Lecture 20:
Hierarchy Theorems and Review

Valentine Kabanets

December 2, 2016

1 Which separation is true?

In the sequence of inclusions $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$, at least one inclusion must be proper. (Otherwise, we would get $\mathsf{P} = \mathsf{EXP}$, contrary to the Time Hierarchy Theorem, that we will prove next.)

Similarly, in the sequence of inclusions $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$, at least one inclusion must be proper. (Otherwise, $\mathsf{L} = \mathsf{PSPACE}$, contrary to the Space Hierarchy Theorem that we will prove next.)

Thus, we know that some separations of complexity classes are true. We just don't know which ones! (It is conjectured that all inclusions mentioned in this section are proper.)

2 Space Hierarchy Theorem

Using diagonalization, we can show that L is properly contained in PSPACE, and that P is properly contained in EXP.

We say that a function $f(n) > \log n$ is *space constructible* ("nice") if there is a space $O(f(n))$ TM that, on input 1^n, outputs $f(n)$ in binary. Note that all "natural" functions are space constructible (n^2, $n \log n$, etc.)

Theorem 1 (Space Hierarchy Theorem)**.** *For any space constructible function $f(n)$, there is a language D in $\mathsf{SPACE}(f(n))$, but not in $\mathsf{SPACE}(o(f(n))$.*

In words, with more space, we can decide more languages. The following proof is optional.

Proof. We will consider "padded" encodings of Turing machines. That is, for a TM M, we consider the strings of the form $\langle M \rangle 10^k$, for every $k \geq 0$, to be valid encodings of the same TM M. This way, we get that every TM M has *infinitely many* possible descriptions, and so M has a description of *arbitrary large* size. It will be important for us to be able to take a "sufficiently large" description of any given TM M.

The language D is defined as the collection of all TM descriptions $\langle M \rangle 10^k$ such that, during the simulation of TM M on input $\langle M \rangle 10^k$, using space at most $f(n)$ and time at most $2^{f(n)}$, M does *not* accept $\langle M \rangle 10^k$.

Basically, the language D "diagonalizes" against all TMs that *can be simulated* in space $f(n)$. Note it's important that we say "can be simulated" rather than "use space $f(n)$". The reason is that when we simulate, on a universal TM, a given TM M such that M uses space $s(n)$, our simulation will incur a constant factor loss, i.e., our simulation will use space $cs(n)$, for some constant c dependent on the TM M. This is because the alphabet of the universal machine may

be different from that of M, and hence several symbols of the universal machine need to be used in order to simulate one symbol of M.

Now, if the simulated TM M runs in space $s(n) = o(f(n))$, then for sufficiently large n, $s(n) < f(n)/c$. Hence, for sufficiently large n, our universal machine will be able to simulate TM M using at most $cs(n) < f(n)$ space. To make use of "sufficiently large n", we padded the encodings of TMs by 10^k, for any k, so that each TM M will have an encoding of arbitrarily large size, and so the asymptotics will kick in: on this input, TM M will use less than $f(n)/c$ space.

Here's an algorithm for deciding the language D.

Algorithm A="On input w of size n,

1. compute $f(n)$, and mark off $f(n)$ cells of tape. Whenever an attempt is made to use more than $f(n)$ cells, then halt and Reject.
2. if w is not of the form $\langle M \rangle 10^k$ for some TM M and some number k, then Reject.
3. Simulate M on input w (for at most $2^{f(n)}$ steps) using at most $f(n)$ space. If the simulation tries to use more than $f(n)$ space or more than $2^{f(n)}$ time, then Reject.
4. If M accepted w during this simulation, then Reject; otherwise, Accept."

Clearly, by construction, the algorithm A uses space $O(f(n))$. Next, we argue that no space $o(f(n))$ TM can decide the language D.

Suppose, towards the contradiction, that some TM M decides D, and that M runs in space $o(f(n))$. Choose k large enough so that the simulation of M on input $\langle M \rangle 10^k$ can be completed in space at most $f(n)$. (That is, we choose k so large as to ensure that M uses space $< f(n)/c$ on input $\langle M \rangle 10^k$, where c is the constant factor slowdown suffered by our universal TM when simulating this TM M.) Then, on this input, the algorithm A given above will have finished simulating M, and will flip the answer of M. Hence, $L(A) = D$ is different from $L(M)$. A contradiction. This finishes the proof of the Space Hierarchy Theorem. □

Corollary 1. NL *is properly contained in* PSPACE.

Proof. NL is in $\mathsf{SPACE}(\log^2 n)$, by Savitch's theorem. By the Space Hierarchy theorem, $\mathsf{SPACE}(\log^2 n)$ is properly contained in $\mathsf{SPACE}(n)$. Hence, NL is properly contained in $\mathsf{SPACE}(n)$, and so in PSPACE. □

3 Time Hierarchy Theorem

We say that a function $t(n) > n \log n$ is *time constructible* ("nice") if there is a TM such that, given input 1^n, it outputs the binary representation of $t(n)$, and the running time of the TM is $O(t(n))$.

