Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ) Υπολογιστική Πολυπλοκότητα

Εργαστήριο Λογικής και Επιστήμης Υπολογισμών Εθνικό Μετσόβιο Πολυτεχνείο

2018-2019



Computational Complexity

Graduate Course

Antonis Antonopoulos

Computation and Reasoning Laboratory National Technical University of Athens

This work is licensed under a Creative Commons Attribution-NonCommercial- NoDerivatives 4.0 International License.

e148abab0c9923b3cc835507f68f4caf404710e9

Πληροφορίες Μαθήματος

Θεωρητική Πληροφορική Ι (ΣΗΜΜΥ) Υπολογιστική Πολυπλοκότητα (ΑΛΜΑ)

- Διδάσκοντες: Σ. Ζάχος, Ά. Παγουρτζής
- Βοηθοί Διδασκαλίας: Α. Αντωνόπουλος, Α. Χαλκή
- Επιμέλεια Διαφανειών: Α. Αντωνόπουλος
- Δευτέρα: 17:00 20:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
 Πέμπτη: 15:00 17:00 (1.1.31, Παλιά Κτίρια ΗΜΜΥ, ΕΜΠ)
- Ώρες Γραφείου: Μετά από κάθε μάθημα, Παρασκευή 13:00-14:00
- Σελίδα: http://courses.corelab.ntua.gr/complexity
- Βαθμολόγηση:

Αιαγώνισμα: 6 μονάδες Ασκήσεις: 2 μονάδες Ομιλία: 2 μονάδες Quiz: 1 μονάδα

Bibliography

Textbooks

- C. Papadimitriou, Computational Complexity, Addison Wesley, 1994
- 2 S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009
- 3 O. Goldreich, Computational Complexity: A Conceptual Perspective, Cambridge University Press, 2008

Lecture Notes

- 1 L. Trevisan, Lecture Notes in Computational Complexity, 2002, UC Berkeley
- 2 J. Katz, Notes on Complexity Theory, 2011, University of Maryland

-

Why Complexity?

Contents

Introduction

- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

Algorithms & Complexity
○●○○○

Problems...

Turing Machines

Undecidability

bility Algorithms & Complexity

○○●○○

Pro blems,...

Turing Machines

Undecidability

Decision Problems

- Have answers of the form "yes" or "no"
- Encoding: each instance x of the problem is represented as a *string* of an alphabet Σ ($|\Sigma| \ge 2$).
- Decision problems have the form "Is x in L?", where L is a language, $L \subseteq \Sigma^*$.
- So, for an encoding of the input, using the alphabet Σ , we associate the following language with the decision problem Π :

 $L(\Pi) = \{x \in \Sigma^* \mid x \text{ is a representation of a "yes" instance of the problem } \Pi\}$

Example

- Given a number x, is this number prime? ($x \in PRIMES$)
- Given graph *G* and a number *k*, is there a clique with *k* (or more) nodes in *G*?

- Computational Complexity: Quantifying the amount of computational resources required to solve a given task.
 Classify computational problems according to their inherent difficulty in complexity classes, and prove relations among them.
- Structural Complexity: "The study of the relations between various complexity classes and the global properties of individual classes. [...] The goal of structural complexity is a thorough understanding of the relations between the various complexity classes and the internal structure of these complexity classes." [J. Hartmanis]

Search Problems

- Have answers of the form of an **object**.
- **Relation** R(x, y) connecting instances x with answers (objects) y we wish to find for x.
- Given instance x, find a y such that $(x, y) \in R$.

Example

FACTORING: Given integer N, find its prime decomposition $N = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$.

Optimization Problems

- For each instance x there is a **set of Feasible Solutions** F(x).
- To each $s \in F(x)$ we map a positive integer c(x), using **the** objective function c(s).
- We search for the solution $s \in F(x)$ which minimizes (or maximizes) the objective function c(s).

Example

• The Traveling Salesperson Problem (TSP): Given a finite set $C = \{c_1, \ldots, c_n\}$ of cities and a distance $d(c_i, c_j) \in \mathbb{Z}^+, \forall (c_i, c_j) \in C^2$, we ask for a permutation π of C, that minimizes this quantity:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

Algorithms & Complexity

○○○○●

Problems...

Turing Machines

Undecidability

Algorithms & Complexity

Turing Machines

Undecidability

What have we learned?

- Computational Complexity classifies problems into classes, and studies the relations and the structure of these classes.
- We have decision problems with boolean answer, or function/optimization problems which output an object as an answer.
- Given some nice properties of polynomials, we identify polynomial-time algorithms as efficient algorithms.

A Model Discussion

Algorithms & Complexity

000000

Problems....

- There are many computational models (RAM, Turing Machines etc).
- The Church-Turing Thesis states that all computation models are equivalent. That is, every computation model can be simulated by a Turing Machine.
- In Complexity Theory, we consider **efficiently computable** the problems which are solved (aka the languages that are decided) in **polynomial number of steps** (*Edmonds-Cobham Thesis*).

Efficiently Computable \equiv Polynomial-Time Computable

Contents

Introduction

Turing Machines

- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

Definition

Turing Machines

Definition

A Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{\text{ves}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subset Q$ is the set of final states.
- $\delta: (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{S, L, R\}$ is the transition function.
- A TM is a "programming language" with a single data structure (a tape), and a cursor, which moves left and right on the tape.
- Function δ is the **program** of the machine.

Algorithms & Complexity

Definitions

Turing Machines

Undecidability

Definition

If for a language L there is a TM M, which if $x \in L$ then M(x) = "yes", and if $x \notin L$ then $M(x) \uparrow$, we call L recursively enumerable.

*By $M(x) \uparrow$ we mean that M does not halt on input x (it runs forever).

Theorem

If L is recursive, then it is recursively enumerable.

Proof: Exercise

Definition

If f is a function, $f: \Sigma^* \to \Sigma^*$, we say that a TM M computes f if, for any string $x \in \Sigma^*$, M(x) = f(x). If such M exists, f is called a recursive function.

• Turing Machines can be thought as algorithms for solving string related problems.

Turing Machines and Languages

Definition

Algorithms & Complexity

Definitions

Let $L \subseteq \Sigma^*$ be a language and M a TM such that, for every string $x \in \Sigma^*$:

- If $x \in L$, then M(x) = "ves"
- If $x \notin L$, then M(x) = "no"

Then we say that M decides L.

- Alternatively, we say that M(x) = L(x), where $L(x) = \chi_L(x)$ is the characteristic function of L (if we consider 1 as "yes" and 0 as "no").
- If L is decided by some TM M, then L is called a **recursive** language.

Algorithms & Complexity

Turing Machines

Undecidability

Definitions

Multitape Turing Machines

• We can extend the previous Turing Machine definition to obtain a Turing Machine with multiple tapes:

Definition

A k-tape Turing Machine M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{\text{halt}}, q_{\text{ves}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta: (Q \setminus F) \times \Gamma^k \to Q \times (\Gamma \times \{S, L, R\})^k$ is the transition function.

Undecidability

Properties of Turing Machines

Bounds on Turing Machines

• We will characterize the "performance" of a Turing Machine by the amount of *time* and *space* required on instances of size *n*, when these amounts are expressed as a function of *n*.

Definition

Let $T : \mathbb{N} \to \mathbb{N}$. We say that machine M operates within time T(n) if, for any input string x, the time required by M to reach a final state is at most T(|x|). Function T is a **time bound** for M.

Definition

Let $S: \mathbb{N} \to \mathbb{N}$. We say that machine M operates within space S(n) if, for any input string x, M visits at most S(|x|) locations on its work tapes (excluding the input tape) during its computation. Function S is a **space bound** for M.

Algorithms & Complexity Turing Machines

Properties of Turing Machines

Linear Speedup

Linear Speedu

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within time T(n). Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within time $T'(n) = \varepsilon T(n) + n + 2$.

Proof: See Th.2.2 (p.32) in [1].

- If, for example, T is linear, i.e. something like cn, then this theorem states that the constant c can be made arbitrarily close to 1. So, it is fair to start using the $\mathcal{O}(\cdot)$ notation in our time bounds.
- A similar theorem holds for space:

Theorem

Let M be a TM that decides $L \subseteq \Sigma^*$, that operates within space S(n). Then, for every $\varepsilon > 0$, there is a TM M' which decides the same language and operates within space $S'(n) = \varepsilon S(n) + 2$.

Multitape Turing Machines

Theorem

Given any k-tape Turing Machine M operating within time T(n), we can construct a TM M' operating within time $\mathcal{O}\left(T^2(n)\right)$ such that, for any input $x \in \Sigma^*$, M(x) = M'(x).

Proof: See Th.2.1 (p.30) in [1].

• This is a strong evidence of the robustness of our model: *Adding a bounded number of strings does not increase their computational capabilities, and affects their efficiency only polynomially.*

Algorithms & Complexity

Turing Machines

Undecidability

NTMs

Nondeterministic Turing Machines

• We will now introduce an **unrealistic** model of computation:

Definition

A Turing Machine *M* is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2, q_3, \dots, q_n, q_{\text{halt}}, q_{\text{yes}}, q_{\text{no}}\}$ is a finite set of states.
- Σ is the alphabet. The tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- $\delta: (Q \setminus F) \times \Gamma \to Pow(Q \times \Gamma \times \{S, L, R\})$ is the transition **relation**.

