CSE696 Week 3: Diagonalization and Reductions and Relativizations

Structure of Reductions

Two more "structural" complexity notions will build a framework for reducibility relations.

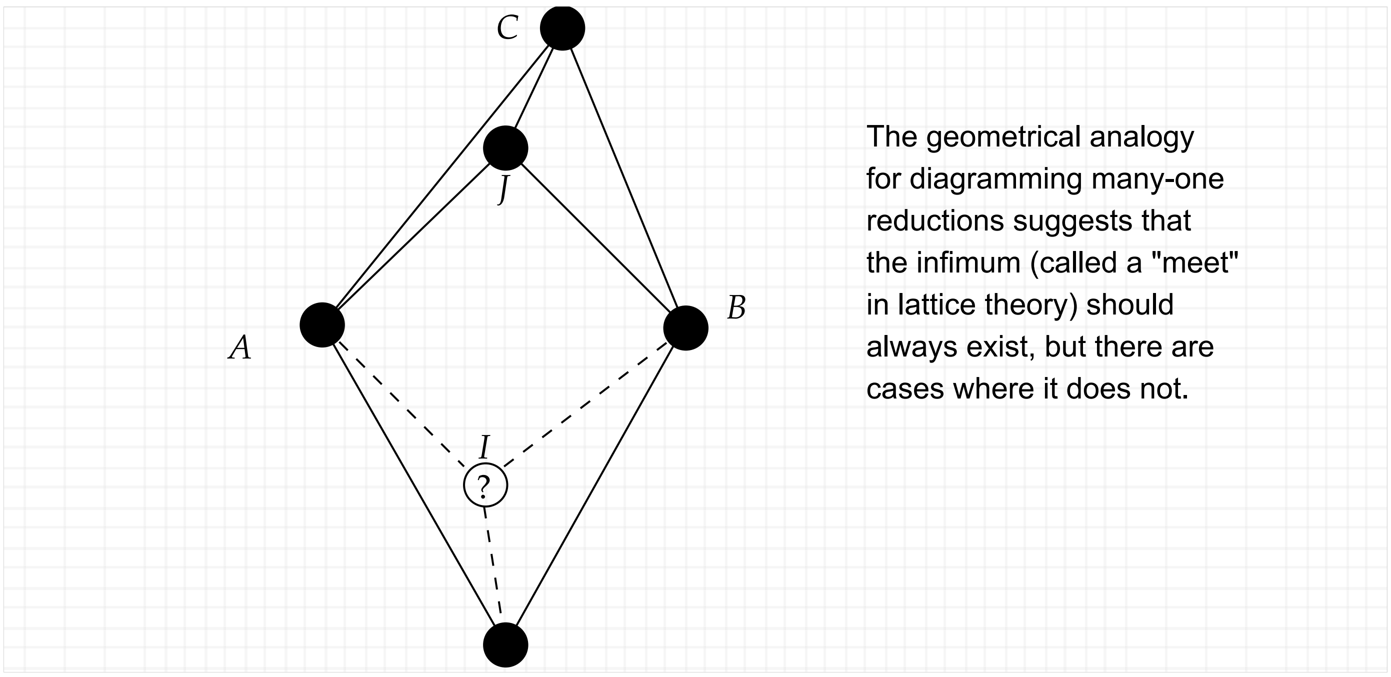
Definition 1: The **join** of two langauges A and B is $A0 \cup B1 = \{x0 : x \in A\} \cup \{y1 : y \in B\}$.

Often the join is written $A \oplus B$ although that can confuse with exclusive-or for the symmetric difference of A and B (which, however, I prefer to write as $A \triangle B$). It is immediate that $A \leq_r A \oplus B$ and $B \leq_r A \oplus B$ for basically any reducibility relation $\leq_r$, because all we have to do is tack on a 0 or a 1 to the string x given in the reduction. The key fact about the join is:

Lemma 1: For basically any reducibility $\leq_r$, not just $\leq_m^p$ or $\leq_T^p$, if A, B, C are any languages such that $A \leq_r C$ and $B \leq_r C$, then $A \oplus B \leq_r C$.

Proof: Given any string x, if $x = \epsilon$ then we know $x \notin A \oplus B$, so we apply the presumed fixed action of the reduction when we know the given string is not in the source language. Otherwise, either $x = y0$ and belongs to $A \oplus B$ if and only if $y \in A$, or $x = y1$ and belongs to $A \oplus B$ if and only if $y \in B$. In the former case, we apply the reduction from A to C; in the latter case, we apply the reduction from B to C. For basically any $\leq_r$, the code managing the two potential applications of a $\leq_r$ reduction belongs to the same class of functions or (oracle) machines that defines $\leq_r$ to begin with. So we get $A \oplus B \leq_r C$. ☒

The upshot is that $A \oplus B$ is a least upper bound for the reductions: it reduces to anything that both A and B reduce to. Technically speaking, this makes the partially ordered structure of (equivalence classes of) languages under $\leq_r$ into an **upper semi-lattice**. Put another way: given any decidable languages A and B, there is always a language J such that $A \leq_r J$ and $B \leq_r J$, and whenever C is a language such that $A \leq_r C$ and $B \leq_r C$, we have $J \leq_r C$. The language J can always be taken as the join of A and B.



