## CSE696 Lectures Week 2, Feb. 2021: Completeness and Diagonalization in the Hierarchies

### Complete Sets in AH

It is much harder to define a language in the arithmetical hierarchy that is not complete for one of the $\sum_k^0$ or $\prod_k^0$ levels, or for $\Delta_k^0 = \mathrm{REC}^{\Sigma_{k-1}^0}$. Note that all decidable languages count as complete for REC, and the numbering scheme identifies this with both $\Delta_0^0$ and $\Delta_1^0$, whereas $\mathrm{REC}^K$ is denoted by $\Delta_2^0$.

- It is a mild exercise to construct a langauge $A \in$ RE that is undecidable yet not complete for RE under $\leq_m$. We will do similar things within complexity classes using $\leq_m^p$.
- Creating an intermediate language under *Turing* reductions $\leq_T$, however, was a major open problem for over a dozen years until solved by the *finite injury piority method* of Friedberg and Muchnik (independently) in 1956-57.
- None of those intermediate languages is "natural" to define. The simplest and most appealing definitions always give complete sets.
- The intuitive reason is the *protean* nature of logic and computation. It is already simplest, so it embeds itself readily into (basically all) other systems.
- We can gain appreciation for this by looking at some more completeness proofs.

**Theorem 1**: $TOT = \{\text{non-oracle DTMs } M\colon M \text{ halts for all inputs}\}$ is complete for $\prod_2^0$.

**Proof**: It is in $\prod_2^0$ since defined by $(\forall x)(\exists t)T(M,x,t)$. Let any $L_S$ defined by $S(x) = (\forall y)(\exists z)R(x,y,z)$ with $R$ decidable be given. To reduce $L_S$ to $TOT$, for any $x$, define $f(x) = M_x$ where the machine $M_x$ behaves as follows: on any input $y$, it tries $z = 0,1,2,\ldots$ and accepts if and when $R(x,y,z)$ holds. Then $x \in L_S \iff M_x$ is total. ☒

**Problem**: How about the langauge of deterministic OTMs that are total for all oracles? The above shows that it is many-one hard for $\prod_2^0$. Does it belong to $\prod_2^0$? (This is where König's Lemma may come in handy.)

**Theorem 2**: $FIN = \{M\colon L(M) \text{ is finite}\}$ is complete for $\sum_2^0$.

**Proof**: It belongs since defined by $(\exists w)(\forall x)\big[|x| \geq |w| \to (\forall \vec{c})\neg T(M,x,\vec{c})\big]$. One of the rules of conversion to *prenex normal form* is that $R \to (\forall x)S(x)$ is equivalent to $(\forall x)\big[R \to S(x)\big]$. So we have $M \in FIN \iff (\exists w)(\forall x)(\forall \vec{c})\big[|x| \geq |w| \to \neg T(M,x,\vec{c})\big]$ with the part in [ $\cdots$ ] decidable.

The language $INF = \{M\colon L(M) \text{ is infinite}\}$ is the literal complement of $FIN$, and we've just shown it to be in $\prod_2^0$. So we need only reduce $L_S$ above to $INF$. The above reduction doesn't quite do that, but we can modify it with an idea called "Looking Back." Make $M'_x$ on input $y$ try the previous $M_x$ on each $y' < y$ first. Only if all those accept does $M'_x(y)$ begin operating on $y$ itself.

The upshot is that if $x \notin L_S$ then some (least) $y_0$ fails, i.e., is such that $(\exists z)R(x, y_0, z)$ fails, so that $M_x(y_0)$ never halts. Then for all $y \geq y_0$, $M'_x(y)$ falls into $y_0$ and so never halts. This makes $L(M'_x)$ finite. Whereas if $x \in L_S$ then $L(M'_x)$ is not only infinite but equals $\Sigma^*$. By the rule $A \leq_m B \iff \widetilde{A} \leq_m \widetilde{B}$, this reduces any given language in $\sum_2^0$ to $FIN$. ☒