Nondeterministic Turing Machines

- In this model, an input is accepted if **there is** *some sequence* of nondeterministic choices that results in "yes".
- An input is rejected if there is *no sequence* of choices that lead to acceptance.
- Observe the similarity with recursively enumerable languages.

Definition

We say that M operates within bound T(n), if for every input $x \in \Sigma^*$ and every sequence of nondeterministic choices, M reaches a final state within T(|x|) steps.

- The above definition requires that M does not have computation paths longer than T(n), where n = |x| the length of the input.
- The amount of time charged is the *depth* of the **computation tree**.

The amount of time charged is the aepar of the computation tree

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

Algorithms & Complexity

NTMs

Turing Machines

Undecidability 000000000

What have we learned?

- A recursive language is decided by a TM.
- A recursive enumerable language is accepted by a TM that halts only if $x \in L$.
- Multiple tape TMs can be simulated by a one-tape TM with quadratic overhead.
- Linear speedup justifies the $\mathcal{O}(\cdot)$ notation.
- Nondeterministic TMs move in "parallel universes", making different choices simultaneously.
- A Deterministic TM computation is a *path*.
- A Nondeterministic TM computation is a *tree*, i.e. exponentially many paths ran simultaneously.

Algorithms & Complexity

Diagonalization

Turing Machines

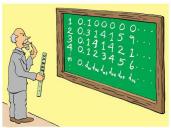
Undecidability

Diagonalization



Suppose there is a town with just one barber, who is male. In this town, the barber shaves all those, and only those, men in town who do not shave themselves. Who shaves the barber?

Diagonalization is a technique that was used in many different cases:



George showed it wouldn't fit in.

Simulation

Diagonalization

Diagonalization

Theorem

The functions from \mathbb{N} *to* \mathbb{N} *are uncountable.*

Proof: Let, for the sake of contradiction that are countable: ϕ_1, ϕ_2, \dots Consider the following function: $f(x) = \phi_x(x) + 1$. This function must appear somewhere in this enumeration, so let $\phi_v = f(x)$. Then $\phi_{y}(x) = \phi_{x}(x) + 1$, and if we choose y as an argument, then $\phi_{v}(y) = \phi_{v}(y) + 1. \square$

• Using the same argument:

Theorem

The functions from $\{0,1\}^*$ to $\{0,1\}$ are uncountable.

Algorithms & Complexity

Simulation

Turing Machines

Undecidability 00000000000

The Universal Turing Machine

- So far, our computational models are specified to solve a single problem.
- Turing observed that there is a TM that can simulate any other TM M, given M's description as input.

Theorem

There exists a TM \mathcal{U} such that for every $x, w \in \Sigma^*$, $\mathcal{U}(x, w) = M_w(x)$. Also, if M_w halts within T steps on input x, then U(x, w) halts within CT log T steps, where C is a constant independent of x, and depending only on M_w 's alphabet size number of tapes and number of states.

Proof: See section 3.1 in [1], and Th. 1.9 and section 1.7 in [2].

Machines as strings

- It is obvious that we can represent a Turing Machine as a string: just write down the description and encode it using an alphabet, e.g. $\{0,1\}.$
- We denote by $\lfloor M \rfloor$ the TM M's representation as a string.
- Also, if $x \in \Sigma^*$, we denote by M_x the TM that x represents.

Keep in mind that:

- Every string represents some TM.
- Every TM is represented by *infinitely many* strings.
- There exists (at least) an uncomputable function from $\{0,1\}^*$ to $\{0,1\}$, since the set of all TMs is countable.

Algorithms & Complexity

Unde cida bility

Turing Machines

Undecidability 0000000000

The Halting Problem

- Consider the following problem: "Given the description of a TM M, and a string x, will M halt on input x?" This is called the HALTING PROBLEM.
- We want to compute this problem!!! (Given a computer program and an input, will this program enter an infinite loop?)
- In language form: $H = \{ LM \mid x \mid M(x) \downarrow \}$, where " \downarrow " means that the machine halts, and " \tau " that it runs forever.

Theorem

H is recursively enumerable.

Proof: See Th.3.1 (p.59) in [1]

• In fact, H is not just a recursively enumerable language: If we had an algorithm for deciding H, then we would be able to derive an algorithm for deciding any r.e. language (**RE**-complete).

The Halting Problem

But....

Theorem

H is not recursive.

Proof:

See Th.3.1 (p.60) in [1]

• Suppose, for the sake of contradiction, that there is a TM M_H that decides H.

• Consider the TM D:

$$D(\sqcup M \sqcup) :$$
 if $M_H(\sqcup M \sqcup ; \sqcup M \sqcup) =$ "yes" **then** \uparrow **else** "yes"

- What is $D(\lfloor D \rfloor)$?
- If $D(\lfloor D \rfloor) \uparrow$, then M_H accepts the input, so $\lfloor D \rfloor$; $\lfloor D \rfloor \in H$, so $D(D) \downarrow$.
- If $D(\lfloor D \rfloor) \downarrow$, then M_H rejects $\lfloor D \rfloor$; $\lfloor D \rfloor$, so $\lfloor D \rfloor$; $\lfloor D \rfloor \notin H$, so $D(D) \uparrow$.

Algorithms & Complexity

Turing Machines

Undecidability

0000000000

Undecidability

More Undecidability

- The HALTING PROBLEM, our first undecidable problem, was the first, but not the only undecidable problem. Its spawns a wide range of such problems, via reductions.
- To show that a problem A is undecidable we establish that, if there is an algorithm for A, then there would be an algorithm for H, which is absurd.

Theorem

The following languages are not recursive:

- $\{M \mid M \text{ halts on all inputs}\}$
- $\{M; x \mid There \ is \ a \ y \ such \ that \ M(x) = y\}$
- $\{M; x \mid The \ computation \ of \ M \ uses \ all \ states \ of \ M\}$
- $\{M; x; y \mid M(x) = y\}$

- Recursive languages are a *proper* subset of recursive enumerable ones.
- Recall that the complement of a language L is defined as:

$$\overline{L} = \{ x \in \Sigma^* \mid x \notin L \} = \Sigma^* \setminus L$$

Theorem

Algorithms & Complexity

Unde cida bility

- ① If L is recursive, so is \overline{L} .
- ② *L* is recursive if and only if *L* and \overline{L} are recursively enumerable.

Proof: Exercise

- Let $E(M) = \{x \mid (q_0, \triangleright, \varepsilon) \stackrel{M*}{\rightarrow} (q, y \sqcup x \sqcup, \varepsilon)\}$
- \bullet E(M) is the language enumerated by M.

Theorem

L is recursively enumerable iff there is a TM M such that L = E(M).

Algorithms & Complexity

Unde cida bility

Turing Machines 0000000000 Undecidability 00000000000

Rice's Theorem

• The previous problems lead us to a more general conclusion:

Any non-trivial property of **Turing Machines is undecidable**

• If a TM M accepts a language L, we write L = L(M).

Theorem (Rice's Theorem)

Suppose that C is a proper, non-empty subset of the set of all recursively enumerable languages. Then, the following problem is undecidable:

Given a Turing Machine M, is $L(M) \in C$?

Rice's Theorem

Proof: See Th.3.2 (p.62) in [1]

- We can assume that $\emptyset \notin \mathcal{C}$ (why?).
- Since C is nonempty, $\exists L \in C$, accepted by the TM M_L .
- Let M_H the TM deciding the HALTING PROBLEM for an arbitrary input x. For each $x \in \Sigma^*$, we construct a TM M as follows:

$$M(y)$$
: if $M_H(x)$ = "yes" then $M_L(y)$ else \uparrow

• We claim that: $L(M) \in \mathcal{C}$ if and only if $x \in H$.

Proof of the claim:

- If $x \in H$, then $M_H(x) =$ "yes", and so M will accept y or never halt, depending on whether $y \in L$. Then the language accepted by M is exactly L, which is in C.
- If $M_H(x) \uparrow$, M never halts, and thus M accepts the language \emptyset , which is not in C. \square

Complexity Classes

Oracles & The Polynomial Hierarchy

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

What have we learned?

- TMs are encoded by strings.
- The Universal TM $\mathcal{U}(x, \lfloor M \rfloor)$ can simulate any other TM M along with an input x.
- The Halting Problem is recursively enumerable, but not recursive.
- Many other problems can be proved undecidable, by a *reduction* from the Halting Problem.
- Rice's theorem states that any non-trivial property of TMs is an undecidable problem.

Complexity Classes

Oracles & The Polynomial Hierarchy

Introductio

Parameters used to define complexity classes:

- Model of Computation (Turing Machine, RAM, Circuits)
- Mode of Computation (Deterministic, Nondeterministic, Probabilistic)
- Complexity Measures (*Time, Space, Circuit Size-Depth*)
- Other Parameters (Randomization, Interaction)

Our first complexity classes

Definition

Let $L \subset \Sigma^*$, and $T, S : \mathbb{N} \to \mathbb{N}$:

- We say that $L \in \mathbf{DTIME}[T(n)]$ if there exists a TM M deciding L, which operates within the *time* bound $\mathcal{O}(T(n))$, where n = |x|.
- We say that $L \in \mathbf{DSPACE}[S(n)]$ if there exists a TM M deciding L, which operates within *space* bound $\mathcal{O}(S(n))$, that is, for any input x, requires space at most S(|x|).
- We say that $L \in \mathbf{NTIME}[T(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within the time bound $\mathcal{O}\left(T(n)\right)$.
- We say that $L \in \mathbf{NSPACE}[S(n)]$ if there exists a *nondeterministic* TM M deciding L, which operates within space bound $\mathcal{O}(S(n))$.