The geometrical analogy for diagramming many-one reductions suggests that the infimum (called a "meet" in lattice theory) should always exist, but there are cases where it does not.

Now here is a mirror-image question:

Research Question: Given any decidable languages A and B, is there always a language I such that $I \leq_r A$ and $I \leq_r B$, and whenever C is a language such that $C \leq_r A$ and $C \leq_r B$, we have $C \leq_r I$?

Such a language I, if it exists, could be called an "infimum" of A and B. Well, for many known reducibilities, this is known to fail. But can you define a reasonable $\leq_r$ so that it holds? And even for polynomial-time many-one reductions $\leq_m^p$, the cases of A, B that lack an infimum are somewhat specialized. Let's try a natural case:

Question': Do SAT and TAUT have an infimum under $\leq_m^p$?

In this case, any language I such that $I \leq_m^p SAT$ and $I \leq_m^p TAUT$ belongs to **NP** $\cap$ **co-NP**. And every language C in **NP** $\cap$ **co-NP** has that property. So this is equivalent to asking whether **NP** $\cap$ **co-NP** includes a language I such that for every C in **NP** $\cap$ **co-NP**, $C \leq_m^p I$. In other words:

Proposition 2: SAT and TAUT have an infimum under $\leq_m^p$ if and only if **NP** $\cap$ **co-NP** has a complete set under $\leq_m^p$.

[Now, there are oracles X relative to which $\mathrm{NP}^X \cap$ **co-**NP^X does not have complete sets, but NP^X always has a complete set under $\leq_m^p$ (where the reduction function does not need to consult X). It follows that this set and its complement do not have an infimum under $\leq_m^p$. But this is trying to fly before we can jump---it will take a few weeks before we define and use "SAT^X" for arbitrary oracles X. This does warn that **Question'** has "barriers" to being answered, but maybe the flexibility to seek an inspired formulation of $\leq_r$ makes the "**Research Question**" fair game.]

The second notion is a language that reduces to $A \oplus B$ but is not an infimum. My name and notation are not standard, but the concept underlies the strongest "silly" results.

Definition 2: The **splice** of two languages A and B by a third language E, which we'll write as $E||(A,B)$, is the language $(A \cap E) \cup (B \cap \sim E)$.

The intent of E is to be an "easy" langauge (not just in polynomial or linear time but even notions of sub-linear time) but "extremely gappy". Having $E \in$ P makes the splice $\leq_m^p$-reduce to $A \oplus B$. "Gappy" means that there are long intervals of lengths $n_0 \dots n_1$ on which E has no strings---and long intervals on which it includes every string. That makes $E||(A,B)$ imitate A for long intervals of lengths alternately with imitating B. If A is easy but B is hard, then $E||(A,B)$ will also be hard---but the long intervals where it looks like A will make it difficult to prove that it is not a finite variation of A, which would make it easy after all.

Diagonalization and Ladner's Theorem

The following theorem was proved by Uwe Schöning as a generalization of Ladner's Theorem and its sequels. I tweaked it a little to become the following, called the "Uniform Diagonalization Theorem" (UDT):

