Turing machine so that for some polynomial p and all x, the machine terminates after at most $p(|x|)$ steps on input x.

- **NP** := the class of languages L for which there is a language L' in **P** and a polynomial p so that:

$$\forall x : x \in L \Leftrightarrow [\exists y \in \{0,1\}^* : |y| \leq p(|x|) \wedge \langle x, y \rangle \in L']$$

Decoding the definition

L is in **NP** iff there is a language *L'* in **P** and a polynomial *p* so that:

$$\forall x : x \in L \Leftrightarrow [\exists y \in \{0,1\}^* : |y| \leq p(|x|) \wedge \langle x, y \rangle \in L']$$

Symbol	Meaning	Example: 3 Coloring problem
L	The decision problem	Given a graph, is it 3-colorable?
x	An instance of decision problem	An undirected graph G
y	An element of the search space (possible solutions)	A coloring of the vertices of G
p(n)	Upper bound for encoding length of elements of search space.	2n
L'	Task of checking whether a possible solution y is an actual solution	Is the coloring of the vertices a valid coloring?

Finding the witness

L is in **NP** iff there is a language L' in **P** and a polynomial p so that:

$$\forall x : x \in L \Leftrightarrow [\exists y \in \{0,1\}^* : |y| \leq p(|x|) \wedge \langle x, y \rangle \in L']$$

Given $L \in \mathbf{NP}$, define

$$L_{\text{pre}} = \{ \langle x, z \rangle \mid \exists y \in \{0,1\}^* : |y| \leq p(|x|) \wedge \langle x, y \rangle \in L' \wedge z \sqsubseteq y \}$$

where $z \sqsubseteq y$ denotes that z is a prefix of y.

If $L_{\text{pre}} \in \mathbf{P}$ then there exists a polynomially time computable function f_L such that $x \in L$ implies that $\langle x, y \rangle \in L'$

Example: Finding mathematical proofs of provable statements in time polynomial in the length of the statement and length of shortest proof, assuming $\mathbf{P} = \mathbf{NP}$.

Reductions

- A ***reduction*** r of L_1 to L_2 is a polynomial time computable map so that

$$\forall\, x\colon \quad x \in L_1 \text{ iff } \quad r(x) \in L_2$$

- We write $L_1 \leq L_2$ if L_1 reduces to L_2.

Transitivity:

$$L_1 \leq L_2 \ \wedge\ L_2 \leq L_3 \ \Rightarrow\ L_1 \leq L_3$$

Downward closure of P:

$$L_1 \leq L_2 \ \wedge\ L_2 \in \boldsymbol{P} \ \Rightarrow\ L_1 \in \boldsymbol{P}$$

NP-hardness and NP-completeness

- A language L is called **NP**-hard iff

$$\forall L' \in \mathbf{NP} : L' \leq L$$

- **NPH** := The set of NP-hard languages.

- **Proposition**: If some **NP**-hard language is in **P**, then $\mathbf{P} = \mathbf{NP}$.

- A language $L \in \boldsymbol{NP}$ that is **NP**-hard is called **NP**-complete.
- **NPC** := the class of **NP**-complete problems.

- **Proposition**:

$$L \in \mathbf{NPC} \Rightarrow [L \in \mathbf{P} \Leftrightarrow \mathbf{P} = \mathbf{NP}]$$

Boolean Formulas and Circuits

- A circuit *C* is a directed ***acyclic*** graph. Nodes in *C* are called ***gates***.
- Gate types: Variables $(x_1, x_2, \ldots, x_n)$, Constants, AND, OR, NOT, COPY.
- m gates are distinguished output gates.

The circuit C computes a Boolean function $C: \{0,1\}^n \to \{0,1\}^m$

- A formula is a circuit where the outdegree of every gate is 1 (except for the output gate which has outdegree 0).

- A CNF is any conjunction of ***clauses*** (disjunctions of literals).
  A DNF is any disjunction of ***terms*** (conjunctions of literals).