Oracles & The Polynomial Hierarchy

Constructible Functions

Complexity Classes

Constructible Functions

• Can we use all computable functions to define Complexity Classes?

Theorem (Gap Theorem)

For any computable functions r and a, there exists a computable function f such that $f(n) \ge a(n)$, and

 $\mathbf{DTIME}[f(n)] = \mathbf{DTIME}[r(f(n))]$

- That means, for $r(n) = 2^{2^{f(n)}}$, the incementation from f(n) to $2^{2^{f(n)}}$ does not allow the computation of any new function!
- So, we must use some restricted families of functions:

Our first complexity classes

- The above are **Complexity Classes**, in the sense that they are sets of languages.
- All these classes are parameterized by a function T or S, so they are families of classes (for each function we obtain a complexity class).

Definition (Complementary complexity class)

For any complexity class C, coC denotes the class: $\{\overline{L} \mid L \in C\}$, where $\overline{L} = \Sigma^* \setminus L = \{ x \in \Sigma^* \mid x \notin L \}.$

• We want to define "reasonable" complexity classes, in the sense that we want to "compute more problems", given more computational resources.

Complexity Classes

Oracles & The Polynomial Hierarchy

Constructible Functions

Constructible Functions

Definition (Time-Constructible Function)

A nondecreasing function $T: \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \ge n$ and there is a TM M that computes the function $x \mapsto \lfloor T(|x|) \rfloor$ in time T(n).

Definition (Space-Constructible Function)

A nondecreasing function $S: \mathbb{N} \to \mathbb{N}$ is space-constructible if $S(n) > \log n$ and there is a TM M that computes S(|x|) using S(|x|)space, given x as input.

- The restriction T(n) > n is to allow the machine to read its input.
- The restriction $S(n) > \log n$ is to allow the machine to "remember" the index of the cell of the input tape that it is currently reading.
- Also, if $f_1(n)$, $f_2(n)$ are time/space-constructible functions, so are $f_1 + f_2, f_1 \cdot f_2$ and $f_1^{f_2}$.

Constructible Functions

Theorem (Hierarchy Theorems)

Let t_1 , t_2 be time-constructible functions, and s_1 , s_2 be space-constructible functions. Then:

- ① If $t_1(n) \log t_1(n) = o(t_2(n))$, then **DTIME** $(t_1) \subseteq \textbf{DTIME}(t_2)$.
- ② If $t_1(n+1) = o(t_2(n))$, then **NTIME** $(t_1) \subseteq \textbf{NTIME}(t_2)$.
- \bigcirc If $s_1(n) = o(s_2(n))$, then **DSPACE** $(s_1) \subseteq \mathbf{DSPACE}(s_2)$.
- 4 If $s_1(n) = o(s_2(n))$, then **NSPACE** $(s_1) \subseteq \textbf{NSPACE}(s_2)$.

Oracles & The Polynomial Hierarchy

Complexity Classes

Simplified Case of Deterministic Time Hierarchy Theorem

Proof (cont'd):

 $\exists n_0: n^{1.4} > cn \log n \ \forall n \geq n_0$ There exists a x_M , s.t. $x_M = \lfloor M \rfloor$ and $|x_M| > n_0$ (why?) Then, $\mathbf{D}(\mathbf{x_M}) = \mathbf{1} - \mathbf{M}(\mathbf{x_M})$ (while we have also that $D(x) = M(x), \ \forall x$) Contradiction!!

• So, we have the hierarhy:

$$\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^2] \subsetneq \mathbf{DTIME}[n^3] \subsetneq \cdots$$

• We will later see that the class containing the problems we can efficiently solve (recall the Edmonds-Cobham Thesis) is the class $\mathbf{P} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[n^c]$.

Simplified Case of Deterministic Time Hierarchy Theorem

Theorem

Complexity Classes

 $\mathbf{DTIME}[n] \subsetneq \mathbf{DTIME}[n^{1.5}]$

Proof (Diagonalization):

See Th.3.1 (p.69) in [2]

Let *D* be the following machine:

On input x, run for $|x|^{1.4}$ steps $\mathcal{U}(M_x, x)$; If $\mathcal{U}(M_x, x) = b$, then return 1 - b; Else return 0;

- Clearly, $L = L(D) \in \mathbf{DTIME}[n^{1.5}]$
- We claim that $L \notin \mathbf{DTIME}[n]$: Let $L \in \mathbf{DTIME}[n] \Rightarrow \exists M : M(x) = D(x) \ \forall x \in \Sigma^*$, and M works for $\mathcal{O}(|x|)$ steps.

The time to simulate M using \mathcal{U} is $c|x| \log |x|$, for some c.

Complexity Classes

Oracles & The Polynomial Hierarchy

Relations among Complexity Classes

- Hierarchy Theorems tell us how classes of the same kind relate to each other, when we vary the complexity bound.
- The most interesting results concern relationships between classes of different kinds:

Theorem

Suppose that T(n), S(n) are time-constructible and space-constructible functions, respectively. Then:

- **① DTIME**[T(n)] \subseteq **NTIME**[T(n)]
- **2 DSPACE**[S(n)] \subseteq **NSPACE**[S(n)]
- **3 NTIME** $[T(n)] \subseteq \mathbf{DSPACE}[T(n)]$
- **4 NSPACE**[S(n)] \subset **DTIME**[$2^{\mathcal{O}(S(n))}$

Corollary

$$\mathbf{NTIME}[T(n)] \subseteq \bigcup_{c>1} \mathbf{DTIME}[c^{T(n)}]$$

Oracles & The Polynomial Hierarchy 00000000000000000000

Complexity Classes Oracles & The Polynomial Hierarchy

Relations among Complexity Classes

Proof:

See Th.7.4 (p.147) in [1]

- Trivial
- 2 Trivial
- We can simulate the machine for each nondeterministic choice. using at most T(n) steps in each simulation. There are *exponentially* many simulations, but we can simulate them one-by-one, reusing the same space.
- 4 Recall the notion of a configuration of a TM: For a k-tape machine, is a 2k - 2 tuple: $(q, i, w_2, u_2, \dots, w_{k-1}, u_{k-1})$ How many configurations are there?
 - |Q| choices for the state
 - n+1 choices for i, and
 - Fewer than $|\Sigma|^{(2k-2)S(n)}$ for the remaining strings

So, the total number of configurations on input size n is at most $nc_1^{S(n)}=2^{\mathcal{O}(S(n))}$.

Oracles & The Polynomial Hierarchy

Complexity Classes

Relations among Complexity Classes

The essential Complexity Hierarchy

Definition

$$\mathbf{L} = \mathbf{DSPACE}[\log n]$$

$$\mathbf{NL} = \mathbf{NSPACE}[\log n]$$

$$\mathbf{P} = \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[n^c]$$

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[n^c]$$

$$\mathbf{PSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[n^c]$$

$$\mathbf{NPSPACE} = \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[n^c]$$

Proof (*cont'd*):

Relations among Complexity Classes

Definition (Configuration Graph of a TM)

The configuration graph of M on input x, denoted G(M, x), has as vertices all the possible configurations, and there is an edge between two vertices C and C' if and only if C' can be reached from C in one step, according to M's transition function.

- So, we have reduced this simulation to REACHABILITY* problem (also known as S-T CONN), for which we know there is a poly-time $(\mathcal{O}(n^2))$ algorithm.
- So, the simulation takes $(2^{\mathcal{O}(S(n))})^2 \sim 2^{\mathcal{O}(S(n))}$ steps. \square

*REACHABILITY: Given a graph G and two nodes $v_1, v_n \in V$, is there a path from v_1 to v_n ?

Complexity Classes

Oracles & The Polynomial Hierarchy

Relations among Complexity Classes

The essential Complexity Hierarchy

Definition

$$\begin{aligned} \mathbf{EXP} &= \bigcup_{c \in \mathbb{N}} \mathbf{DTIME}[2^{n^c}] \\ \mathbf{NEXP} &= \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}[2^{n^c}] \\ \mathbf{EXPSPACE} &= \bigcup_{c \in \mathbb{N}} \mathbf{DSPACE}[2^{n^c}] \\ \mathbf{NEXPSPACE} &= \bigcup_{c \in \mathbb{N}} \mathbf{NSPACE}[2^{n^c}] \end{aligned}$$

 $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXP \subseteq NEXP$

Certificate Characterization of NP

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ a binary relation on strings.

- *R* is called **polynomially decidable** if there is a DTM deciding the language $\{x; y \mid (x, y) \in R\}$ in polynomial time.
- *R* is called **polynomially balanced** if $(x, y) \in R$ implies $|y| \le |x|^k$, for some $k \ge 1$.

Theorem

Let $L \subseteq \Sigma^*$ be a language. $L \in \mathbf{NP}$ if and only if there is a polynomially decidable and polynomially balanced relation R, such that:

$$L = \{x \mid \exists y \, R(x, y)\}$$

- This y is called **succinct certificate**, or **witness**.
- So, an **NP Search Problem** is the problem of *computing* witnesses.

