CSE491/596 Lecture Fri. 11/20/20: Relativization and Randomness

The most important example of using an oracle Turing machine, IMHO, is the manner in which a function f can be computed if we have an oracle for its associated "undergraph" language:

$$L_f = \{\langle x,w\rangle : w \leq f(x)\}.$$

Here $\leq$ is with respect to the natural ordering of strings---which puts shorter strings before longer ones and otherwise lists same-length strings in alphabetical order. But we can also regard f as a numerical function $f: \mathbb{N} \to \mathbb{N}$ under the natural correspondence of Σ^* and $\mathbb{N}$, whereupon $\leq$ is just ordinary numerical less-than-or-equal.

The method is simply **binary search**. You've probably seen the algorithm in some form, but let's view it in the oracle machine form. The one thing we need to start the binary search, on any given x where we want to compute $f(x)$, is a quick way to compute a bound B_x over how big $f(x)$ could be. For instance, B_x can be 0^{m+1} where we know $|f(x)| \leq m$ (and where $m = |x|^k$, say).

```
lo = $\epsilon$; hi = $B_x$;
while (hi - lo > 1) {     INV: lo $\leq$ $f(x)$ < hi
   mid = a string midway between lo and hi;
   query: is <x,mid> in the oracle language $L_f$?
   if (yes) {
      lo = mid;
   } else {
      hi = mid;
   }
}  //on exit, hi = lo+1 and by INV, lo must equal $f(x)$.
return lo;
```

Even though there are exponentially many possibilities for $f(x)$ among strings of length $\leq |x|^k$, indeed order-of 2^{n^k} possibilities where $n = |x|$, the binary search needs at most n^k iterations, each writing a query of length up to n^k. Thus the whole thing runs in time order-of $(n^k)^2 = n^{2k}$. If $m = O(n)$, which is the most usual case where the length of $f(x)$ is proportional to the length of x, then $k = 1$ and this all runs in $O(n^2)$ time.

Example: Recall the factoring language was defined as FACT $= \{\langle N,k\rangle :$ N has a prime factor p such that $p \leq k\}$. Let us instead define it as FACT $= \{\langle N,k\rangle :$ N has a prime factor p such that $p \geq k\}$. Then FACT is the undergraph of the function $f(N) =$ the greatest prime factor of N. The function exnables us to factor N by repeatedly finding and dividing out its prime factors. And binary search tells us that the language FACT is enough to compute the function:

Theorem: Factoring is computable in quadratic time given FACT as an oracle.

This is summarized by saying that s*earch reduces to decision* and is a major reason why languages not functions are used as the main objects in complexity theory. A second example defines $L'_f = \{\langle x, w\rangle : w \sqsubseteq f(x)\}$, where $\sqsubseteq$ means "is a prefix of." Consider this code:

```
w = $\epsilon$; if ($\langle x,w\rangle \notin L'_f$) then output **fail**.
while(true) {
    if ($\langle x,w0\rangle \in L'_f$) {
        w = w0;
    else if ($\langle x,w1\rangle \in L'_f$)
        w = w1;
    else return w;
}
```

On input x, this stops when $w = f(x)$ and returns w. So it computes $f(x)$. The main example is the idea of building up an answer string w---such as a satisfying truth assignment---bit by bit:

Theorem: If SAT is in P, then not only could we decide whether a given Boolean formula ϕ is satisfiable, we would be able to find a satisfying assignment in polynomial time, or tell that none exists.

Relativized Classes and Turing Reductions

If C is a class of *machines*, then we write C^B to be the class of languages $L(M^B)$ over all machines M. Thus for any language B,

- $\mathsf{P}^B = \left\{L(M^B) : \textit{The DOTM M runs in polynomial time (with oracle B)}\right\}$
- $\mathsf{NP}^B = \left\{L(N^B) : \textit{The NOTM N runs in polynomial time (with oracle B)}\right\}$
- $\mathsf{PSPACE}^B = \left\{L(M^B) : \textit{The DOTM M runs in polynomial space (with oracle B)}\right\}$

If $A \in \mathsf{P}^B$ then we also write $A \leq_T^p B$ and say that A **polynomial-time Turing reduces** to B. (Older usage: A "Cook-reduces" to B, in constrast to saying "Karp-reduces" for $\leq_m^p$.) Note: if $A \leq_m^p B$ then $A \leq_T^p B$, i.e. Turing reductions are more general than mapping reductions.