- Any function on n variables can be described by a CNF (DNF) formula containing at most 2^n clauses (terms), each containing at most n literals.

Proving **NP**-hardness

To start:

SAT

- Given: CNF formula F on n variables.
- Question: Does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$?

Cook's theorem

SAT is **NP**-complete

And from then on:

- **Lemma**: If L_1 is **NP**-hard and $L_1 \leq L_2$ then L_2 is **NP**-hard.

Proof of Cook's Theorem

SAT

- Given: CNF formula F on n variables.
- Question: Does there exist $x \in \{0,1\}^n$ such that $F(x) = 1$?

CircuitSAT

- Given: Boolean Circuit C on n variables.
- Question: Does there exist $x \in \{0,1\}^n$ such that $\mathrm{C}(x) = 1$?

Proof steps:

1. CircuitSAT is **NP**-hard
2. CircuitSAT $\leq$ SAT

CircuitSAT is NP-hard

Main technical content:

Lemma
Let M be a Turing Machine running in time at most $q(n)$ on inputs of length n, where q is a polynomial. Then given fixed input length n there is a circuit C_n with at most $O(q(n)^2)$ gates such that

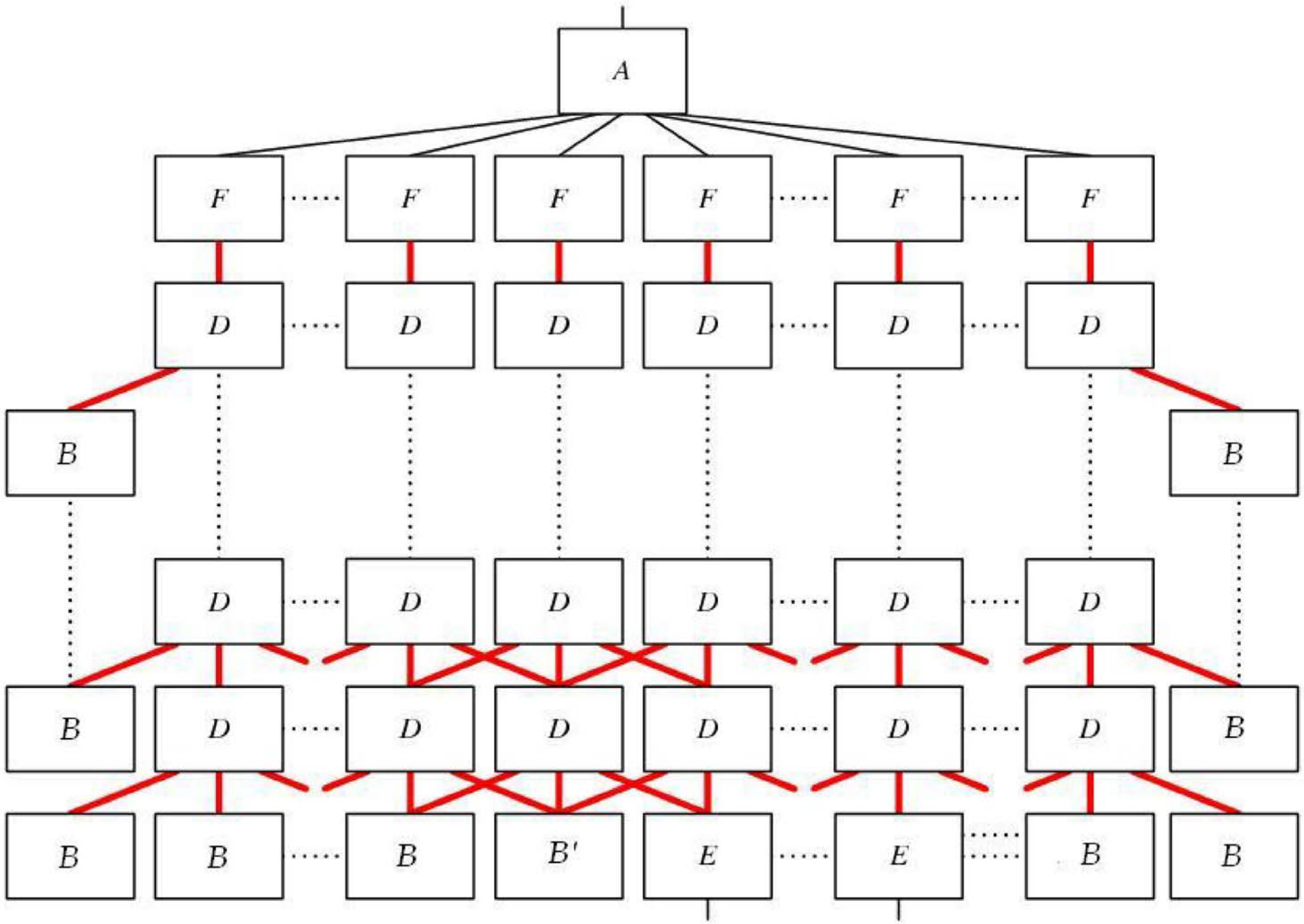
$$\forall x \in \{0,1\}^n : C_n(x) = 1 \Leftrightarrow x \in L(M)$$