Complexity Classes

Oracles & The Polynomial Hierarchy

Certificates & Quantifiers

Can creativity be automated?

As we saw:

- Class **P**: Efficient Computation
- Class NP: Efficient Verification
- So, if we can efficiently verify a mathematical proof, can we create it efficiently?

If P = NP...

- For every mathematical statement, and given a page limit, we would (quickly) generate a proof, if one exists.
- Given detailed constraints on an engineering task, we would (quickly) generate a design which meets the given criteria, if one exists.
- Given data on some phenomenon and modeling restrictions, we would (quickly) generate a theory to explain the data, if one exists.

Proof:

Certificates & Quantifiers

See Pr.9.1 (p.181) in [1]

 (\Leftarrow) If such an R exists, we can construct the following NTM deciding L:

"On input x, guess a y, such that $|y| \le |x|^k$, and then test (in poly-time) if $(x, y) \in R$. If so, accept, else reject." Observe that an accepting computation exists if and only if $x \in L$.

(⇒) If $L \in \mathbf{NP}$, then \exists an NTM N that decides L in time $|x|^k$, for some k. Define the following R:

" $(x, y) \in R$ if and only if y is an **encoding** of an accepting computation of N(x)."

R is polynomially <u>balanced</u> and <u>decidable</u> (*why?*), so, given by assumption that *N* decides *L*, we have our conclusion. \Box

5

Complexity Classes

Oracles & The Polynomial Hierarchy

Certificates & Quantifiers

Complementary complexity classes

- Deterministic complexity classes are in general closed under complement (coL = L, coP = P, coPSPACE = PSPACE).
- Complementaries of non-deterministic complexity classes are very interesting:
- The class coNP contains all the languages that have succinct disqualifications (the analogue of succinct certificate for the class NP). The "no" instance of a problem in coNP has a short proof of its being a "no" instance.
- So:



• Note the *similarity* and the *difference* with $\mathbf{R} = \mathbf{RE} \cap co\mathbf{RE}$.

Space Computation

Quantifier Characterization of Complexity Classes

Definition

We denote as $C = (Q_1/Q_2)$, where $Q_1, Q_2 \in \{\exists, \forall\}$, the class C of languages L satisfying:

- $\bullet x \in L \Rightarrow Q_1 y R(x, y)$
- $\bullet x \notin L \Rightarrow Q_2 y \neg R(x, y)$
- \bullet **P** = (\forall/\forall)
- NP = (\exists/\forall)
- $coNP = (\forall/\exists)$

Complexity Classes Oracles &

Oracles & The Polynomial Hierarchy

Space Computation

- We generate all nodes *u* one after the other, *reusing* space.
- The algorithm has recursion depth of $\lceil \log n \rceil$.
- For each recursion level, we have to store s, t, k and u, that is, $\mathcal{O}(\log n)$ space.
- Thus, the total space used is $\mathcal{O}(\log^2 n)$. \square

Savitch's Theorem

• REACHABILITY \in **NL**.

See Ex.2.10 (p.48) in [1]

Theorem (Savitch's Theorem)

REACHABILITY \in **DSPACE**[log² n]

Proof:

See Th.7.4 (p.149) in [1]

REACH(x, y, i): "There is a path from x to y, of length $\leq i$ ".

- We can solve REACHABILITY if we can compute REACH(x, y, n), for any nodes $x, y \in V$, since any path in G can be at most n long.
- If i = 1, we can check whether REACH(x, y, i).
- If i > 1, we use recursion:

- 4

Complexity Classes

Oracles & The Polynomial Hierarchy

Space Computation

Savitch's Theorem

Corollary

NSPACE[S(n)] \subseteq **DSPACE**[$S^2(n)$], for any space-constructible function $S(n) \ge \log n$.

Proof:

- Let M be the nondeterministic TM to be simulated.
- We run the algorithm of Savitch's Theorem proof on the configuration graph of *M* on input *x*.
- Since the configuration graph has $c^{S(n)}$ nodes, $\mathcal{O}\left(S^2(n)\right)$ space suffices. \square

Corollary

PSPACE = NPSPACE

NL-Completeness

- In Complexity Theory, we "connect" problems in a complexity class with partial ordering relations, called **reductions**, which formalize the notion of "a problem that is at least as hard as another".
- A reduction must be computationally weaker than the class in which we use it.

Definition

A language L_1 is **logspace reducible** to a language L_2 , denoted $L_1 \leq_m^\ell L_2$, if there is a function $f: \Sigma^* \to \Sigma^*$, computable by a DTM in $\mathcal{O}(\log n)$ space, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

We say that a language *A* is **NL**-complete if it is in **NL** and for every $B \in \mathbf{NL}$, $B \leq_m^{\ell} A$.

Complexity Classes

Oracles & The Polynomial Hierarchy

Space Computation

Certificate Definition of NL

- We want to give a characterization of **NL**, similar to the one we gave for **NP**.
- A certificate may be polynomially long, so a logspace machine may not have the space to store it.
- So, we will assume that the certificate is provided to the machine on a separate tape that is **read once**.

NL-Completeness

Theorem

Space Computation

REACHABILITY is **NL**-complete.

Proof:

See Th.4.18 (p.89) in [2]

- We 've argued why REACHABILITY \in **NL**.
- Let $L \in \mathbf{NL}$, that is, it is decided by a $\mathcal{O}(\log n)$ NTM N.
- Given input x, we can construct the *configuration graph* of N(x).
- We can assume that this graph has a *single* accepting node.
- We can construct this in logspace: Given configurations C, C' we can in space $\mathcal{O}(|C| + |C'|) = \mathcal{O}(\log |x|)$ check the graph's adjacency matrix if they are connected by an edge.
- It is clear that $x \in L$ if and only if the produced instance of REACHABILITY has a "yes" answer. \square

58

Complexity Classes

Oracles & The Polynomial Hierarchy

Space Computation

Certificate Definition of NL

Definition

A language L is in **NL** if there exists a deterministic TM M with an additional special read-once input tape, such that for every $x \in \Sigma^*$:

$$x \in L \Leftrightarrow \exists y, |y| \in poly(|x|), M(x, y) = 1$$

where by M(x, y) we denote the output of M where x is placed on its input tape, and y is placed on its special read-once tape, and M uses at most $\mathcal{O}(\log |x|)$ space on its read-write tapes for every input x.

• What if remove the read-once restriction and allow the TM's head to move back and forth on the certificate, and read each bit multiple times?

Immerman-Szelepscényi

Theorem (The Immerman-Szelepscényi Theorem)

 $\overline{REACHABILITY} \in \mathbf{NL}$

Proof: See Th.4.20 (p.91) in [2]

• It suffices to show a $\mathcal{O}(\log n)$ verification algorithm A such that: $\forall (G, s, t), \exists$ a polynomial certificate u such that: A((G, s, t), u) = "yes" iff t is not reachable from s.

- A has read-once access to u.
- G's vertices are identified by numbers in $\{1, \dots, n\} = [n]$
- C_i : "The set of vertices reachable from s in $\leq i$ steps."
- Membership in C_i is easily certified:
- $\forall i \in [n]: v_0, \ldots, v_k$ along the path from s to $v, k \leq i$.
- The certificate is at most polynomial in n.

Oracles & The Polynomial Hierarchy

Complexity Classes

Space Computation

The Immerman-Szelepscényi Theorem

Proof (cont'd):

- Since the verifier has been convinced of $|C_n|$, we can use the 1st type of certificate to convince the verifier that $t \notin C_n$.
- Certifying that $v \notin C_i$, given $|C_i|$ The certificate is the list of certificates that $u \in C_i$, for every $u \in C_i$.

The verifier will check:

- Each certificate is valid
- 2 Vertex u, given a certificate for u, is larger than the previous.
- \bigcirc No certificate is provided for v.
- 4 The total number of certificates is exactly $|C_i|$.

Complexity Classes

The Immerman-Szelepscényi Theorem

Proof (cont'd):

- We can check the certificate using read-once access:
 - $v_0 = s$
 - ② for j > 0, $(v_{j-1}, v_j) \in E(G)$
 - $v_k = v$
 - 4 Path ends within at most *i* steps
- We now construct two types of certificates:
 - ① A certificate that a vertex $v \notin C_i$, given $|C_i|$.
 - 2 A certificate that $|C_i| = c$, for some c, given $|C_{i-1}|$.
- Since $C_0 = \{s\}$, we can provide the 2nd certificate to convince the verifier for the sizes of C_1, \ldots, C_n
- C_n is the set of vertices *reachable* from s.

6

Complexity Classes

Oracles & The Polynomial Hierarchy

Space Computation

The Immerman-Szelepscényi Theorem

Proof (*cont'd*):

Certifying that $v \notin C_i$, given $|C_{i-1}|$

The certificate is the list of certificates that $u \in C_{i-1}$, for every $u \in C_{i-1}$. The verifier will check:

- Each certificate is valid
- 2 Vertex u, given a certificate for u, is larger than the previous.
- 3 No certificate is provided for v or for a neighbour of v.
- 4 The total number of certificates is exactly $|C_{i-1}|$.

Certifying that $|C_i| = c$, given $|C_{i-1}|$

The certificate will consist of n certificates, for vertices 1 to n, in ascending order.