Fact: For any language A, $A \leq_T^p \widetilde{A} : M^{\widetilde{A}}(x)$ queries $y = x$ and accepts x iff the oracle says **no**.

Theorem: $\mathsf{P}^{\mathsf{TQBF}} = \mathsf{NP}^{\mathsf{TQBF}} = \mathsf{PSPACE}$.

Proof: PSPACE is contained in $\mathsf{P}^{\mathsf{TQBF}}$ because **TQBF** is PSPACE-complete under $\leq_m^p$ and an OTM can carry out a many-one reduction f by making just one query $y = f(x)$ and accepting x iff the oracle says "yes" to y. Now let $A \in \mathsf{NP}^{\mathsf{TQBF}}$ via a *nondeterministic* OTM (NOTM) N that runs in time $O(n^k)$. Then on any input x of length n, $N(x)$ can make up to n^k nondeterministic steps and can write queries y (which are quantified Boolean sentences that may be true or false) of length up to n^k. Without loss of generality, we may suppose the nondeterministic steps are binary-branching. A **deterministic**, **non-oracle** Turing machine M can accept A the following way:

1. Use n^k tape cells to cycle through all possible nondeterministic sequences $r \in \{0,1\}^{n^k}$.
2. For each r, simulate $N(x)$ with r deterministically until it writes a query y (which gets stored on a worktape of n^k cells that counts against the space bound.)
3. Answer y deterministically by solving TQBF in linear space, which here means space $O(n^k)$.
4. Then go back to step 2 until the next query y', answer it per step 3, and repeat until the computation path of $N^{\mathsf{TQBF}}(x)$ with r finishes. If it accepts, then accept x; else try the next r, and finally reject if no r works.

This all adds up to $O(n^k)$ space used by M, so A belongs to PSPACE. ☒

The subversive impact of $\mathsf{P}^{\mathsf{TQBF}} = \mathsf{NP}^{\mathsf{TQBF}}$ comes from the following "meta-theorems":

Meta-Theorem 1: Every theorem about (un-)decidability and reductions and relationships among deterministic and nondeterministic time and space that is taught in this course **relativizes**, meaning that for any oracle language B (or oracle function f), the statements hold when all machines and classes are defined with access to B (or f). For example, the relativized diagonal language

$$D^B = \left\{\langle M\rangle : \textit{M is a deterministic OTM and } M^B \textit{ does not accept } \langle M\rangle\right\}$$

is not "decidable in B" nor even "computably enumerable in B"---the latter meaning there is no OTM Q such that $L(Q^B) = D^B$. The proof is essentially the same: $Q^B(\langle Q\rangle) = ??$. This is because the proof only pays attention to the input/output behavior of the programs, taking no care about how they process information internally---and might cheat! Thus, for any B, we have:

1. $\mathsf{RE}^B \neq \mathsf{co\text{-}RE}^B$
2. $\mathsf{RE}^B \cap \mathsf{co\text{-}RE}^B = \mathsf{REC}^B$.
3. $A_{TM}^B = \left\{\langle M, w\rangle : w \in L(M^B)\right\}$ is complete for RE^B under $\leq_m^{\log}$. (The reductions do not need to use the oracle---they are only translating syntax with no regard to the oracle feature.)

It is even possible to define a "relativized version" $3\mathrm{SAT}^B$ in which special formula variables reference the oracle's yes/no answers and prove by extending the Cook-Levin idea that it is complete for NP^B (again under $\leq_m^{\log}$ reductions that do not need the oracle for themselves). This is broadly known as the "principle of relativization for elementary methods." **But** this means:

Meta-Theorem 2: Such elementary methods cannot prove $\mathsf{P} \neq \mathsf{NP}$.

