Theory Thing Episode 5: P vs. NP and the Relativization Barrier

Maruth Goyal

UT Austin

Spring 2020

Table of Contents

Turing Machines

Model of computation introduced by **Alan Turing**.

1. Think of a tape extending infinitely to the right
2. Our machine runs along this tape
3. In one step, it can do the following:
    1. **Read** a symbol from the current position on the tape

Turing Machines

Model of computation introduced by **Alan Turing**.

1. Think of a tape extending infinitely to the right
2. Our machine runs along this tape
3. In one step, it can do the following:
    1. **Read** a symbol from the current position on the tape
    2. **Change its state**

Turing Machines

Model of computation introduced by **Alan Turing**.

1. Think of a tape extending infinitely to the right
2. Our machine runs along this tape
3. In one step, it can do the following:
    1. **Read** a symbol from the current position on the tape
    2. **Change its state**
    3. **Write** a symbol to the current position of the tape

Turing Machines

Model of computation introduced by **Alan Turing**.

1. Think of a tape extending infinitely to the right
2. Our machine runs along this tape
3. In one step, it can do the following:
    1. **Read** a symbol from the current position on the tape
    2. **Change its state**
    3. **Write** a symbol to the current position of the tape
    4. **Move** left,right, or **halt**, or stay in place.

Turing Machines

There exists various other extensions to this model, including but not limited to

1. multiple tapes
2. read/write tapes
3. oblivious TMs

However, for our purposes we will only consider the above simple model.

Turing Machines

Definition: Turing Machines

A Turing Machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

1. Q: set of states

Turing Machines

Definition: Turing Machines

A Turing Machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

1. Q: set of states
2. Σ: set of tape alphabet symbols

Turing Machines

Definition: Turing Machines

A Turing Machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

1. Q: set of states
2. Σ: set of tape alphabet symbols
3. $\delta : (Q \setminus F) \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \to, \text{halt}\}$: transition function

Turing Machines

Definition: Turing Machines

A Turing Machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

1. Q: set of states
2. Σ: set of tape alphabet symbols
3. $\delta : (Q \setminus F) \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow, \text{halt}\}$: transition function
4. q_0: the initial state of the machine

Turing Machines

Definition: Turing Machines

A Turing Machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where

1. Q: set of states
2. Σ: set of tape alphabet symbols
3. $\delta : (Q \setminus F) \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow, \text{halt}\}$: transition function
4. q_0: the initial state of the machine
5. F: set of accepting states

Turing Machines

Definition

A set of words which a given Turing Machine M accepts is called the language of that Turing Machine, $\mathcal{L}(M)$.

Turing Machines

Definition

A set of words which a given Turing Machine M accepts is called the language of that Turing Machine, $\mathcal{L}(M)$.

Remark

A given language L may be represented by any number of Turing Machines, but a given Turing Machine M has a well-defined language.

Time Complexity

Definition

A language $L \in \mathrm{TIME}(f(n))$ iff there exists a Turing Machine M such that M can decide L using $O(f(n))$ operations.

Definition

A language $L \in P$ iff $\exists k$ such that $L \in \mathrm{TIME}(n^k)$.

Table of Contents

Nondeterministic Turing Machines

Until now, we have only considered deterministic models of computation. Non-determinism allows our Turing Machines to "guess".

Nondeterministic Turing Machine Example

Consider the problem of determining if a given boolean formula has a satisfying assignment.

1. For an ordinary Turing Machine, we would need $O(2^n)$ time to check every single assignment to see if there's a satisfying assignment.

Nondeterministic Turing Machine Example

Consider the problem of determining if a given boolean formula has a satisfying assignment.

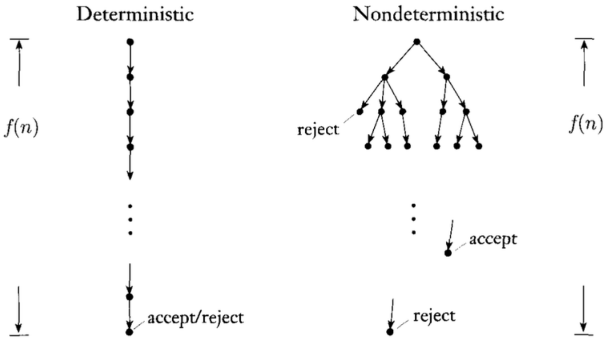
1. For an ordinary Turing Machine, we would need $O(2^n)$ time to check every single assignment to see if there's a satisfying assignment.
2. But what if, we could just try all of them at the same time.

Nondeterministic Turing Machine Example

Consider the problem of determining if a given boolean formula has a satisfying assignment.