The verifier will check all certificates, and count the vertices that have been certified to be in C_i . If $|C_i| = c$, it accepts. \square

The Immerman-Szelepscényi Theorem

Corollary

For every space constructible $S(n) > \log n$:

$$NSPACE[S(n)] = coNSPACE[S(n)]$$

Proof:

- Let $L \in \mathbf{NSPACE}[S(n)]$. We will show that $\exists S(n)$ space-bounded NTM \overline{M} deciding \overline{L} :
- \overline{M} on input x uses the above certification procedure on the *configuration graph* of M. \square

Corollary

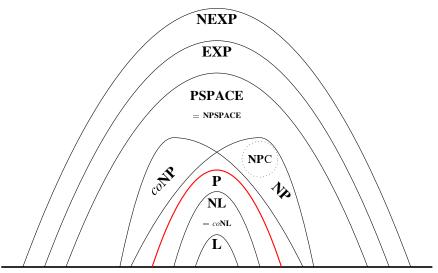
$$NL = coNL$$

Complexity Classes

Oracles & The Polynomial Hierarchy

Space Computation

Our Complexity Hierarchy Landscape



What about Undirected Reachability?

- UNDIRECTED REACHABILITY captures the phenomenon of configuration graphs with both directions.
- H. Lewis and C. Papadimitriou defined the class SL (Symmetric Logspace) as the class of languages decided by a Symmetric Turing Machine using logarithmic space.
- Obviously,

$$L\subseteq SL\subseteq NL$$

- As in the case of **NL**, UNDIRECTED REACHABILITY is **SL**-complete.
- But in 2004, Omer Reingold showed, using expander graphs, a deterministic logspace algorithm for UNDIRECTED REACHABILITY, so:

Theorem (Reingold, 2004)

L = SL

Oracles & The Polynomial Hierarchy

Complexity Classes

0000

Reductions & Completeness

Karp Reductions

Definition

A language L_1 is **Karp reducible** to a language L_2 , denoted by $L_1 \leq_m^p L_2$, if there is a function $f: \Sigma^* \to \Sigma^*$, computable by a polynomial-time DTM, such that for all $x \in \Sigma^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Definition

Let C be a complexity class.

- We say that a language *A* is C-hard (or \leq_m^p -hard for C) if for every $B \in C$, $B \leq_m^p A$.
- We say that a language A is C-complete, if it is C-hard, and also $A \in C$.

Karp reductions vs logspace redutions

Theorem

A logspace reduction is a polynomial-time reduction.

Proof:

See Th.8.1 (p.160) in [1]

- Let *M* the logspace reduction TM.
- *M* has $2^{\mathcal{O}(\log n)}$ possible configurations.
- The machine is deterministic, so no configuration can be repeated in the computation.
- So, the computation takes $\mathcal{O}(n^k)$ time, for some k.

Complexity Classes Oracles & The Polynomial Hierarchy

Reductions & Completeness

Circuits and CVP

Definition (CVP)

Circuit Value Problem (CVP): Given a circuit C and an assignment x to its variables, determine whether C(x) = 1.

• CVP \in **P**.

Example

REACHABILITY \leq_m^{ℓ} CVP: Graph $G \to \operatorname{circuit} R(G)$:

- The gates are of the form:
 - $g_{i,i,k}$, $1 \le i, j \le n$, $0 \le k \le n$.
 - $h_{i,i,k}, 1 \le i, j, k \le n$
- $g_{i,i,k}$ is **true** iff there is a path from i to j without intermediate nodes bigger than k.
- $h_{i,j,k}$ is **true** iff there is a path from i to j without intermediate nodes bigger than k, and k is used.

Circuits and CVP

Reductions & Completeness

Definition (Boolean circuits)

For every $n \in \mathbb{N}$ an *n*-input, single output Boolean Circuit *C* is a directed acyclic graph with n sources and one sink.

- \bullet All nonsource vertices are called *gates* and are labeled with one of \land (and), \vee (or) or \neg (not).
- The vertices labeled with \wedge and \vee have fan-in (i.e. number or incoming edges) 2.
- The vertices labeled with \neg have fan-in 1.
- \bullet For every vertex v of C, we assign a value as follows: for some input $x \in \{0,1\}^n$, if v is the i-th input vertex then $val(v) = x_i$, and otherwise val(v) is defined recursively by applying v's logical operation on the values of the vertices connected to v.
- The *output* C(x) is the value of the output vertex.

Complexity Classes Oracles & The Polynomial Hierarchy

Reductions & Completeness

Circuits and CVP

Example

- Input gates: $g_{i,i,0}$ is **true** iff $(i = j \text{ or } (i,j) \in E(G))$.
- For k = 1, ..., n: $h_{i,i,k} = (g_{i,k,k-1} \land g_{k,i,k-1})$
- For k = 1, ..., n: $g_{i,i,k} = (g_{i,i,k-1} \vee h_{i,i,k})$
- The output gate $g_{1,n,n}$ is **true** iff there is a path from 1 to n using no intermediate paths above n (sic).
- We also can compute the reduction in logspace: go over all possible i, j, k's and output the appropriate edges and sorts for the variables $(1, ..., 2n^3 + n^2)$.

Reductions & Completeness

Composing Reductions

Theorem

Reductions & Completeness

If $L_1 \leq_m^{\ell} L_2$ and $L_2 \leq_m^{\ell} L_3$, then $L_1 \leq_m^{\ell} L_3$.

Proof:

See Prop.8.2 (p.164) in [1]

- \bullet Let R, R' be the aforementioned reductions.
- We have to prove that R'(R(x)) is a logspace reduction.
- But R(x) may by longer than $\log |x|$...
- We simulate $M_{R'}$ by remembering the head position *i* of the input string of $M_{R'}$, i.e. the output string of M_R .
- If the head moves to the right, we increment i and simulate M_R long enough to take the ith bit of the output.
- If the head stays in the same position, we just remember the i^{th} bit.
- If the head moves to the left, we decrement i and **start** M_R **from the beggining**, until we reach the desired bit.

Complexity Classes

Oracles & The Polynomial Hierarchy

Reductions & Completeness

P-Completeness

Theorem

If two classes C and C' are both closed under reductions and there is an $L \subseteq \Sigma^*$ complete for both C and C', then C = C'.

- Consider the **Computation Table** T of a poly-time TM M(x): T_{ij} represents the contents of tape position j at step i.
- But how to remember the head position and state? At the i^{th} step: if the state is q and the head is in position j, then $T_{ij} \in \Sigma \times Q$.
- We say that the table is **accepting** if $T_{|x|^k-1,j} \in (\Sigma \times \{q_{yes}\})$, for some i.
- Observe that T_{ij} depends only on the contents of the **same** of **adjacent** positions at time i 1.

Closure under reductions

- Complete problems are the maximal elements of the reductions partial ordering.
- Complete problems capture the essence and difficulty of a complexity class.

Definition

A class C is **closed under reductions** if for all $A, B \subseteq \Sigma^*$: If A < B and $B \in C$, then $A \in C$.

- P, NP, coNP, L, NL, PSPACE, EXP are closed under Karp and logspace reductions.
- If an NP-complete language is in P, then P = NP.
- If L is **NP**-complete, then \bar{L} is co**NP**-complete.
- If a coNP-complete problem is in NP, then NP = coNP.

Complexity Classes

Oracles & The Polynomial Hierarchy

Reductions & Completeness

P-Completeness

Theorem

CVP is **P**-complete.

Proof:

See Th. 8.1 (p.168) in [1]

- We have to show that for any $L \in \mathbf{P}$ there is a reduction R from L to CVP.
- R(x) must be a variable-free circuit such that $x \in L \Leftrightarrow R(x) = 1$.
- T_{ij} depends only on $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$.
- Let $\Gamma = \Sigma \cup (\Sigma \times Q)$.
- Encode $s \in \Gamma$ as (s_1, \ldots, s_m) , where $m = \lceil \log |\Gamma| \rceil$.
- Then the computation table can be seen as a table of binary entries $S_{ii\ell}$, $1 < \ell < m$.
- $S_{ij\ell}$ depends only on the 3m entries $S_{i-1,j-1,\ell'}, S_{i-1,j,\ell'}, S_{i-1,j+1,\ell'}$, where $1 < \ell' < m$.

P-Completeness

Proof (cont'd):

• So, there are m Boolean Functions $f_1, \ldots, f_m : \{0, 1\}^{3m} \to \{0, 1\}$ s.t.;

$$S_{ij\ell} = f_{\ell}(\overrightarrow{S}_{i-1,j-1}, \overrightarrow{S}_{i-1,j}, \overrightarrow{S}_{i-1,j+1})$$

- Thus, there exists a Boolean Circuit C with 3m inputs and m outputs computing T_{ii} .
- C depends only on M, and has constant size.

- R(x) will be $(|x|^k 1) \times (|x|^k 2)$ copies of C.
- The input gates are fixed.
- R(x)'s output gate will be the first bit of $C_{|x|^k-1,1}$.
- The circuit C is fixed, so we can generate indexed copies of C, using $\mathcal{O}(\log |x|)$ space for indexing.