Proof: If they could, then by the principle of relativization, they would prove $\mathsf{P}^B \neq \mathsf{NP}^B$ for all languages B. But we just proved that $\mathsf{P}^{\mathsf{TQBF}} = \mathsf{NP}^{\mathsf{TQBF}}$. ☒

More briefly put: the formal system behind CSE491/596 can prove its own inability to prove the conjectured side of the greatest problem in the field (unless CSE491/596 is inconsistent). It cannot prove the other side $\mathsf{P} = \mathsf{NP}$ either, owing to the counterpart result:

Theorem: We can build a language B such that $\mathsf{P}^B \neq \mathsf{NP}^B$.

[Proof Sketch: For any language B, NP^B includes the language

$$L^B = \left\{0^n : (\exists y)[|y| = n \wedge y \in B]\right\}.$$

Now suppose we have any finite set F and deterministic OTM M that runs in polynomial time $p(n)$. We can choose a number n such that $2^n > p(n)$ and all strings F have length $< n$. Now simulate M^F on input 0^n while keeping track of (i) all strings y of length n that $M^F\left(0^n\right)$ queries, (ii) if $M^F\left(0^n\right)$ queries a string of length $> n$, let m be the length of the longest such string, else $m = n+1$, and (iii) whether M^F accepts 0^n. Note that all queries of length n and higher get answered "no" since F has no strings of those lengths.

- If M accepts 0^n, then do nothing at length n. We have $0^n \notin L^F$ but $0^n \in L(M^F)$.
- If M rejects 0^n, then by $2^n > p(n)$, there must be some string $z \in \{0,1\}^n$ that was not queried. Add z to F.
- Either way, there must be some string $z' \in \{0,1\}^m$ that M did not query either. Add z' to F to make F'.
- Then the second step gave us $0^n \in L^{F'} \setminus L\left(M^{F'}\right)$ and the third step did nothing to change that.

Moreover, because we added 1^m to F', if we repeat this process with a new polynomial $p'(n)$-time machine M' using oracle F', all further alterations will occur at lengths $> m$ and hence not disturb the way we guaranteed that M cannot agree with L^F on the string 0^n. Thus we can repeat the process on M' without disturbing how we "digonalizes against" M. As we continue, the sequence of finite sets builds a language B. Since every polynomial-time OTM eventually gets tried and defeated, we get that the final L^B does not belong to P^B. ☒]

Note, by the way, that there is no contradiction or paradox in the *fact* of having $\mathsf{P}^B \neq \mathsf{NP}^B$ yet $\mathsf{P}^A = \mathsf{NP}^A$ (with $A = \mathsf{TQBF}$). The issue is what range of techniques can be used to *prove* it. The great lack in the field as it stands is that no one has been able to formulate an incisive measure of how much information has been incrementally internally "processed." But we have come to understand the blockages---called **barriers**---better and better. Most of the ones after the "relativzation barrier" have to do with *randomness* and its relation to computational *hardness*.

Randomized Complexity

Our last "big fact" in classical complexity theory is that random bits can be a time-saving resource. How this can happen is best conveyed by an example.

Suppose you have three $n \times n$ matrices A, B, C for which it is claimed that $AB = C$. What is the quickest way you can check this? You can:

- Multiply AB out and see if the answer equals C. Uisng the basic matrix multiplication algorithm, this takes time $O(n^3)$. Using various forms of *Strassen's Algorithm*, one can push this down to $O(n^{2.81\ldots})$, or down below $O(n^{2.5})$ with much bigger constants in the "O".
- Try checking $A \cdot Bu = Cu$ for multiple vectors u. Per vector u, this takes only $O(n^2)$ time. If you ever get $A \cdot Bu \neq Cu$ then you know $AB \neq C$. But if you keep getting equal results for trial vectors u, can you really be sure that $AB = C$?

The answer is that although you can never be certain, you can make the odds of "$AB = C$" being a **false positive** very low by judiciously *random* choices of trial vectors u, while still using only $\widetilde{O}(n^2)$ time overall. The tilde means "ignoring some power of $\log n$." Let's do this rigorously when the matrices are over the field mod-2, which is actually the worst case.