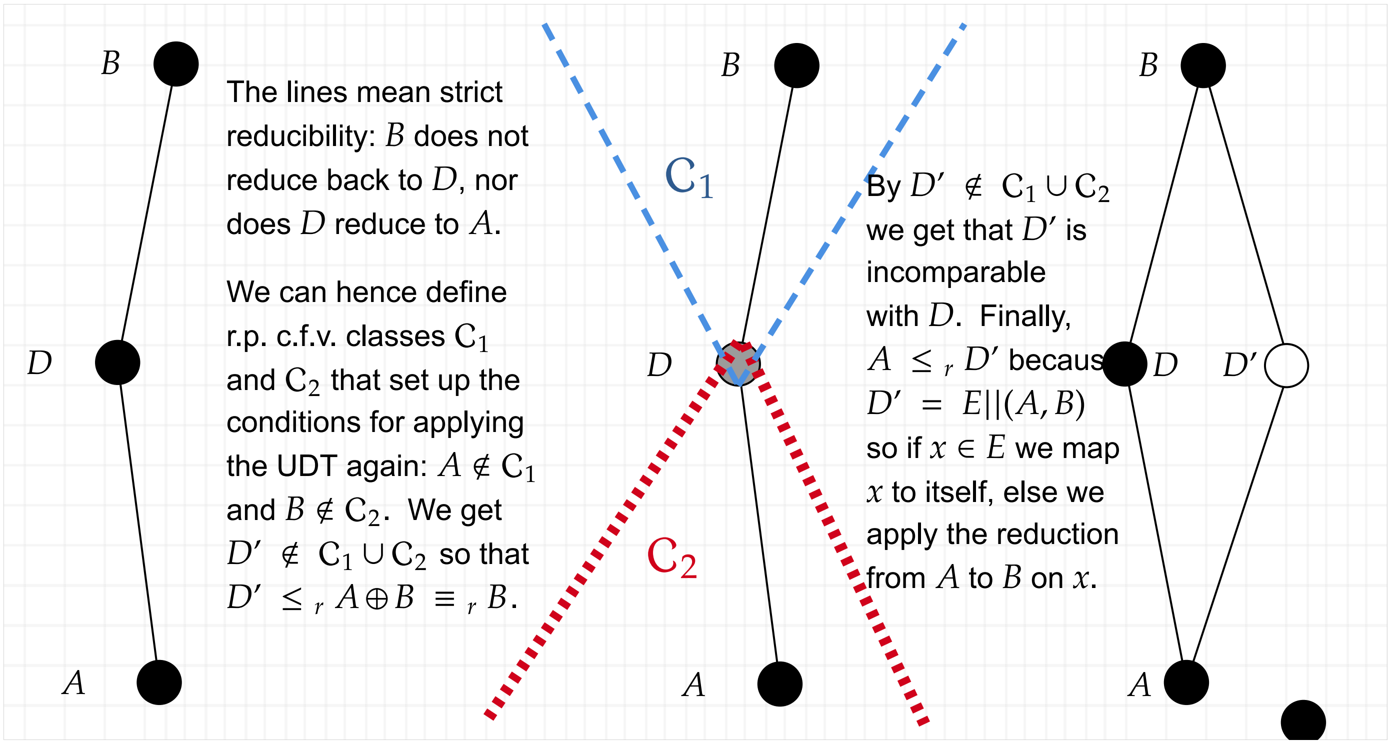
Theorem 3: Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be r.p. c.f.v. classes, and let A, B be decidable languages such that $A \notin \mathcal{C}_1$ and $B \notin \mathcal{C}_2$. Then we can find $E \in$ DTIME$[n+1]$ such that $E||(A,B)$ is in neither $\mathcal{C}_1$ not $\mathcal{C}_2$.

Before proving it, let's show how the conclusion of Ladner's Theorem follows: Suppose P $\neq$ NP. Then $A = SAT$ does not belong to $\mathcal{C}_1 =$ P and $B = \emptyset$ does not belong to $\mathcal{C}_2 =$ NPC. Then the language $D = E||(A,B)$ is neither in P nor NP-complete, but it $\leq_m^p$ reduces to $SAT \oplus \emptyset$, so it belongs to NP. Hence D is NP-intermediate.

Proof: We will first describe a process while "looking forward", then we will view the same process "looking back." Let $[Q_j]$ be the presentation of $\mathcal{C}_1$ and $[R_k]$ that of $\mathcal{C}_2$. We begin in "accepting mode" by looking for the least string y_1 such that $A(y_1) \neq Q_1(y_1)$. By $A \notin \mathcal{C}_1$ there must be such an y_1, indeed by $\mathcal{C}_1$ also being c.f.v., there must be infinitely many. We also keep count of the number t_1 of steps until such y is found. At that moment, a "genie" defines E to include all strings of length up through t_1. Then we switch over to "rejecting mode" and seek the first z_1 of length $t_1 + 1$ or higher such that $B(z_1) \neq R_1(z_1)$. Again there must be one, and we take t_1' to be the total elapsed number of steps upon finding and verifying it. Then the genie defines all strings of length $t_1 + 1$ through t_1' to be non-members of E, i.e., members of $\sim E$. Then we switch back to "accepting mode" in order to seek y_2 of length $t'_1 + 1$ or higher such that $A(y_2) \neq Q_2(y_2)$ and lake t_2 to be the timestamp upon finding and verifying it. Then the search for z_2 such that $B(z_2) \neq R_2(z_2)$ is commenced from length $t_2 + 1$. This process proceeds alternating forever. The language E it creates is such that $(E \cap A) \cup (\sim E \cap B)$ preserves all differences from every Q_j and R_k machine, so the language D is not in $\mathrm{C}_1 \cup \mathrm{C}_2$.

To tell the complexity of E, now we do the looking back: On any input x, take $n = |x|$ and run the forward process for n steps. If the process is in accepting mode at step n, then accept x, else reject x. This takes $n + 1$ steps and defines the same language E, because of how the "genie" extends the same accept mode or reject mode to include all string lengths out to the number t_i or t'_i of steps at the end of the stage in the process that includes time n. Thus, the complexity is as stated. ☒

Now we can play more tricks. Let's start with what we get from Ladner's Theorem:



The lines mean strict reducibility: B does not reduce back to D, nor does D reduce to A.