1. For an ordinary Turing Machine, we would need $O(2^n)$ time to check every single assignment to see if there's a satisfying assignment.
2. But what if, we could just try all of them at the same time.
3. Equivalently, think of it as: *guess the satisfying assignment if it exists.*

Nondeterministic Turing Machine



Nondeterministic Time Complexity

Definition

A language $L \in \mathrm{NTIME}(f(n))$ iff there exists a Nondeterministic Turing Machine M such that the longest branch for any input is at most $O(f(n))$ in length.

Nondeterministic Time Complexity

Definition

A language $L \in \mathrm{NTIME}(f(n))$ iff there exists a Nondeterministic Turing Machine M such that the longest branch for any input is at most $O(f(n))$ in length.

Definition

A language $L \in \mathrm{NP}$ iff $\exists k$ such that $L \in \mathrm{NTIME}(n^k)$.

Nondeterministic Time Complexity

Definition

A language $L \in \mathrm{NTIME}(f(n))$ iff there exists a Nondeterministic Turing Machine M such that the longest branch for any input is at most $O(f(n))$ in length.

Definition

A language $L \in \mathrm{NP}$ iff $\exists k$ such that $L \in \mathrm{NTIME}(n^k)$.

Equivalent Definition

A language $L \in \mathrm{NP}$ iff there exists a certificate C that can be verified by a deterministic Turing Machine in polynomial time.

Nondeterministic time Complexity

Examples

The problem of determining if a boolean circuit has a satisfying assignment is in NP. i.e, $\mathrm{CIRCUIT} - \mathrm{SAT} \in \mathrm{NP}$.

1. Once the machine has guessed an assignment, it must just plug and check. Hence, each branch is at most linear in size.

Nondeterministic time Complexity

Examples

The problem of determining if a boolean circuit has a satisfying assignment is in NP. i.e, $\mathrm{CIRCUIT-SAT} \in \mathrm{NP}$.

1. Once the machine has guessed an assignment, it must just plug and check. Hence, each branch is at most linear in size.
2. Equivalently, a satisfying assignment is a certificate that can be verified by a deterministic Turing Machine in polynomial time. It simply needs to plug and chug.

NP-hardness

We want a way to think about the relative "hardness" of problems. We do so by thinking about "reductions".

NP-hardness

We want a way to think about the relative "hardness" of problems. We do so by thinking about "reductions".

Definition

A language L reduces to L' under polynomial time reductions iff there exists a polynomial time computable function f such that $x \in L \iff f(x) \in L'$.

NP-hardness

We want a way to think about the relative "hardness" of problems. We do so by thinking about "reductions".

Definition

A language L reduces to L' under polynomial time reductions iff there exists a polynomial time computable function f such that $x \in L \iff f(x) \in L'$.

Definition

A language L is said to be NP-hard iff every language $A \in \text{NP}$ reduces to L under polynomial time reductions. Intuitively, L is at least as hard as A.

Definition

A language L is said to be NP-complete iff it's NP-hard and $L \in \mathrm{NP}$.

NP-completeness

Definition

A language L is said to be NP-complete iff it's NP-hard and $L \in \text{NP}$.

NP-complete problems often let us reason about the entire complexity class by just reasoning about one problem.

NP-completeness

Definition

A language L is said to be NP-complete iff it's NP-hard and $L \in \mathrm{NP}$.

NP-complete problems often let us reason about the entire complexity class by just reasoning about one problem.

Examples

(Cook-Levin Theorem): $\mathrm{CIRCUIT-SAT}$ is an NP-complete problem.

Table of Contents

Oracle Access

Definition

We define C^D as being the set of languages decidable in C using Turing Machines with access to an oracle for D.

Examples

P^{NP} is the set of languages that can be decided in polynomial time by a Turing Machine with access to an oracle which can decide any language $L \in \mathrm{NP}$ in one operation.

The Big Question

Clearly, $P \subset \mathrm{NP}$, but is $\mathrm{NP} \subset P$?

Table of Contents

Proofs that Relativize

Definition

A proof relativizes iff it treats Turing Machines as a black box.

Proofs that Relativize

Definition

A proof relativizes iff it treats Turing Machines as a black box.

Remark

If one proves $C \subset D$ using a relativizing proof, then they have also shown $C^O \subset D^O$ for all oracles O. This is because by treating the Turing Machine as a black-box, adding oracle access doesn't violate any assumptions.

Examples

In the proof that the Halting Problem is undecidable, we assume that there exists an M that decides it, and come up with a Turing Machine for which it must always give the wrong answer. However, we never look into M at any point. It's just a black-box. Hence, it relativizes.