For any natural number m, $\mathbb{Z}_m$ stands for the integers modulo m. If m is a prime number p, then $\mathbb{Z}_p$ is a *field* (so that one can divide as well as multiply) and we write it as $\mathbb{F}_p$. The field structure helps us prove the following result more easily.

Lemma: Suppose A, B, C are $n \times n$ matrices over $\mathbb{F}_p$ such that $AB \neq C$. Then

$$\Pr_{u \in \mathbb{F}_p^n}[(A(Bu) \neq Cu] \geq \frac{p-1}{p}.$$

Proof: Write $D = AB - C$. Note that we are not going to *calculate* D, because that would take the (standardly cubic) time for multiplying A and B that we are trying to avoid, but we are allowed to *argue based on its existence*. By linearity, $ABu \neq Cu \iff Du \neq 0$. So D has at least one row i with a nonzero entry, and its use may give a nonzero entry in the i-th place of the column vector $v = Du$.

Note that

$$v_i = \sum_{j=1}^{n} D[i,j]u_j \,.$$

Let j_0 be a column in which row i has entry $c = D[i,j_0] \neq 0$. For any vector u, we can write

$$v_i = cu_{j_0} + a \quad where \quad a = \sum_{j \neq j_0} D[i,j]u_j.$$

The key observation is that because $\mathbb{F}_p$ is a field, for any $c \neq 0$, the values cu_{j_0} run through all p possible values as u_{j_0} runs through all p possibilities. Regardless of the value of a determined by the rest of row i and the rest of the vector u, the values $cu_{j_0} + a$ run through all p possibilities with equal probability. Hence the probability that $v_i \neq 0$ is exactly $\frac{p-1}{p}$. The probability of getting $v \neq 0$ (which could come from other nonzero entries too) is at least as great. The upshot is:

- If $AB = C$ then you will never be deceived: you will always get equal values from $A(Bu)$ and Cu and will correctly answer "yes, equal."
- If $AB \neq C$ and you try k vectors u at random, if you ever get $A(Bu) \neq Cu$ then you will know to answer "no, unequal" with 100% confidence.
- If you get equality each time, you will answer "yes, equal" but there is a $\frac{1}{p^k}$ chance of error.

If you consider, say, a 1-in-n^3 chance of being wrong as minuscule, then you only need to pick k so that $p^k > n^3$, so $k = \frac{3}{\log p} \log n$ will suffice. Presuming p is fixed, this means $O(\log n)$ trials will suffice. The resulting $O\left(n^2 \log n\right) = \widetilde{O}\left(n^2\right)$ running time handily beats the time for multiplying AB out. Thus we trade off *sureness* for *time*. ☒

For arithmetic modulo m not prime, or without any modulus, the analysis is messier---but not only is the essence the same, but the asymptotic order of k in terms of n and the confidence target $\epsilon(n)$ is much the same---it didn't really depend on p to begin with.

Randomized Complexity Classes

The matrix example makes the probability easiest to figure because it is *linear*, but it does not show a difference between "polynomial" and "exponential". This is enshrined in the definitions of the complexity classes BPP, RP, and co-RP. It is convenient to think of polynomial-time computable predicates $R(x,y)$ where y ranges over $\{0,1\}^{p(n)}$ with equal length rather than say $|y| \leq p(n)$ (with $n = |x|$ as usual). Then y is a sequence of $p(n)$ coin-flips.