We can hence define r.p. c.f.v. classes C_1 and C_2 that set up the conditions for applying the UDT again: $A \notin \mathrm{C}_1$ and $B \notin \mathrm{C}_2$. We get $D' \notin \mathrm{C}_1 \cup \mathrm{C}_2$ so that $D' \leq_r A \oplus B \equiv_r B$.

By $D' \notin \mathrm{C}_1 \cup \mathrm{C}_2$ we get that D' is incomparable with D. Finally, $A \leq_r D'$ becaus $D' = E||(A, B)$ so if $x \in E$ we map x to itself, else we apply the reduction from A to B on x.

We can continue in merry fashion: We can make as many mutually incomparable languages $D_3, D_4, D_5, \ldots$ as we wish between A and B. We can apply Ladner's Theorem to place other languages properly between these and the endpoints A and B. Slightly more subtle ways of defining C_1 and C_2 at each stage can create any desired finite pattern of which languages reduce to and from a new language D'' and which don't (provided this is consistent with the pre-existing relations). The upshot is a "general nonsense" noted by lots of people circa 1980:

Theorem 4: For any reasonable and effective reducibility relation $\leq_r$, every countable partial order $\mathfrak{O}$ can be embedded into the equivalence classes of languages under $\equiv_r$. If $\mathfrak{O}$ has the join property

$(\forall\mathfrak{a}\forall\mathfrak{b})(\exists\mathfrak{c})(\forall\mathfrak{d})[\mathfrak{a} \leq_r \mathfrak{c} \wedge \mathfrak{b} \leq_r \mathfrak{c} \wedge (\mathfrak{a} \leq \mathfrak{d} \wedge \mathfrak{b} \leq \mathfrak{d} \longrightarrow \mathfrak{c} \leq \mathfrak{d})]$ then the image of this embedding also has that property (i.e., is an *upper semi-lattice* too). ♣

[Can you embed every *full lattice* among languages such that every pair has a greatest lower bound? Klaus Ambos-Spies investigated that. I forget how far the answer goes---we all soon realized it would not go as far as resolving P versus NP.]

This also proves that a nontrivial difference of r.p. classes is generally not r.p. For example:

Theorem 5: If NP $\neq$ P, then NP $\setminus$ P is not recursively presentable.

Proof: Else, with $\mathcal{C}_1 =$ NP $\setminus$ P, $A_1 = \emptyset$, $\mathcal{C}_2 =$ P, $A_2 = SAT$ we will get a language $D \notin \mathcal{C}_1 \cup \mathcal{C}_2$ that reduces to SAT, a contradiction. ☒

There is only one theorem of this kind that I proved that could be regarded as surprising beforehand.

Definition 3: Given a recursive enumeration of languages $A_1, A_2, A_3, \dots$ define their "infinite join" (we also said "completion") to be $A_\omega = \{\langle x, k\rangle : x \in A_k\}$.

For example, TQBF is basically the infinite join of the languages B_k which are complete for the respective levels of the polynomial hierarchy. This is the sense in which we said that PSPACE is the "ω-completion" of the polynomial hierarchy. The intuition seems fine there, but can be misleading in other cases: For each k, define VC_k to be the language of undirected graphs that have a vertex cover of size k. Then each language VC_k belongs to DTIME$[O(n)]$ (where the "O" hides a factor proportional to 2^k), but their completion is the NP-complete Vertex Cover problem. Likewise CLIQUE and DOMINATING SET and many other NP-complete problems break down this way.

The other notion needed to formulate the infinite case of the UDT is that of a "recursive presentation of infinitely many recursive presentations" but that is readily left to the imagination.

Theorem 6: If $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \dots$ is a recursive presentation of r.p. c.f.v. classes and $A_1, A_2, A_3, \dots$ is a recursive enumeration of decidable languages such that for each k, $A_k \notin \mathcal{C}_k$, then (for any reasonable effective reducibility $\leq_r$) we can build a language $D \notin \cup_k \mathcal{C}_k$ such that $D \leq_r A_\omega$.

The proof is a moderately straightforward extension of that of the UDT: instead of alternating "accept mode" and "reject mode", we alternate being "like A_k" for a sequence of k that could go 1-2-1-2-3-2-1-2-3-4-3-2-1-2-3-4-5-4-3... So long as every k comes up infinitely often, we make D different from the languages of all the machines presenting $\mathcal{C}_k$. By "looking back", we can arrange that the mapping h from a string x of length n to the "k" that is in effect after n steps of the process is computable in linear time. The language we get is then the simple "infinite splice"

$$D = \{x : x \in A_k \textit{ where } k = h(x)\},$$

which basically reduces to A_ω via h and the pairing function. The next corollary applies only the fact that for every decidable language A, the class of languages that are finite variations of A is r.p.