Complexity Classes Oracles & The Polynomial Hierarchy

Reductions & Completeness

CIRCUIT SAT & SAT

Example

- Gate g of $C \rightarrow$ clauses in R(C):
 - g variable gate: add $(\neg g \lor x) \land (g \lor \neg x)$ $\equiv g \Leftrightarrow x$
 - g TRUE gate: add (g)
 - g **FALSE** gate: add $(\neg g)$
 - g **NOT** gate & pred(g) = h: add $(\neg g \lor \neg h) \land (g \lor h)$
 - g **OR** gate & $pred(g) = \{h, h'\}$: add

 $(\neg h \lor g) \land (\neg h' \lor g) \land (h \lor h' \lor \neg g)$ $\equiv g \Leftrightarrow (h \vee h')$

- g **AND** gate & $pred(g) = \{h, h'\}$: add
 - $(\neg g \lor h) \land (\neg g \lor h') \land (\neg h \lor \neg h' \lor g)$ $\equiv g \Leftrightarrow (h \wedge h')$
- g output gate: add (g)
- \circ R(C) is satisfiable if and only if C is.
- The construction can be done within $\log |x|$ space.

CIRCUIT SAT & SAT

Definition (CIRCUIT SAT)

Given Boolen Circuit C, is there a truth assignment x appropriate to C, such that C(x) = 1?

Definition (SAT)

Given a Boolean Expression ϕ in CNF, is it satisfiable?

Example

Reductions & Completeness

CIRCUIT SAT \leq_m^{ℓ} SAT:

- Given $C \to \text{Boolean Formula } R(C)$, s.t. $C(x) = 1 \Leftrightarrow R(C)(x) = T$.
- Variables of $C \rightarrow$ variables of R(C).
- Gate g of $C \rightarrow \text{variable } g \text{ of } R(C)$.

Complexity Classes

Oracles & The Polynomial Hierarchy

Reductions & Completeness

Bounded Halting Problem

• We can define the time-bounded analogue of HP:

Definition (Bounded Halting Problem (BHP))

Given the code $\lfloor M \rfloor$ of an NTM M, and input x and a string 0^t , decide if M accepts x in t steps.

Theorem

BHP is NP-complete.

Proof:

- \bullet BHP \in **NP**.
- Let $A \in \mathbf{NP}$. Then, $\exists NTM M$ deciding A in time p(|x|), for some $p \in poly(|x|)$.
- The reduction is the function $R(x) = \langle \bot M \bot, x, 0^{p(|x|)} \rangle$.

Cook's Theorem

Theorem (Cook's Theorem) *SAT is* **NP**-complete.

Proof:

See Th.8.2 (p.171) in [1]

- SAT \in NP.
- Let $L \in \mathbf{NP}$. We will show that $L \leq_m^{\ell} \mathbf{CIRCUIT} \mathbf{SAT} \leq_m^{\ell} \mathbf{SAT}$.
- Since $L \in \mathbf{NP}$, there exists an NPTM M deciding L in n^k steps.
- Let $(c_1, \ldots, c_{n^k}) \in \{0, 1\}^{n^k}$ a certificate for M (recall the binary encoding of the computation tree).

81

Oracles & The Polynomial Hierarchy

Complexity Classes

Reductions & Completeness

NP-completeness: Web of Reductions

- Many NP-complete problems stem from Cook's Theorem via reductions:
 - 3SAT, MAX2SAT, NAESAT
 - IS, CLIQUE, VERTEX COVER, MAX CUT
 - $TSP_{(D)}$, 3COL
 - SET COVER, PARTITION, KNAPSACK, BIN PACKING
 - INTEGER PROGRAMMING (IP): Given m inequalities oven n variables $u_i \in \{0, 1\}$, is there an assignment satisfying all the inequalities?
- Always remember that these are **decision versions** of the corresponding **optimization problems**.
- But 2SAT, 2COL \in **P**.

Cook's Theorem

Reductions & Completeness

Proof (cont'd):

See Th. 8.2 (p. 171) in [1]

- If we fix a certificate, then the computation is *deterministic* (the language's Verifier V(x, y) is a DPTM).
- So, we can define the **computation table** $T(M, x, \overrightarrow{c})$.
- As before, all non-top row and non-extreme column cells T_{ij} will depend *only* on $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$ and the nondeterministic choice c_{i-1} .
- We now fixed a circuit C with 3m + 1 input gates.
- Thus, we can construct in $\log |x|$ space a circuit R(x) with variable gates $c_1, \ldots c_{n^k}$ corresponding to the **nondeterministic choices** of the machine.
- R(x) is satisfiable if and only if $x \in L$.

82

Complexity Classes

Oracles & The Polynomial Hierarchy

Reductions & Completeness

NP-completeness: Web of Reductions

Example

SAT \leq_m^{ℓ} IP:

• Every clause can be expressed as an inequality, eg:

$$(x_1 \lor \bar{x}_2 \lor \bar{x}_3) \longrightarrow x_1 + (1 - x_2) + (1 - x_3) \ge 1$$

- This method is generalized by the notion of *Constraint Satisfaction Problems*.
- A Constraint Satisfaction Problem (CSP) generalizes SAT by allowing clauses of arbitrary form (instead of ORs of literals).

3SAT is the subcase of qCSP, where arity q=3 and the constraints are ORs of the involved literals.

Reductions & Completeness

Quantified Boolean Formulas

Definition (Quantified Boolean Formula)

A **Quantified Boolean Formula** *F* is a formula of the form:

$$F = \exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \ \phi(x_1, \dots, x_n)$$

where ϕ is *plain* (quantifier-free) boolean formula.

• Let TQBF the language of all true QBFs.

Example

$$F = \exists x_1 \forall x_2 \exists x_3 \left[(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \right]$$

The above is a True QBF ((1, 0, 0)) and (1, 1, 1) satisfy it).

Oracles & The Polynomial Hierarchy

Reductions & Completeness

Ouantified Boolean Formulas

Proof (cont'd):

See Th. 19.1 (p.456) in [1] - Th.4.13 (p.84) in [2]

- Now, let M a TM with space bound p(n).
- We can create the configuration graph of M(x), having size $\mathcal{O}(p(n))$
- M accepts x iff there is a path of length at most $2^{\mathcal{O}(p(n))}$ from the initial to the accepting configuration.
- Using Savitch's Theorem idea, for two configurations C and C' we have:

$$REACH(C, C', 2^i) \Leftrightarrow \Leftrightarrow \exists C'' \left[REACH(C, C'', 2^{i-1}) \land REACH(C'', C', 2^{i-1}) \right]$$

- But, this is a bad idea: Doubles the size each time.
- Instead, we use additional variables: $\exists C'' \forall D_1 \forall D_2 \left[(D_1 = C \land D_2 = C'') \lor (D_1 = C'' \land D_2 = C') \right] \Rightarrow$ $REACH(D_1, D_2, 2^{i-1})$

Complexity Classes

Reductions & Completeness

Quantified Boolean Formulas

Theorem

TQBF is **PSPACE**-complete.

Proof:

See Th. 19.1 (p.456) in [1] - Th.4.13 (p.84) in [2]

- TOBF \in **PSPACE**:
 - Let ϕ be a QBF, with n variables and length m.
 - Recursive algorithm $A(\phi)$:
 - If n = 0, then there are only constants, hence $\mathcal{O}(m)$ time/space.
 - If n > 0:

$$A(\phi) = A(\phi|_{x_1=0}) \lor A(\phi|_{x_1=1}), \text{ if } Q_1 = \exists, \text{ and } A(\phi) = A(\phi|_{x_1=0}) \land A(\phi|_{x_1=1}), \text{ if } Q_1 = \forall.$$

- Both recursive computations can be run on the same space.
- So $space_{n,m} = space_{n-1,m} + \mathcal{O}(m) \Rightarrow space_{n,m} = \mathcal{O}(n \cdot m)$.

Complexity Classes Oracles & The Polynomial Hierarchy

Oracles & The Polynomial Hierarchy

Reductions & Completeness

Quantified Boolean Formulas

Proof (cont'd):

See Th. 19.1 (p.456) in [1] - Th.4.13 (p.84) in [2]

- The base case of the recursion is $C_1 \to C_2$, and can be encoded as a quantifier-free formula.
- The size of the formula in the i^{th} step is

$$s_i \le s_{i-1} + \mathcal{O}(p(n)) \Rightarrow \mathcal{O}(p^2(n)).$$

*Logical Characterizations

• **Descriptive complexity** is a branch of computational complexity theory and of finite model theory that characterizes complexity classes by the *type of logic* needed to express the languages in them.

Theorem (Fagin's Theorem)

The set of all properties expressible in Existential Second-Order Logic is precisely **NP**.

Theorem

The class of all properties expressible in Horn Existential Second-Order Logic with Successor is precisely **P**.

• HORNSAT is **P**-complete.

Complexity Classes

Oracles & The Polynomial Hierarchy

Reductions & Completeness

What have we learned?

- Reductions relate problems with respect to hardness.
- Complete problems reflect the difficulty of the class.
- REACHABILITY is **NL**-complete.
- Immerman-Szelepscényi's Theorem implies that NL = coNL.
- Circuit Value Problem (CVP) is **P**-complete under logspace reductions.
- CIRCUIT SAT and SAT are **NP**-complete.
- True Quantified Boolean Formula (TQBF) is **PSPACE**-complete.

What have we learned?

- We define complexity classes using a computation model/mode and complexity measures.
- Time/Space constructible functions are used as complexity measures.
- Classes of the same kind form proper hierarchies.
- **NP** is the class of *easily verifiable* problems: given a *certificate*, one can efficiently verify that it is correct.
- Savitch's Theorem implies that **PSPACE** = **NPSPACE**.