Definition: A language L belongs to BPP if there is a polynomial p and a polynomial-time decidable predicate $R(x, y)$ such that for all n and x of length n:

$$\begin{aligned} x \in L &\implies \Pr_{|y|=p(n)}[R(x,y)] > 3/4; \\ x \notin L &\implies \Pr_{|y|=p(n)}[R(x,y)] < 1/4. \end{aligned}$$

If the second probability is always 0 then L is in RP; if instead the first probability is always 0 then A is in co-RP; together these cases are called having *one-sided error*. Note that the first probability being always 1 is equivalent to saying it is always 0 for the complementary predicate $\widetilde{R}(x, y)$, which is where RP and co-RP start to get confusing. The same ability to flip between R and its negation tells right away that BPP is closed under complements, which makes it less confusing. For BPP, we can also combine the conditions into one, namely

$$\Pr_{|y|=p(n)}[L(x) = R(x,y)] > 3/4.$$

But this is often less helpful than having the two separate probabilities. Note that if the second probability is 0 then $R(x, y)$ is impossible when $x \notin A$. It follows that having $R(x, y)$ be true makes y a valid *witness* for $x \in A$, so we have proved the following:

Proposition: RP $\subseteq$ NP and co-RP $\subseteq$ co - NP. ☒

Of course $L \in$ RP $\iff \widetilde{L} \in$ co - RP, so whether a problem belongs to RP or to co-RP depends on which side one takes as the "yes" side. If you regard $AB = C$ as the yes side and $ABu = Cu$ as the verifying predicate "$R(\langle A, B, C\rangle,\ u)$", then the matrix example has one-sided error of the "co-RP type", meaning that if the answer is yes then you can never be bluffed into thinking the answer is no; but in a true-negative case there is a tiny chance of getting a false positive (i.e., thinking $AB = C$ because every u that you tried gave $A(Bu) = Cu$). You could say that the language

$$L = \{\langle A, B, C\rangle : AB = C\} \text{ belongs to co-RPTIME}\left[\widetilde{O}\left(n^2\right)\right],$$

but this notation gets ugly and hides the dependence between the error probability and the time allowed for multiple trials. It is, however, even surer than for the $AB = C$ matrix case:

Amplification Lemma: If $A \in$ BPP with associated $R(x, y)$ and $p(n)$, then for any polynomial $q(n)$ we can build a polynomial-time decidable $R'(x, z)$ and associated polynomial $p'(n)$ such that for all x,

$$\begin{aligned} x \in A &\implies \Pr_{|z|=p'(n)}[R'(x,z)] > 1 - 2^{-q(n)}; \\ x \notin A &\implies \Pr_{|z|=p'(n)}[R'(x,z)] < 2^{-q(n)}. \end{aligned}$$