Corollary 7 (surprising?): It is impossible to define a recursive presentation $S_1, S_2, S_3, \ldots$ of PSPACE without there being k such that $L(S_k)$ is a finite variation of the language B_k above. Likewise, every recursive presentation $[N_k]$ of NP has k such that $L(N_k)$ is a finite variation of the language VC_k. This snags you no matter how you define the formal encoding of the languages B_k and VC_k, etc.

Proof: With C_k as the finite variations of $L(S_k)$ for each k, denying the conclusion sets up the condition $B_k \notin \mathrm{C}_k$. The language D that pops out thus diagonalizes out of PSPACE, but it reduces to $B_\omega \equiv_r TQBF$, so it stays in PSPACE. ⊠

Are there further interesting applications? It can be molded into a kind of fixed-point theorem "up to finite variations" but I didn't find anything concrete from it. It is like an all-carbs, no-protien diet.

The one route to greater significance that I can map out is to formulate and prove plausible general conditions under which the languages D cannot be *downward self reducible* in the way that SAT is: $\phi \in SAT \iff \phi[x_1 = 0] \in SAT \vee \phi[x_1 = 1] \in SAT$. Note that when we apply Ladner's situation with $A \in$ P, the language D will have vastly long intervals where $x \in D \iff x \in A$, where whether x is in the interval (i.e., whether x is in the easy splicing language E) is decidable in $O(|x|)$ time. On these intervals, D is as easy as A and E together. When A is chosen to be $\emptyset$, the long intervals are empty, so that D is mega-"gappy", but we should allow for A to be any language in P. Then let us say: "D masquerades as a language in P for vast intervals of instance lengths."

Question: Can we prove that NP $\neq$ P implies that SAT cannot masquerade as a language in P for vast intervals of instance lengths?

Note that SAT has many easy instances. However, it is also believed to have hard instances ϕ of "all" input lengths under natural encodings. At least one of the self-reduction step-downs $\phi_0 = \phi[x_1 = 0]$ and $\phi_1 = \phi[x_1 = 1]$ from such a ϕ must also be an almost-as-hard instance. What happens when further step-downs hit a vast easy interval of instance lengths? The intuition says they cannot without ϕ becoming easy, but the mechanics are trickier. The reason for further interest is that there are theorems to the effect that if NP $\neq$ P is not provable in certin logical theories (that "cheat", IMHO), then SAT must masquerade as a language in P for vast intervals of instance lengths. So this would argue for NP $\neq$ P to be (quasi-)provable, or false.

The obstacle to the question, however, is that in "relativized worlds B" we can prove NP$^B \neq$ P^B via languages L^B that are mega-gappy. Now we hook up with Arora-Barak, section 3.5.

The Lazy Separating Oracle

Theorem 7: We can build arbitrarily sparse decidable languages B such that $\mathsf{NP}^B \neq \mathsf{P}^B$.

Proof: We can take a single recursive presentation $[P_k]$ of polynomial-time bounded oracle TMs such that for all languages B, $\left\{L\left(P_k^B\right)\right\} = \mathsf{P}^B$. For any language B, define the "oracle-dependent language"

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

This is a **tally language**---that is, a subset of 0^*---and always belongs to NP^B, indeed to $\mathsf{NP}[B]$. To show $\mathsf{NP}^B \neq \mathsf{P}^B$, we build B in stages so that for each k, $L^B \neq L\left(P_k^B\right)$. We can handle the OTMs P_k one at a time "with arbitrary leisure": Let $p_k(n)$ be the polynomial running time of P_k. Then $p_k(n)$ is also a limit on the number of different query strings y of length n that P_k^C can submit on input 0^n, for any oracle C, and is also a bound on the length of any query at all. So suppose $C =$ the construction of the language B thus far, having included at most $k-1$ strings to diagonalize against the OTMs P_1 through P_{k-1}. Take n large enough so that $p_k(n) \ll 2^n$ and n is greater than the length of any query at a previous stage. Indeed, take n as vastly large as we wish...

Once having settled on a choice, run the computation $P_k^C\left(0^n\right)$. If it accepts, do nothing---leave C as it was, so that the language B will have no strings of length n. This will ensure that for the final oracle B, $L^B \neq L\left(P_k^B\right)$. If it rejects, then by $p_k(n) \ll 2^n$ there must be some string y of length n that was not queried. Adding y to B then does not change the oracle computation, but it makes the "reject" answer false. So again we will have $L^B \neq L\left(P_k^B\right)$. We thus bring this about for all k, so the final oracle B---for which we can make both B and L^B mega-gappy---gives $L^B \notin \mathsf{P}^B$. Thus we get $\mathsf{NP}^B \neq \mathsf{P}^B$. ☒

Combined with $\mathsf{NP}^A = \mathsf{P}^A$ this says that NP versus P "relativizes both ways." There are many such results in complexity theory, and in my diet analogy, they are "sugars." Much of this course will, however, gain concreteness from the idea of making an "oracle" more active in the role of a *prover* in interactions with machines.