The Relativization Barrier

1. We all want to solve $P \stackrel{?}{=} NP$, but we do not know how (yet).
2. However, we do know something about what kinds of proofs definitely will not work.
3. These are known as proof barriers. The relativization barrier is one of these.

The Relativization Barrier

1. We all want to solve $P \stackrel{?}{=} NP$, but we do not know how (yet).
2. However, we do know something about what kinds of proofs definitely will not work.
3. These are known as proof barriers. The relativization barrier is one of these.

Relativization Barrier

No proof that relativizes can be used to solve $P \stackrel{?}{=} \mathrm{NP}$.

Table of Contents

Baker-Gill-Solovay

In 1975 Baker, Gill, and Solovay showed the following result

Theorem

There exist oracles A, B such that

1. $P^A = \mathrm{NP}^A$
2. $P^B \neq \mathrm{NP}^B$

Baker-Gill-Solovay

In 1975 Baker, Gill, and Solovay showed the following result

Theorem

There exist oracles A, B such that

1. $P^A = \mathrm{NP}^A$
2. $P^B \neq \mathrm{NP}^B$

Remark

This proves the existence of the relativization barrier. If your relativizing proof shows, for instance, $P = \mathrm{NP}$, then it also shows $P^A = \mathrm{NP}^A$ for all oracles A. But, BGS tells us there exists an oracle B such that the $P^B \neq \mathrm{NP}^B$, thus we have a contradiction.

Proof of Baker-Gill-Solovay

Theorem

There exists an oracle A such that $P^A = \mathrm{NP}^A$.

Proof of Baker-Gill-Solovay

Theorem

There exists an oracle A such that $P^A = \mathrm{NP}^A$.

Proof.

Choose $A = EXP$. In general, any sufficiently large class will work. $\square$

Proof of Baker-Gill-Solovay

Theorem

There exists an oracle B such that $P^B \neq \mathrm{NP}^B$.

Proof

We want to construct a language B such that $NP^B \subsetneq P^B$. Define

$$L_B = \{1^n \mid \text{there is a string of length } n \text{ in } B\}$$

Then, observe that $L_B \in \mathrm{NP}^B$, since a machine in NP can just guess the string of length n if it exists. But now, we want to design B such that $L_B \notin P^B$.

Proof of Baker-Gill-Solovay

Proof cont'd

We let $M_1, M_2, \ldots$ be the Turing Machines with Oracle access to B that run in time at most, say, $2^n/10$. Note, the choice of 10 here is essentially arbitrary. We just want something smaller than 2^n (observe, this is weaker than P). Note that these can be enumerated due to the time bound.

Proof of Baker-Gill-Solovay

Proof cont'd

We let $M_1, M_2, \ldots$ be the Turing Machines with Oracle access to B that run in time at most, say, $2^n/10$. Note, the choice of 10 here is essentially arbitrary. We just want something smaller than 2^n (observe, this is weaker than P). Note that these can be enumerated due to the time bound.

Proof cont'd

Now, notice that since each M_i has time less than 2^n, it cannot query all strings of length n. We will exploit this in our design of B. In particular, we will put one of the strings which each M_i does not query in B. This will be our *needle in a haystack*.

Proof of Baker-Gill-Solovay

Proof cont'd

Construction of B:

1. In the i^{th} step, pick a string s of length greater than any strings currently in B, which is not queried by M_i.

Proof of Baker-Gill-Solovay

Proof cont'd

Construction of B:

1. In the i^{th} step, pick a string s of length greater than any strings currently in B, which is not queried by M_i.
2. If M_i accepts $1^{|s|}$, throw it away

Proof of Baker-Gill-Solovay

Proof cont'd

Construction of B:

1. In the i^{th} step, pick a string s of length greater than any strings currently in B, which is not queried by M_i.
2. If M_i accepts $1^{|s|}$, throw it away
3. Otherwise, add s to B.

Observe we now have a pathalogical example for every M_i. Hence, $L_B \notin P^B$. Thus, $NP^B \subsetneq P^B$.

Table of Contents

Other Barriers

There are other known similar proof barriers:

1. The Natural Proofs Barrier

Other Barriers

There are other known similar proof barriers:

1. The Natural Proofs Barrier
2. The Algebraizing Proofs Barrier

Other Barriers

There are other known similar proof barriers:

1. The Natural Proofs Barrier
2. The Algebraizing Proofs Barrier
3. The Relativization Barrier

It is known that proofs for $P \stackrel{?}{=} \text{NP}$ must cross these barriers.

Questions?

Thank You!