Furthermore the function $1^n \mapsto C_n$ is polynomial time computable.

Proof: The tableau method: Representing computation tableau of M with cell state vectors $c_{i,t} \in \{0,1\}^s$, where i is cell index and t is time.

Crucial observation: $c_{i,t} = h\left(c_{t-1,i-1}, c_{t-1,i}, c_{t-1,i+1}\right)$,
where h is **fixed** Boolean function, hence computable by **fixed** Boolean circuit D.

Assembling the circuit



CircuitSAT is NP-hard

Given L in NP we have a language L' in **P** and a polynomial p so that:

$$\forall x : x \in L \Leftrightarrow [\exists y \in \{0,1\}^* : |y| \leq p(|x|) \wedge \langle x, y \rangle \in L']$$

Reduction r from L to CircuitSAT (cheating slightly).

On input x do:

- Apply previous lemma to L' to compute circuit C such that
$$\langle x, y \rangle \in L' \Leftrightarrow C(\langle x, y \rangle) = 1$$
- Hardwire input x in C to obtain circuit C_x such that
$$\langle x, y \rangle \in L' \Leftrightarrow C_x(y) = 1$$

Now (still cheating slightly):

$$x \in L \Leftrightarrow \exists y : \langle x, y \rangle \in L' \Leftrightarrow \exists y : C_x(y) = 1 \Leftrightarrow C_x \in \text{CircuitSAT}$$

Cheat: Boolean circuits take a fixed number of input bits!

Reduction from CircuitSAT to SAT

Let C be a Boolean circuit on n variables.

Construct CNF formula F with

- Variables: One variable g for every gate g of C.
- Clauses:
  - For each gate of C, clauses that express the computation of the gate.
    E.g., $g \Leftrightarrow h_1 \wedge h_2$ expresses that gate g is the Boolean conjunction of gates h_1 and h_2. For every gate this is a Boolean function on at most 3 variables, which can be expressed as a CNF formula.
    $$(\neg g \vee h_1) \wedge (\neg g \vee h_2) \wedge (g \vee \neg h_1 \vee \neg h_2)$$
  - For the output gate g_{out} of C, the unit clause (g_{out}).

From tutorials: coNP

coNP := $\{\{0,1\}^* \setminus L \mid L \in \mathbf{NP}\}$

DNF tautology is **coNP**-complete.

Hypothesis: **NP** ≠ **coNP** (stronger than **P**≠**NP**)

- Implies that there are propositional statements that are universally true, but with no short mathematical proof of this.
- Implies that NP-hard optimization problems can not be solved to optimality by local search algorithms that in each iteration consider polynomially many neighbors.
- **FAC** is not **NP**-complete, since **FAC** ∈ **coNP** ∩ **NP.**

FAC

- Given: Integers n and m.
- Question: Does n have a prime factor smaller than m?