Relativizing Circuits and SAT

Boolean circuits can also be relativized to oracle languages A, where wlog. $A \subseteq \{0,1\}^*$. An A-**gate** has some number m of input wires and outputs 1 iff the string $u \in \{0,1\}^m$ held by the wires belongs to A. Now we want to verify that the bedrock simulation of time $t(n)$ bounded TMs M by $O\left(t(n)^2\right)$-sized circuits carries over to OTMs and oracle circuits. [This is short of asking about the $O(t(n)\log t(n))$-sized simulation.]

It is convenient to consider M to be physically a 1-tape TM but to interleave a virtual oracle tape with its regular tape(s). Let us also suppose that cell 0 is reserved to hold the result of the oracle call and M always scans that cell when entering its query state. The query y is determined to be the longest binary contents of cells $2, 4, 6, 8, \dots$ (terminated by a blank cell or whatever). The output $A(u)$ is written to cell 0 while the other even-numbered cells are unchanged.

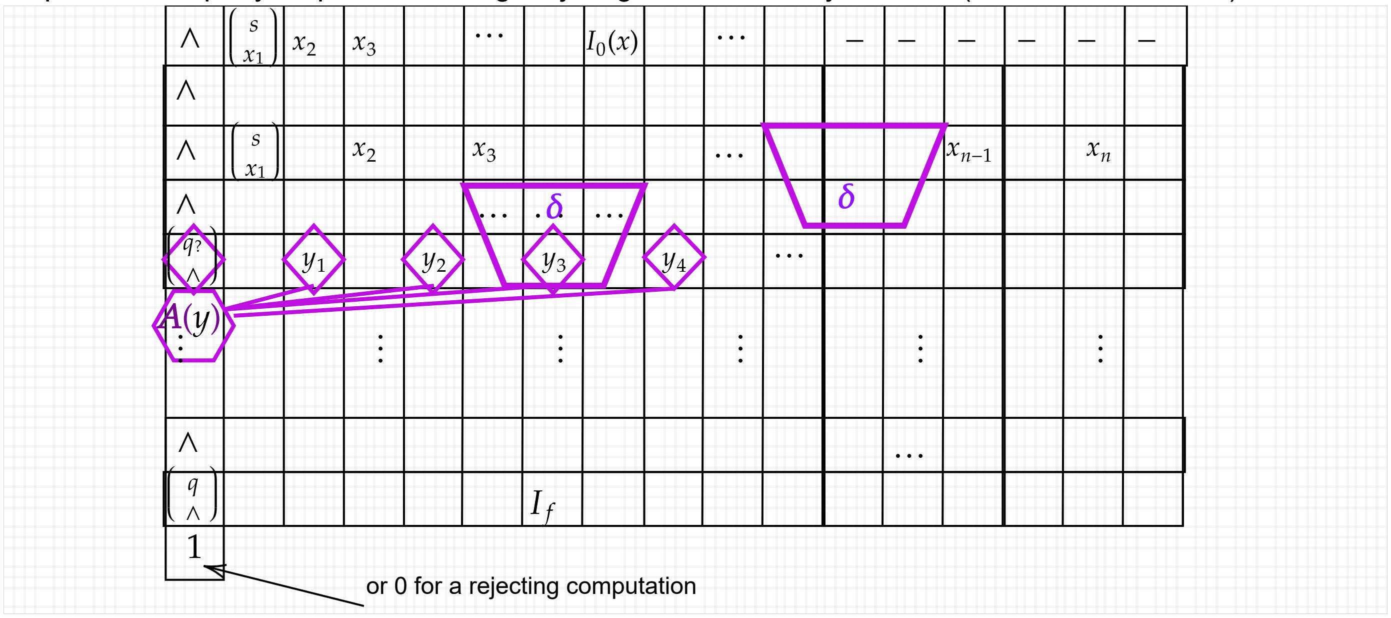
[Footnote 1: This convention does not run afoul of the problem with allowing the query string to be preserved on the oracle tape. That issue comes when M can append a bit to y and re-submit the new string y' in the next step. For example, you could have the oracle

$$SAT' = \left\{\phi, a_1 a_2, \cdots a_i : \phi\left[x_1 \leftarrow a_1, x_2 \leftarrow a_2, \dots, x_i \leftarrow a_i\right] \in SAT\right\}.$$

This oracle can be used to construct a satisfying assignment to $\phi(x_1, \dots, x_n)$ whenever ϕ is satisfiable. If the query portion $a_1 \cdots a_i$ is preserved at each query step, then the OTM would run in time $O(n)$, thus seeming to give a linear-time reduction from search to SAT'. But a linear-time algorithm for SAT' in place of the oracle would not yield a linear-time algorithm for computing the assignment. IMHO, if one defines "linear-time Turing reducibility" then it should preserve linear time computability from oracle to target function. The only way I know to do this is to erase the query after every call, or position the head where the query cannot be elongated.]