Complexity Classes

Oracles & The Polynomial Hierarchy

Contents

- Introduction
- Turing Machines
- Undecidability
- Complexity Classes
- Oracles & The Polynomial Hierarchy
- The Structure of NP
- Randomized Computation
- Non-Uniform Complexity
- Interactive Proofs
- Inapproximability
- Derandomization of Complexity Classes
- Counting Complexity
- Epilogue

Oracle TMs and Oracle Classes

Definition

Complexity Classes

A Turing Machine $M^{?}$ with *oracle* is a multi-string deterministic TM that has a special string, called **query string**, and three special states: q_7 (query state), and q_{YES} , q_{NO} (answer states). Let $A \subseteq \Sigma^*$ be an arbitrary language. The computation of oracle machine M^A proceeds like an ordinary TM except for transitions from the query state: From the q_2 moves to either q_{YES} , q_{NO} , depending on whether the current query string is in A or not.

- The answer states allow the machine to use this answer to its further computation.
- The computation of $M^{?}$ with oracle A on iput x is denoted as $M^A(x)$.

Complexity Classes

Oracles & The Polynomial Hierarchy

Oracle Classes

Oracle TMs and Oracle Classes

Theorem

There exists an oracle B for which $\mathbf{P}^B \neq \mathbf{N}\mathbf{P}^B$.

Proof:

Th.14.5, p.340-342 in [1]

- We will find a language $L \in \mathbf{NP}^B \setminus \mathbf{P}^B$.
- Let $L = \{1^n \mid \exists x \in B \text{ with } |x| = n\}.$
- $L \in \mathbf{NP}^B$ (why?)
- We will define the oracle $B \subseteq \{0,1\}^*$ such that $L \notin \mathbf{P}^B$:
- Let M_1^2, M_2^2, \ldots an enumeration of all PDTMs with oracle, such that every machine appears *infinitely many* times in the enumeration.
- We will define *B* iteratively: $B_0 = \emptyset$, and $B = \bigcup_{i>0} B_i$.
- In i^{th} stage, we have defined B_{i-1} , the set of all strings in B with length < i.
- Let also X the set of **exceptions**.

Oracle TMs and Oracle Classes

Definition

Complexity Classes

Let C be a time complexity class (deterministic or nondeterministic). Define C^A to be the *class* of all languages decided by machines of the same sort and time bound as in C, only that the machines have now oracle access to A. Also, we define: $C_1^{C_2} = \bigcup_{L \in C_2} C_1^L$.

For example, $\mathbf{P}^{NP} = \bigcup_{L \in \mathbf{NP}} \mathbf{P}^{L}$. Note that $\mathbf{P}^{SAT} = \mathbf{P}^{NP}$.

Theorem

There exists an oracle A for which $\mathbf{P}^A = \mathbf{NP}^A$.

Proof:

Th.14.4, p.340 in [1]

Take *A* to be a **PSPACE**-complete language. Then:

$$\mathbf{PSPACE} \subseteq \mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A = \mathbf{PSPACE}^{\mathbf{PSPACE}} \subseteq \mathbf{PSPACE}. \quad \Box$$

Complexity Classes

Oracles & The Polynomial Hierarchy

Oracle Classes

Proof (cont'd):

- We simulate $M_i^B(1^i)$ for $i^{\log i}$ steps.
- How do we answer the oracle questions "Is $x \in B$ "?
- If |x| < i, we look for x in B_{i-1} .
- $\bullet \to \mathbf{If} \ x \in B_{i-1}, M_i^B \ \text{goes to} \ q_{YES}$ \rightarrow **Else** M_i^B goes to q_{NO}
- If |x| > i, M_i^B goes to q_{NO} , and $x \to X$.
- Suppose that after at most $i^{\log i}$ steps the machine *rejects*.
 - Then we define $B_i = B_{i-1} \cup \{x \in \{0,1\}^* : |x| = i, x \notin X\}$ so $1^i \in L$, and $L(M_i^B) \neq L$.
 - Why $\{x \in \{0,1\}^* : |x| = i, x \notin X\} \neq \emptyset$?
- If the machine *accepts*, we define $B_i = B_{i-1}$, so that $1^i \notin L$.
- If the machine fails to halt in the allotted time, we set $B_i = B_{i-1}$, but we know that the same machine will appear in the enumeration with an index sufficiently large.

Complexity Classes

The Limits of Diagonalization

• As we saw, an oracle can transfer us to an alternative computational "universe".

(We saw a universe where P = NP, and another where $P \neq NP$)

- Diagonalization is a technique that relies in the facts that:
 - TMs are (effectively) represented by strings.
 - A TM can simulate another without much overhead in time/space.
- So, diagonalization or any other proof technique relies only on these two facts, holds also for *every* oracle.
- Such results are called **relativizing results**. E.g., $\mathbf{P}^A \subset \mathbf{NP}^A$, for every $A \in \{0, 1\}^*$.
- The above two theorems indicate that P vs. NP is a nonrelativizing result, so diagonalization and any other relativizing method doesn't suffice to prove it.

Complexity Classes

Oracles & The Polynomial Hierarchy

Oracle Classes

*Random Oracles

• We proved that:

$$\bullet \exists A \subseteq \Sigma^* : \mathbf{P}^A = \mathbf{N}\mathbf{P}^A
\bullet \exists B \subset \Sigma^* : \mathbf{P}^B \neq \mathbf{N}\mathbf{P}^B$$

- What if we chose the oracle language at random?
- Now, consider the set $\mathcal{U} = Pow(\Sigma^*)$, and the sets:

$$\{A \in \mathcal{U}: \mathbf{P}^A = \mathbf{NP}^A\}$$

$$\{B \in \mathcal{U}: \mathbf{P}^B \neq \mathbf{NP}^B\}$$

• Can we compare these two sets, and find which is *larger*?

Theorem (Bennet, Gill)

$$\mathbf{Pr}_{B\subset\Sigma^*}\left[\mathbf{P}^B\neq\mathbf{NP}^B\right]=1$$

Cook Reductions

- A problem *A* is **Cook-Reducible** to a problem *B*, denoted by $A \leq_T^p B$, if there is an oracle DTM M^B which in polynomial time decides *A* (making at most polynomial many queries to *B*).
- That is: $A \in \mathbf{P}^B$

$$\bullet \ A \leq^p_m B \Rightarrow A \leq^p_T B$$

$$\bullet \ \overline{A} \leq^p_T A$$

Theorem

P, **PSPACE** are closed under \leq_T^p .

• Is **NP** closed under \leq_T^p ?

(cf. Problem Sets!)

Oracles & The Polynomial Hierarchy

Complexity Classes

The Polynomial Hierarchy

Polynomial Hierarchy Definition

$$\bullet \ \Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathbf{P}$$

$$\bullet \ \Delta_{i+1}^p = \mathbf{P}^{\Sigma_i^p}$$

$$\bullet \ \Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$$

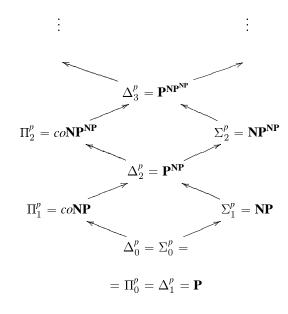
$$\bullet \ \Pi_{i+1}^p = co\mathbf{NP}^{\Sigma_i^p}$$

$$\mathbf{PH} \equiv \bigcup_{i \geqslant 0} \Sigma_i^p$$

$$\Sigma_0^p = \mathbf{P}$$

•
$$\Delta_1^p = \mathbf{P}, \Sigma_1^p = \mathbf{NP}, \Pi_1^p = co\mathbf{NP}$$

The Polynomial Hierarchy



- $\Sigma_i^p, \Pi_i^p \subseteq \Sigma_{i+1}^p$
- $A, B \in \Sigma_i^p \Rightarrow A \cup B \in \Sigma_i^p, A \cap B \in \Sigma_i^p$
- $\frac{A}{A} \in \Pi_i^p \Rightarrow$
- $A, B \in \Delta_i^p \Rightarrow$ $A \cup B, A \cap B \text{ and}$ $\overline{A} \in \Delta_i^p$

10.1

Oracles & The Polynomial Hierarchy

Complexity Classes

Proof (cont.):

If $L \in \Sigma_i^p$, we must show the existence or R:

- $L \in \Sigma_i^p \Rightarrow \exists \text{ NTM } M^K, K \in \Sigma_{i-1}^p$, which decides L.
- $K \in \Sigma_{i-1}^p \Rightarrow \exists S \in \Pi_{i-2}^p : (z \in K \Leftrightarrow \exists w : (z, w) \in S).$
- We must describe a relation R (we know: $x \in L \Leftrightarrow$ accepting computation of $M^K(x)$)
- Query Steps: "yes" $\rightarrow z_i$ has a certificate w_i st $(z_i, w_i) \in S$.
- So, $R(x) = "(x, y) \in R$ iff yrecords an accepting computation of M? on x, together with a certificate w_i for each **yes** query z_i in the computation."
- We must show $\{x; y : (x, y) \in R\} \in \Pi_{i-1}^p$:
 - Check that all steps of $M^{?}$ are legal (poly time).
 - Check that $(z_i, w_i) \in S$ (in Π_{i-2}^p , and thus in Π_{i-1}^p).
 - For all "no" queries z'_i , check $z'_i \notin K$ (another $\Pi_{i-1}^{p^-}$).