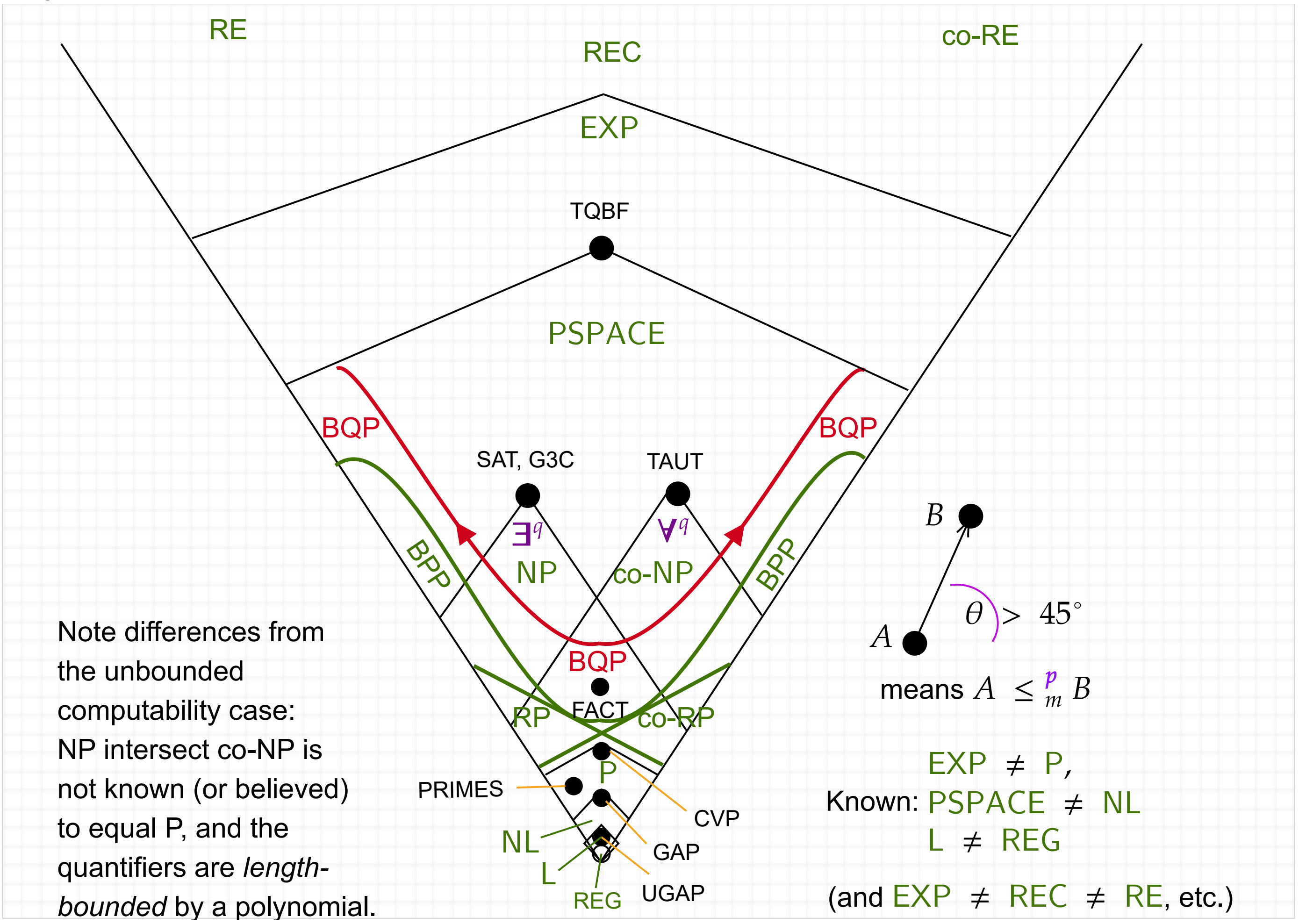
Moreover, we can achieve this even if the original R and p only give a "non-negligible" advantage, meaning that for some polynomial $r(n) \geq n$,

$$\begin{aligned} x \in A &\implies \Pr_{|y|=p(n)}[R(x,y)] > \frac{1}{2} + \frac{1}{r(n)}; \\ x \notin A &\implies \Pr_{|y|=p(n)}[R(x,y)] < \frac{1}{2} - \frac{1}{r(n)}. \end{aligned}$$

Proof Sketch: Regard $z = \langle y_1, y_2, \ldots, y_{q'(n)} \rangle$ where $q'(n) = O(q(n))$ and define $R'(x,z)$ to be the majority vote of the polynomially-many trials $R(x, y_j)$. ☒

There is a similar amplification lemma for one-sided error; in fact, the details of getting the exponentially small error are simpler because you don't need majority vote. A philosophical point is that the the theoretical software error can be reduced below the chance of hardware error---but when you see something like https://www.wnycstudios.org/podcasts/radiolab/articles/bit-flip (which I heard on NPR two weeks ago), maybe that's not so reassuring...

The definition of the quantum complexity class BQP is similar, except that in place of getting y such that $R(x,y)$ by rolling classical dice, we have a *quantum circuit* C in place of R and get the effect of y by measurements. Amplification and many other properties hold similarly; the main external difference is that the factoring problem and some others belong to BQP but (hopefully!) not to BPP. Well, we have to start by defining quantum circuits and algorithms in the last big section of this course. Here is a diagram that adds these randomized classes to the "landscape":



Note differences from the unbounded computability case: NP intersect co-NP is not known (or believed) to equal P, and the quantifiers are *length-bounded* by a polynomial.

means $A \leq_m^p B$

Known: EXP $\neq$ P, PSPACE $\neq$ NL, L $\neq$ REG

(and EXP $\neq$ REC $\neq$ RE, etc.)