Footnote 2: Historically, this issue is dwarfed by that of whether the oracle tape should count against the space bound.]

This easily brings the oracle into the relation $I_{t-1} \vdash_{M^A} I_t$ between IDs. We can program fixed circuitry for this relation including an A-gate with output to cell 0. The A-gate physically needs $\Theta(t)$ input wires at step t since the query could be that long---we may suppose that shorter queries are terminated by recognizable blanks. We need not arrange M to be oblivious in order to use the "bedrock" simulation: if step t is not a query step then the A-gate just gives the identity function (in cell 0 and overall).



Thus we can simulate $M^A(x)$ on inputs x of length n via $O\left(t(n)^2\right)$-sized oracle circuits $[C_n]$, where each C_n has n input "gates" (each of which can branch into multiple input *wires*) and is otherwise composed of binary NAND gates and t-ary A-gates. If M^A is verifying the relation $R^A(x,y)$ defining a language in NP^A---which equals $\mathsf{NP}\left[\mathsf{P}^A\right]$---then each C_n has n input gates $x_1, \ldots x_n$ and also p input gates $y_1, \ldots, y_p$ for the potential witness string y. We have proved:

Theorem: For all oracles A, every language L in P^A has polynomial-sized circuits C_n of simple Boolean gates and A-gates, such that for all x, $x \in L \iff C_n^A(x) = 1$, with $n = |x|$. Moreover, the function from n to C_n is computable in polynomial time (depending on the polynomial running time of a single-tape OTM accepting L).

The latter clause is called **polynomial-time uniformity** and sinply gives another way of defining P^A. Without the clause, we have the notation P/poly for languages accepted by circuit families $[C_n]_{n=1}^{\infty}$ where the C_n have size $n^{O(1)}$. For not-necessarily-uniform oracle circuits the notation is $\mathsf{P}^A/\mathsf{poly}$.

Now to translate the circuit into a Boolean formula, for each output wire w of a NAND gate with inputs u, v we get the equation

$$w = NAND(u,v)$$

and for an A-gate with output wires v_o connecting from cell 0, we get

$$v_o = A(u_1, \ldots, u_m).$$

The collection of these "equational clauses", together with (w_o) for the output wire w_o, defines the corresponding instance of SAT^A. We can use v_o in equations

$$w = NAND(u,v_o) \quad \equiv \quad (u \vee w) \wedge (v_o \vee w) \wedge (\bar{u} \vee \bar{v}_o \vee \bar{w}).$$

An alternate style of relativizing SAT is to treat $A(u_1, \ldots, u_m)$ as a literal in clauses, *viz.*:

$$(u \vee w) \wedge (A(u_1, \ldots, u_m) \vee w) \wedge \left(\bar{u} \vee \overline{A(u_1, \ldots, u_m)} \vee \bar{w}\right).$$

This avoids having to treat equations separately but creates double-decker literals in clauses. I will go completely the other way: I will prefer to regard SAT as a problem about logical equations (which will be broadened to algebraic equations and then to equations from quantum circuits). Then it is most natural to keep the potential oracle calls $v_o = A(u_1, \ldots, u_m)$ as equations. Having thus implicitly settled on a definition of "relativized SAT", that is, the language SAT^A for any language A, we can state the relativized Cook-Levin theorem and some corollaries.

Theorem: For all languages A, SAT^A is complete for NP^A under $\leq_m^p$.

Proof: SAT^A is in NP^A since we can use A to evaluate the equations with $A(\cdots)$. Let any $L \in$ NP^A be given, then there is a relation $R(x,y) \in$ P^A with associated length-bounding polynomial $p(n)$ such that for all x, $x \in L \iff (\exists^p y)R(x,y)$. As detailed above, we can take a single-tape OTM M deciding $R(x,y)$ in polynomial time with oracle A and simulate M by a uniform family of oracle circuits $[C_n]$. Then for each n we get a formula ϕ_n^A such that when the bits of x are substituted for the variables $x_1, \ldots, x_n$, there is an assignment to the variables $y_1, \ldots y_{p(|x|)}$ and the other wire variables that satisfies every equation including $w_o = 1$ for the output wire, if and only if $C_n(x) = 1$, which is if and only if $x \in L$. The mapping from x to ϕ_n^A is computable in time $n^{O(n)}$ without recourse to A, since the equations $v_o = A(u_1, \ldots, u_m)$ as part of ϕ_n^A are just syntax. Thus $L \leq_m^p SAT^A$. ☒

Corollary: For all $k \geq 1$, the language B_k is complete for $\sum_k^p$ under $\leq_m^p$, and the language B_k' of true Π_k propositional sentences if complete for $\prod_k^p$ under $\leq_m^p$.

