# Lecture 12. A glimpse of computational complexity

This year’s offering of the complexity course focuses on concrete models, and we’ll use this last lecture to say something about various models defined by Turing machines. The standard (and recent) references are textbooks by Arora and Barak [AB09] and Goldreich [Gol08]. For automata and Turing machines, a good textbook is [Sip].

Due to the vast amount of the materials, we cannot provide proof for most theorems, though we’ll sketch some key ideas of the proof.

## Time and space

: Languages that can be decided in time on a deterministic Turing machine.

**Theorem (Hierarchy)**: .

(Proof idea: Diagonalization.)

: Languages that can be decided in polynomial time on a deterministic Turing machine.

* Examples: Matching, MaxFlow, LP, PRIMES, …

: Languages that can be decided in exponential time on a deterministic Turing machine.

: Languages that can be decided in space on a deterministic Turing machine

: Languages that can be decided in polynomial space on a deterministic Turing machine.

**Theorem**. TQBF is **PSPACE**-complete, where

for some , constant k, and quantifiers and of polynomial lengths.

(Proof idea: recursively use to search the middle configuration and use to verify both parts.)

: Languages that can be decided in exponential space on a deterministic Turing machine.

: Languages that can be decided in logarithmic space on a deterministic Turing machine.

## Nondeterministic, Alternation, Advice and Randomness

**NTIME**(): Languages that can be decided in time on a nondeterministic Turing machine.

:

* Languages that can be decided in polynomial time on a nondeterministic Turing machine.
* Languages that can be *verified* in polynomial time on a deterministic Turing machine.

**NP**-Complete: Problems in **NP** to which any other **NP** problem can be reduce.

* SAT, Clique, HamiltonianCycle, TSP, SubsetSum, IP, …

While most problems in **NP** are either in **P** or **NP**-complete, there are also problems in between.

**Theorem** (Ladner) If , then there are languages that are neither in **P** or **NP**-complete.

There are some specific problems not known to be in P or NPC. Some examples: Polynomial Identity Testing, Graph Isomorphism, Factoring, DiscreteLog.

One can also define **NEXP**, languages decidable in exponential time on a nondeterministic Turing machine. This class is of course very large. Inside the smaller class **PSPACE**, people defined more classes.

Above **NP**, one can define more classes by using more (alternating) quantifiers.

for some **.**

for some **.**

Basically, they are one-level upper than **NP** and **co-NP**. One can define and for larger *k*. Taking the union for all *k* (independent of the problem input size), we get the class **PH** (Polynomial Hierarchy). The classes and are usually called the *k*-th layer of **PH**, and the first layer contains **NP** and **co-NP**. It’s not hard to see from the definition that and . While each layer has complete problems, **PH** doesn’t unless it collapses to a certain level.

Most researchers believe that **PH** doesn’t collapse, and use this as a computational assumption. A basic fact is that if , then , thus **PH** collapses to the *k*-th level. For example, one can show that if the problem Graph Isomorphism is NP complete, then **PH** collapses to the second level.

Advice is a string given to a Turing machine. It can depend on the input size, but not the input itself. We use to denote the language decided by a -time Turing machine augmented with a -bit advice for inputs of size *n*. The class **P/poly** coincides with the class of languages decidable by polynomial-size circuits. The following theorem says that it’s unlikely that the power of advice is powerful for **NP**.

**Theorem** (Karp-Lipton) **.**

(Proof idea. implies the existence of a circuit family to decide an NP language. This existence is the one to show .)

Randomness plays an important role in algorithm designing. A language L is in **BPP** if it can be decided with bounded error, say, 1/3, by a polynomial-time Turing machine. Though many people believe that , we can’t even rule out the possibility that What we do know includes the facts that and that **.**

(Proof idea of : Reduce the error to exponential small and then find a random string that is correct for all inputs of a certain size. Use this string as the advice. Proof idea of : Reduce the error to exponential small and then prove that iff there exists a set of shifts of the random string s.t. at least one of them makes the machine to accept.)

## Interaction

Interactive proof system: NP plus interaction.

**IP**: Languages decidable by interactive proof system with polynomial-time Verifier.

**Theorem**. **.**

(Proof idea: Some technique called *algebraization*…)

A special type of interactive proof systems is one in which Verifier sends only public coins. These are called **AM**[k], where k is the number of rounds. Let .

**Theorem**. for any constant *k*. Thus .

(Proof idea: Switch the first two rounds, thus Verifier first sends public coins. If the original soundness error is small enough, then the switching maintains a decent soundness.)

One can also define as the languages decidable by *k*-round interactive proof systems.

**Theorem**. . Thus all interactive proofs can be made public-coin.

(Proof idea: The key tool is to efficiently certify the approximate size of a set.)

One can also define one-round **IP**, which is the class **MA**. It’s like NP but now Verifier is a **BPP**-machine (namely, Bounded-error, Polynomial-time and Probabilistic machine).

Another important one-round IP is , in which Verifier uses at most random bits and at most queries to the proof.

**Theorem**. **.**

The PCP theorems have direct implications in approximation algorithms (to show impossibility results) and many other connections to other areas in complexity theory.

One can also consider more provers, who are allowed to discuss before seeing the input. We use **MIP** to denote the class of languages decidable by such multiple prover systems.

**Theorem**. .

## References

[AB09] Sanjeev Arora and Boaz Barak, **Complexity theory: A modern approach**, *Cambridge University Press*, 2009.

[Gol08] Oded Goldreich, **Complexity theory: A conceptual approach**, *Cambridge University Press*, 2008.

[Sip12] Michael Sipser, **Introduction to the Theory of Computation**, 3 edition, *Course Technology*, 2012.