18.405J/6.841J: Advanced Complexity Theory Spring 2016

Lecture 2: Relativization

Prof. Dana Moshkovitz

Scribe: Di Liu
Scribe Date: Fall 2012

Overview. Last week we showed $NTIME(n) \nsubseteq TISP(n^{1.2}, n^{0.2})$. However, we see that this is a relatively weak conclusion. Can we prove $P \neq NP$ with similar techniques?

We will show that such techniques alone cannot prove $NP = P$ or $NP \nsubseteq P$.

How does a standard complexity proof work? Take some Turing machine M_1 that recognizes a language L.
... Now let M_2 return the inverse of M_1.
... Now let M_3 simulate M_2 on part of the input and do something else with another part.
... Now let M_4 add a quantifier
... And so on.

These proofs use black-boxes; for instance with no knowledge of M_1, if M_2 uses M_1 and eventually M_i cannot exist, M_1 could not have existed.

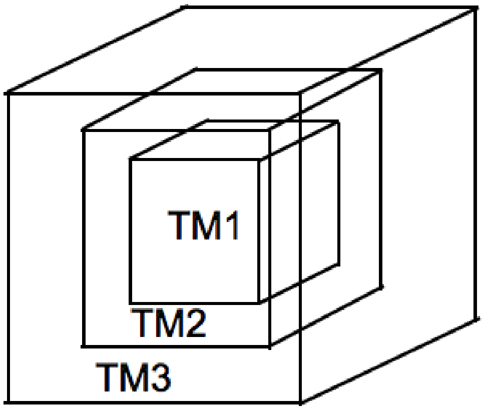

Figure 1: Nested black-box Turing machines are like Matryoska dolls.

These sorts of techniques are extremely useful:

- Hierarchy theorems are proven this way
- Time-space lower bounds are proven this way
- **Theorem 3.4: Ladners Theorem**[1], which says that if $P \neq NP$, then there are many languages in $NP \setminus P$ that are not NP-complete, is proven this way

- ... Many other examples

These techniques will necessarily fail. We will identify one thing every such proof must satisfy, and show that any proof that shows either $P = NP$ or $P \neq NP$ cannot satisfy this one property.

Define an **oracle** as a language. A Turing machine may query an oracle by writing an input to its tape, calling the oracle, and then reading whether or not the input is in the oracle's language back off of the tape. The reading and writing count toward the computation time of the Turing machine, but any "work" the oracle did to get the answer does not.

Suppose I have M_2, which uses M_1, and M_1 has access to oracle O. We can recognize the same language by giving the oracle to M_2 instead of M_1. When the inner Turing machine needs to use the oracle, we can pause simulation of it and calculate the answer in M_2. This way we can "bubble up" the oracle as many layers as we like. Thus this proof technique is "relativizing": changing the underlying model (adding or changing the oracle) does not affect the validity of the proof.

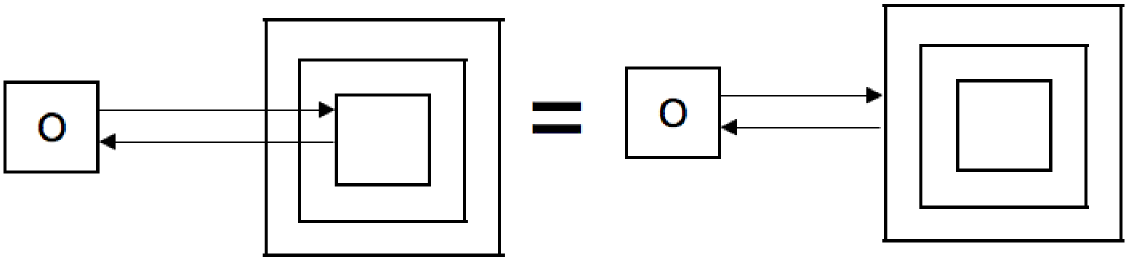

Figure 2: Nested black-box Oracles are like bubbles.

Let C = class of languages that can be solved by TM with certain resources.
Let O = some language
Let C^O = a class of languages that can be solved by TMs with the same resources + oracle access to O.

Observation. If $NP \not\subseteq P$ can be proved using "this technique" (relativizing proof techniques), then for any oracle O,

$$NP^O \not\subseteq P^O$$

Remarks

- We use a notation that lookcs like powering, and for numbers, $a^c \neq b^c \rightarrow a \neq b$. Somehow people inherently assume this holds for complexity classes. This is likely FALSE for classes and oracles!