Proof: We have $\sum_k^p =$ NP$^{B_{k-1}}$ $=$ NP$^{B'_{k-1}}$ $=$ NP$[B'_{k-1}]$ for all k by the polynomial hierarchy theorem. Thus, $SAT^{B'_{k-1}}$ is complete for $\sum_k^p$ by the relativized Cook-Levin theorem. Is saying that $SAT^{B'_{k-1}}$ is equivalent to B_k an acceptable handwave...? [Maybe full rigor requires unwinding the details of the proof of the polynomial hierarchy theorem again. Anyway, ☒]

Corollary: Whenever NP^A $\neq$ P^A, we can construct languages in NP$^A \setminus$ P^A that are not NP^A-complete; indeed, we can embed every countable upper semiliattice between SAT^A and $\varnothing$ using only "gappy" subsets of SAT^A.

It also follows that there is a single *universal* countable upper semilattice, namely the one formed by the structure of decidable languages under $\leq_m^p$ to begin with. In this sense, universality is "a dime a dozen."

[There was a question during the lecture about heat loss in computing functions $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$, to which I noted that the expanded function of $m+n$ arguments

$$F(x_1, \ldots, x_n, a_1, \ldots, a_m) = (x_1, \ldots, x_n, a_1 \oplus y_1, \ldots, a_m \oplus y_m) \; ,$$

is invertible. This is the basic idea of *reversible computation* and will be a main ingredient of quantum computing.]

The Karp-Lipton Theorem

What happens if SAT itself has polynomial-sized circuits $[C_n]$? We can relativize this question, too, to any oracle A. The fact and proof become a nice exercise in transposing a Π_2 logical definition into a Σ_2 one.

Theorem: If $SAT \in$ P / poly then $\sum_2^p = \prod_2^p$ (so PH collapses to $\sum_2^p \cap \prod_2^p$).

This is an example of a theorem of the kind that Avi Wigderson lampooned as "if pigs can whistle, then horses can fly." Neither side of the implication is believed. However, they can be rescued from being counterfactual by relativizing them:

Theorem': For all oracle languages A, if $SAT \in$ P^A / poly then $\sum_2^{p,A} = \prod_2^{p,A}$.

We will prove the unrelativized form, but it becomes a straightforward exercise to see that the proof *relativizes*. A key point is that we don't suppose the ability to *find* for each s the circuit C_s that solves SAT instances of size s, else we would have $SAT \in$ P, which of course collapses the hierarchy all the way to P. Rather the hypothesis is that among the exponentially many possible circuits of a given size s, a circuit C_s that is correct on size-s instances *exists*. We still need to spend a $\forall$ quantifier to *verify* this circuit in order to fixate its usability.

The second key point is not counterfactual at all. The self-reducibility of SAT gives every supposed decider C_s for it a "self-proving" property. If we want to prove that $C_s(\psi) = 1$ really signifies that ψ is satisfiable, then we can run C_s on $\psi_0 = \psi[x_1 = 0]$ and $\psi_1 = \psi[x_1 = 1]$. For self-consistency at least one of $C_s(\psi_0)$ and $C_s(\psi_1)$ must give 1, and we can choose it and recurse as in the self-reduction procedure. If C_s is correct, then at the end we get a satisfying assignment for ψ. We therefore need no further quantification. Thus we will be able to switch a $\forall\exists$ quantification defining an arbitrary language in $\prod_2^p$ into $\exists\forall$ form, which gives the conclusion.

Proof: Let any Π_2 predicate $Q(x) = \forall^p y \exists^p z R(x,y,z)$ be given. This gives us a polynomial-time computable function $f(x,y) = \phi_{x,y}$ such that $\phi_{x,y} \in SAT \iff \left(\exists^p z\right) R(x,y,z)$. Given x of length n, there is a polynomial limit $s(n)$ on the size of $\phi_{x,y}$, and we may conveniently suppose that all ϕ in the range of $f(x,y)$ over x of length n and y of length (up to) $p(n)$ have size $s = s(n)$. Then:

$$Q(x) \iff (\exists C_s)(\forall \psi)[C_s(\phi_{x,y}) = 1 \wedge [C_s(\psi) = 1 \longleftrightarrow SR(C_s, \psi)\ satisfies\ \psi]],$$

where SR is the self-reduction procedure described just above. This gives a Σ_2-definition of $Q(x)$, and the conclusion follows. ☒