Theorem

Let L be a language , and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced relation R such that the language $\{x;y:(x,y)\in R\}$ is in Π_{i-1}^p and

$$L = \{x : \exists y, s.t. : (x, y) \in R\}$$

Proof (by Induction):

Th.17.8, p.425-526 in [1]

• For
$$i = 1$$
:
 $\{x; y : (x, y) \in R\} \in \mathbf{P}$, so $L = \{x | \exists y : (x, y) \in R\} \in \mathbf{NP} \checkmark$

• (For i > 1:)
If $\exists R \in \Pi_{i-1}^p$, we must show that $L \in \Sigma_i^p \Rightarrow \exists$ NTM with Σ_{i-1}^p oracle: NTM(x) guesses a y and asks Π_{i-1}^p oracle whether $(x, y) \notin R$.

10

Complexity Classes

Oracles & The Polynomial Hierarchy

Main Theorems

Corollary

Let L be a language , and $i \geq 1$. $L \in \Pi_i^p$ iff there is a polynomially balanced relation R such that the language $\{x;y:(x,y)\in R\}$ is in Σ_{i-1}^p and

$$L = \{x : \forall y, |y| \le |x|^k, s.t. : (x, y) \in R\}$$

Corollary

Let L be a language, and $i \geq 1$. $L \in \Sigma_i^p$ iff there is a polynomially balanced, polynomially-time decicable (i+1)-ary relation R such that:

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 ... Q y_i, s.t. : (x, y_1, ..., y_i) \in R\}$$

where the i^{th} quantifier Q is \forall , if i is even, and \exists , if i is odd.

Main Theorems

Remark

$$\Sigma_{i}^{p} = \underbrace{(\exists \forall \exists \cdots Q_{i}}_{i \text{ quantifiers}} / \underbrace{\forall \exists \forall \cdots Q_{i}'}_{i \text{ quantifiers}}$$

$$\Pi_i^p = \underbrace{(\forall \exists \forall \cdots Q_i}_{i \text{ quantifiers}} / \underbrace{\exists \forall \exists \cdots Q_i'}_{i \text{ quantifiers}})$$

Theorem

If for some $i \ge 1$, $\sum_{i=1}^{p} \Pi_{i}^{p}$, then for all j > i:

$$\Sigma_j^p = \Pi_j^p = \Delta_j^p = \Sigma_i^p$$

Or, the polynomial hierarchy *collapses* to the i^{th} level.

Proof: Th.17.9, p.427 in [1]

- It suffices to show that: $\Sigma_i^p = \Pi_i^p \Rightarrow \Sigma_{i+1}^p = \Sigma_i^p$.
- Let $L \in \Sigma_{i+1}^p \Rightarrow \exists R \in \Pi_i^p : L = \{x | \exists y : (x, y) \in R\}$
- $\bullet \ \Pi_i^p = \Sigma_i^p \Rightarrow R \in \Sigma_i^p$
- $\bullet (x, y) \in R \Leftrightarrow \exists z : (x, y, z) \in S, S \in \Pi_{i-1}^p.$
- So, $x \in L \Leftrightarrow \exists y; z : (x, y, z) \in S, S \in \Pi_{i-1}^{\tilde{p}}$, hence $L \in \Sigma_i^p$.

Complexity Classes

Oracles & The Polynomial Hierarchy

Main Theorems

Theorem

If there is a **PH**-complete problem, then the polynomial hierarchy collapses to some finite level.

Proof: Th.17.11, p.429 in [1]

- Let *L* is **PH**-complete.
- Since $L \in \mathbf{PH}$, $\exists i \geq 0 : L \in \Sigma_i^p$.
- But any $L' \in \sum_{i+1}^p$ reduces to L.
- Since PH is closed under reductions, we imply that $L' \in \Sigma_i^p$, so $\Sigma_i^p = \Sigma_{i+1}^p$.

Theorem

$PH \subseteq PSPACE$

• $PH \stackrel{?}{=} PSPACE$ (Open). If it was, then PH has complete problems, so it collapses to some finite level.

Corollary

Complexity Classes

If **P=NP**, or even **NP**=co**NP**, the Polynomial Hierarchy collapses to the first level.

QSAT, Definition

Given expression ϕ , with Boolean variables partitioned into i sets X_i , is ϕ satisfied by the overall truth assignment of the expression:

$$\exists X_1 \forall X_2 \exists X_3 \dots Q X_i \phi$$

where Q is \exists if *i* is *odd*, and \forall if *i* is even.

Theorem

For all $i \geq 1$ QSAT_i is Σ_i^p -complete.

106

Complexity Classes

Oracles & The Polynomial Hierarchy

Main Theorems

Relativized Results

Let's see how the inclusion of the Polynomial Hierarchy to Polynomial Space, and the inclusions of each level of **PH** to the next relativizes:

- $\mathbf{PH}^A \neq \mathbf{PSPACE}^A$ relative to *some* oracle $A \subseteq \Sigma^*$.
 - (Yao 1985, Håstad 1986)
- $\mathbf{Pr}_A[\mathbf{PH}^A \neq \mathbf{PSPACE}^A] = 1$

(Cai 1986, Babai 1987)

• $(\forall i \in \mathbb{N}) \ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}$ relative to *some* oracle $A \subseteq \Sigma^*$.

(Yao 1985, Håstad 1986)

•
$$\mathbf{Pr}_A[(\forall i \in \mathbb{N}) \ \Sigma_i^{p,A} \subsetneq \Sigma_{i+1}^{p,A}] = 1$$

(Rossman-Servedio-Tan, 2015)

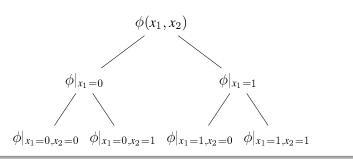
Self-Reducibility of SAT

- For a Boolean formula ϕ with n variables and m clauses.
- It is easy to see that:

$$\phi \in SAT \Leftrightarrow (\phi|_{x_1=0} \in SAT) \lor (\phi|_{x_1=1} \in SAT)$$

- Thus, we can **self-reduce** SAT to instances of smaller size.
- Self-Reducibility Tree of depth *n*:

Example



Complexity Classes

Oracles & The Polynomial Hierarchy

The Complexity of Optimization Problems

What about TSP?

- We can solve TSP using a hypothetical algorithm for the NP-complete decision version of TSP:
- We can find the cost of the optimum tour by **binary search** (in the interval $[0, 2^n]$).
- When we find the optimum cost C, we fix it, and start changing intercity distances one-by one, by setting each distance to C+1.
- We then ask the **NP**-oracle if there still is a tour of optimum cost at most *C*:
 - If there is, then this edge is not in the optimum tour.
 - If there is not, we know that this edge is in the optimum tour.
- After at most n^2 (polynomial) oracle queries, we can extract the optimum tour, and thus have the solution to TSP.

Self-Reducibility of SAT

Definition (FSAT)

The Complexity of Optimization Problems

FSAT: Given a Boolean expression ϕ , if ϕ is satisfiable then return a satisfying truth assignment for ϕ . Otherwise return "no".

- **FP** is the function analogue of **P**: it contains functions computable by a DTM in poly-time.
- FSAT \in **FP** \Rightarrow SAT \in **P**.
- What about the opposite?
- If $SAT \in \mathbf{P}$, we can use the self-reducibility property to fix variables one-by-one, and retrieve a solution.
- \bullet We only need 2n calls to the *alleged* poly-time algorithm for SAT.

Complexity Classes

Oracles & The Polynomial Hierarchy

The Complexity of Optimization Problems

The Classes P^{NP} and FP^{NP}

- **P**^{SAT} is the class of languages decided in pol time with a SAT oracle (*Polynomial number of adaptive queries*).
- SAT is **NP**-complete \Rightarrow **P**^{SAT}=**P**^{NP}.
- **FP**^{NP} is the class of **functions** that can be computed by a poly-time DTM with a SAT oracle.
- FSAT, TSP \in **FP**^{NP}.

Definition (Reductions for Function Problems)

A function problem A reduces to B if there exists $R, S \in \mathbf{FL}$ such that:

- $\bullet x \in A \Rightarrow R(x) \in B.$
- If z is a correct output of R(x), then S(z) is a correct output of x.

Theorem

TSP is **FP^{NP}**-complete.

What have we learned?

- Oracle TMs have one-step oracle access to some language.
- There exist oracles $A, B \subseteq \Sigma^*$ such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{N}\mathbf{P}^B$.
- Relativizing results hold for *every* oracle.
- A Cook reduction $A \leq_T^p B$ is a poly-time TM deciding A, by using B as an oracle.
- The Polynomial Hierarchy can be viewed as:
 - Oracle hierarchy of consecutive **NP** oracles.
 - Quantifier hierarchy of alternating quantifiers.
- If for some $i \ge 1$ $\sum_{i=1}^{p} \prod_{i=1}^{p} \prod_{j=1}^{p} \sum_{i=1}^{p} \prod_{j=1}^{p} \prod_{i=1}^{p} \prod_{j=1}^{p} \prod_{j=1$ problem, then PH collapses to some finite level.
- Optimization problems with decision version in **NP** (such as TSP) are in $\mathbf{FP}^{\mathbf{NP}}$.

The Complexity of Optimization Problems

Complexity Classes

The Complexity Universe