Theorem 2 (Time Hierarchy Theorem)**.** *For any time constructible function $t(n)$, there is a language in* $\mathsf{TIME}(t(n))$ *which is not in* $\mathsf{TIME}(o(t(n)/\log t(n)))$.

Proof sketch. The proof idea is the same as in the case of space hierarchy from above: we define a "diagonal" language that cannot be decided by any TM running in time $o(t(n)/\log t(n))$.

The main difference is in the loss of additional $\log t(n)$ factor in the simulation. This loss is due to the fact that our universal machine (which does the simulation) needs to decrement the counter to count out $t(n)/\log t(n)$ steps of the simulated machine. Decrementing the counter takes time linear in the binary length of the counter, which is $O(\log t(n))$. Another reason for this loss is that

we need to simulate a TM with arbitrary (constant) number of tapes; using a universal 2-tape TM, we can simulate any time t TM in time $O(t \log t)$.

That's why we have to diagonalize against only those TMs that run in $o(t/\log t)$ time. □

Corollary 2. P *is properly contained in* EXP.

4 Why can't we prove $\mathsf{P} \neq \mathsf{NP}$ using Time Hierarchy?

Note that $\mathsf{P} \subseteq \mathsf{TIME}(n^{\log \log n})$ (for obvious reasons). We can construct a diagonal language D that is, say, in $\mathsf{TIME}(n^{\log n})$, but not in $\mathsf{TIME}(n^{\log \log n})$. Thus, in particular, $D \notin \mathsf{P}$.

If we could somehow show that this $D \in \mathsf{NP}$, we would get that $\mathsf{P} \neq \mathsf{NP}$! The problem is: we don't know (nor really believe) that this particular language D is in NP. We need more ideas!

There is another reason why diagonalization by itself is insufficient to resolve the "P vs. NP" question. It is outlined below.

5 Barriers

The material in this section is optional.

Why can't we resolve the "P vs. NP" question? One of the basic answers is: We need "new techniques"! But what are the old techniques and why are they not useful for resolving the big open questions such as the "P vs. NP"?

There are several formal answers. The first one was given by Baker, Gill, and Solovay in 1975. Their message was:

> "black-box" (relativizing) techniques alone are not enough to resolve "P vs. NP".

5.1 Relativization

What are *relativizing* techniques? We need the notion of *oracle Turing machines*. These are TMs with an extra tape, oracle tape, where they can write a query (a string), and in the next step of computation, they get back the answer whether that string is in the oracle or not. When we choose a particular oracle A (which, recall, is just a language $A \subseteq \{0,1\}^*$), our oracle machine will get its oracle queries answered according to that language A. This oracle TM is denoted M^A to stress that it has oracle access to the oracle A.

If you recall proofs such as $\mathsf{P} \subseteq \mathsf{NP}$ or $\mathsf{EXP} \not\subseteq \mathsf{P}$, you will convince yourself that the same proofs also show that, relative to *any* oracle A, $\mathsf{P}^A \subseteq \mathsf{NP}^A$ and $\mathsf{EXP}^A \not\subseteq \mathsf{P}^A$. In general, we say that a proof technique/argument is *relativizing* if it remains valid relative to every oracle A. That is, if we prove some statement (e.g., $\mathsf{EXP} \not\subseteq \mathsf{P}$) using relativizing arguments, then our proof can be modified to also show the same statement relative to any given oracle A; the modification is syntactic – add oracle access to A everywhere in the proof where you talk about TMs.

The two basic techniques, simulation (which means that there is a TM that can simulate any other TM with not too much overhead) and diagonalization (which means we can construct a language not accepted by any TM of certain type), are examples of relativizing techniques!

Such techniques are "black-box" in the sense that we are treating the TM as a "black-box": Given a TM M, we don't open it up, but simply run (simulate) it on a given input x. If that TM were an oracle TM with oracle A, we would still just run it on a given input x. If our proof methods

are based only on the idea "just run it" for Turing machines (i.e., be completely black-box), then, as Baker, Gill, Solovay argued, we will never resolve the "P vs. NP" question!

More formally, they proved the following:

Theorem 3 (Baker, Gill, Solovay)**.** *There are oracles A and B such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.*

As a consequence, *there cannot be a relativizing proof of $\mathsf{P} = \mathsf{NP}$, or a relativizing proof of $\mathsf{P} \neq \mathsf{NP}$.* The reason is that such a proof would remain valid relative to any oracle (by the definition of a relativizing proof), but there are oracles relative to which "P vs. NP" has opposite answers.