  For example, $\exists B$ such that $RP^B \neq P^B$, $BPP^B \neq P^B$, although we believe $P = RP = BPP$. Later in the semester we will show that $BPP = P$ under hardness assumptions.

> *"There is probably an example without the likely."*
> – Dana Moshkovitz

- Note that there is a reason we defined oracles the way we did; as something passed to a Turing machine. We know NP is also defined as recognizing the set of languages recognized by non-deterministic finite automata, or the set of languages recognized by Turing machines which are passed a verifier. However, there is no sense of what an oracle is in those and many other definitions of complexity classes (would it be given to the prover or the verifier?).

Theorem 3.7: Baker-Gill-Solovay, 1975 [1][2]

There exist oracles A, B such that:

(i) $P^A = NP^A$, but

(ii) $P^B \neq NP^B$

Significance: there can be no relativizing proof for $P = NP$.

Proof.

(i) Let $A = EXP$, any EXP-complete language. Then $NP^{EXP} = P^{EXP}$. (We may choose a complete language from any of many very powerful classes. Note for instance that an NP-complete language would not be powerful enough, but a PSPACE-complete language is.)

(ii) - Idea behind constructing B:

I need to specify a problem in NP^B not in P^B. Let

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

Why is this a good choice? Notice that $L_B \in NP^B$, since we can guess all strings of length n and ask if they're in B.

Think of a machine M that tries to decide L_B, if a string of length n is in B. The only way it knows anything about B is through the oracle.

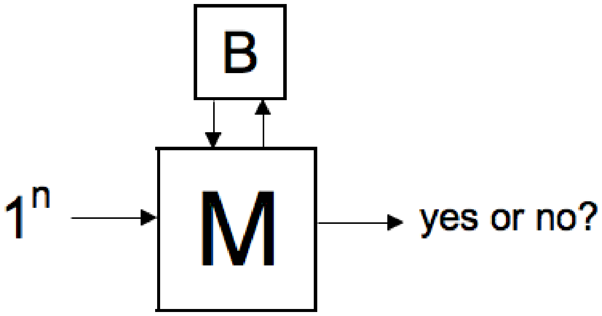

Figure 3: *Think of the poor machine that has to decide L_B.* –Dana Moshkovitz

Suppose no matter how M queries, B responds "no". Can M possibly say "yes"? No, because B would always respond "no" if it's empty, in which case M should say "no".

We will construct B such that the real answer is "yes". This problem is known as a "Needle in a haystack!"

Figure 4: *Why doesn't it look as good as the one in my notes?* –Dana Moshkovitz

- Formally:

  Let an **oracle machine** be a Turing machine with access to some fixed oracle, B.

  Let $M_1, M_2, M_3, ...$ be all possible oracle machines. (We can enumerate them because Turing machines which can query oracles can be represented by finite strings of a finite number of characters, which there are countably many of. The oracle is fixed and separate from this list of machines.) Limit machines to decide L_B in time $2^n/10$ (note that this is a weaker constraint than P). We want none of these machines to decide whether a string is in L_B within time $2^n/10$.

  Since there are 2^n strings of length n, and $2^n > 2^n/10$, there are necessarily strings of length n that are not queried by M_i. Thus to construct B:

  On step i, choose a string s of length much greater than the length of any string already in B, that is not queried by M_i.

  - If M_i accepts s, do not add it to B.
  - If M_i rejects s, add it to B.

  Thus for no i does M_i recognize L_B.

What did we prove? Did we show that relativizing proof techniques are useless? No! We showed that if someone proves $NP \nsubseteq P$, then some part of the proof must be non-relativizing.

However, it is likely that it will use relativizing technizues in addition to other techniques.

Note that nonrelativizing proofs do not necessarily put you in the clear. Even if a statement is proved with a proof that is not relativizing, it may also be provable with a proof that is.

What proof techniques don't relativize?

$IP = PSPACE$.

PSet problem 3.

References

[1] Theodore P. Baker, John Gill, Robert Solovay: Relativizatons of the P =? NP Question. SIAM J. Comput. 4(4): 431-442 (1975)

[2] Sanjeev Arora, Boaz Barak: Computational Complexity - A Modern Approach. Cambridge University Press 2009: I-XXIV

18.405J / 6.841J Advanced Complexity Theory
Spring 2016