Proof of Theorem 3. Take $A = TQBF$, the PSPACE-complete language. We have that $\mathsf{P}^A \subseteq \mathsf{NP}^A$ (for trivial reasons), then $\mathsf{NP}^A \subseteq \mathsf{NPSPACE}$ (as we can answer A-queries ourselves in PSPACE), then by Savitch's theorem we have $\mathsf{NPSPACE} = \mathsf{PSPACE}$, and finally, by the definition of PSPACE-completeness, we get $\mathsf{PSPACE} \subseteq \mathsf{P}^A$. Thus, we got

$$\mathsf{P}^A \subseteq \mathsf{NP}^A \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{P}^A.$$

This implies that all inclusions are in fact equalities, and so in particular $\mathsf{P}^A = \mathsf{NP}^A$.

The construction of oracle B is done in stages, and uses diagonalization. We omit the details here (but you should consult the textbook). □

5.2 Non-relativizing techniques

Do we have non-relativizing techniques? Yes, we do! In fact, already the proof of the Cook-Levin theorem is non-black-box: to reduce the computation of a nondeterministic TM on a given input x to a 3-cnf ϕ_x, we had to "open up" the TM and look inside at the sequence of configurations the TM goes through, and moreover, exploit a very special property of a TM computation: that the computation is *locally checkable*. The latter simply means that to decide if a certain symbol in position i of a configuration at time t is correct, we need only to check the symbols in positions $i-1, i, i+1$ in the configuration at time $t-1$. This "local checkability" is used to get a 3-cnf formula which is satisfiable iff the computation is accepting.

Another example is NP-completeness of 3-COLORING. Again, to reduce computation to a graph problem, we essentially relied on the local checkability of computation. This reduction completely breaks down if we allow NP machines to have arbitrary oracles.

It was argued by Arora, Impagliazzo and Vazirani in the early 1990's that "local checkability" of the computation is all one needs to know about computation in order to prove any true complexity statement.[1] Of course, this in itself doesn't make it any easier to prove new complexity results – no more so than knowing the axioms of set theory makes it any easier to prove Fermat's Last Theorem! What we want is to know how to make use of local checkability to prove something interesting.

One example is the proof that $\mathsf{PSPACE} = \mathsf{IP}$ we saw.

6 Overview

6.1 Computability

- simple model of computation: Finite Automaton (FA)

[1] They also define the logical theory for relativizing complexity: All relativizing complexity statements are provable in that theory, and everything provable in the theory is relativizing. Statements with contrary relativizations, such as "P vs. NP", are *independent* of this logical theory.

  - variant: DFA = NFA , accept *regular* languages; can be represented also by regular expressions.
  - lower bounds: non-regular languages via the Pumping Lemma.

- general model of computation: Turing Machine (TM) = our model for an *algorithm*
  - variants: multi-tape, multi-head, etc.
  - DTM = NTM, accept semi-decidable languages;
  - decidable languages (proper subset of semi-decidable)
  - lower bounds: undecidable, non-semidecidable via diagonalization (A_{TM}, E_{TM}, etc.)
  - reductions ($A \leq_m B$): to show hardness (of B) and easiness (of A)
  - self-reference: Recursion Theorem (self-printing programs, impossibility of perfect virus-checker), Gödel's Incompleteness Theorems in Logic (powerful proof systems cannot be both sound and complete)
  - application to Information Theory: Kolmogorov complexity as a measure of information contained in a string, which can be algorithmically extracted; incompressible = random

6.2 Complexity

Complexity can be viewed as "scaling down" computability by taking into account Time and Space complexity measures. Here, efficient = polytime.

We can think of P as corresponding to Decidable, and NP as corresponding to Semi-Decidable. Unlike in Computability, we don't know if $P \neq NP$ (which is conjectured).

- P and NP: NP-completeness (SAT is NP-complete, lots of natural NP-complete problems, e.g., SAT, Clique, VC, IS, SubsetSum, HamCycle, TSP, 3-COL)
- Space: $\mathsf{NPSPACE} = \mathsf{PSPACE}$ (Savitch); $\mathsf{NL} = \mathsf{coNL}$ (Immerman-Szelepcsenyi).
- lower bounds: Time/Space Hierarchy Theorems (in more time/space, can decide more languages) via diagonalization.
- limits of diagonalization: there exist oracles A and B such that $\mathsf{P}^A = \mathsf{NP}^A$, and $\mathsf{P}^B \neq \mathsf{NP}^B$. So need non-relativizing techniques to resolve big open questions in complexity!
- randomized computation: RP, BPP; Polynomial Identity Testing
- interactive proof system for #SAT (an example of a non-relativizing proof technique!): uses *interaction* and *randomness*
- PCP Theorem.