## Index sets and Subrecursive Classes

Now $FIN$ is an **index set**, that is, a set of the form $I_{\mathcal{C}} = \{M : L(M) \in \mathcal{C}\}$ for some class $\mathcal{C}$ of c.e. languages. It is $I_{\mathsf{FIN}}$ where $\mathsf{FIN}$ is the class of finite languages. **Rice's Theorem** says that every index set other than $I_{\varnothing} = \varnothing$ and $I_{\mathsf{RE}} = \Sigma^*$ is undecidable. (Note that the subscripted $\varnothing$ is the empty *class* of languages, whereas the other $\varnothing$ is the empty langauge.) This is "weak beer"---we can classify index sets more precisely.

One technical point to note is the definition of $L(M_1) = L(M_2)$ for a given pair of Turing machines $M_1$ and $M_2$. If we know in advance that both $M_1$ and $M_2$ are total, then we have $(\forall x)[M_1(x) = M_2(x)]$ and the part in $[\cdots]$ is decidable, so we get a $\Pi_1$ definition. But if one or both are not total, then we must invoke a further quantification over computations. Then:

$$L(M_1) = L(M_2) \equiv (\forall x)[(\exists c)T(M_1, x, c) \longleftrightarrow (\exists d)T(M_2, x, d)]$$

We cannot simply bring out both $\exists$ quantifiers. But we can write the equivalence in two pieces:

$$(\forall x)[((\exists c)T(M_1, x, c) \land (\exists d)T(M_2, x, d)) \lor ((\forall c')\neg T(M_1, x, c') \land (\forall d')\neg T(M_2, x, d'))]$$

Then condensing gives

$$(\forall x)[((\exists c, d)T(M_1, x, c) \land T(M_2, x, d)) \lor ((\forall c')\neg T(M_1, x, c') \land \neg T(M_2, x, c'))]$$

Now $c'$ is not quantified on the left of the central $\lor$, so we can bring out the $(\forall c')$ first and finally get a $\Pi_2$ predicate:

$$(\forall x, c')(\exists c, d)[(T(M_1, x, c) \land T(M_2, x, d)) \lor (\neg T(M_1, x, c') \land \neg T(M_2, x, c'))] .$$

This is now amazingly hard to read, but it works. So equality of two machines' languages is always $\Pi_2$ at worst. The consequence of interest to us is:

**Proposition 3**: *If* you have a recursive enumeration $[Q_k]$ of machines that generate a class $\mathcal{C}$, then the index set $I_{\mathcal{C}}$ is $\Sigma_3$-definable via $(\exists k)L(M) = L(Q_k)$. ☒

This holds regardless of whether the machines $Q_k$ are total, but that will be our main source of interest:

**Definition 1**: A class $\mathcal{C}$ of recursive languages is **recursively presentable** (**r.p.**) if there is a recursive enumeration $[Q_k]_{k=1}^{\infty}$ of *total* machines such that $\mathcal{C} = \{L(Q_k)\}$.

For example, P and NP are r.p. by their associated "natural" enumerations of machines. The latter's machines $[N_k]$ are nondeterministic, but we can use the exponential-time DTMs $[M_k]$ obtained by a fixed NTM-to-DTM conversion in their place. Perhaps less obvious is that the class NPC of NP-complete languages is r.p.: Use a recursive presentation $[F_k]$ of the class FP of polynomial-time computable functions and ahhh...let's come back to this.

Anyway, every r.p. class $\mathcal{C}$ has $I_{\mathcal{C}} \in \sum_3^0$. And aside from $FIN$ and its complement $INF$, most of them are complete for $\sum_3^0$ under $\leq_m$. Before putting that up for consideration, let's motivate the r.p. notion some more.

## Recursive Presentations and "Looking Back"

Here are two other definitions, the second of which is a tacit admission that asymptotic complexity ignores concrete bounds. I use the strict definition of DTIME$[t(n)]$ as languages accepted by TMs $M$ such that for all $x$, $M(x)$ halts within $t(|x|)$ steps. Some use the lax definition that applies this only "for sufficiently large" $x$. A machine $M$ that abides by the latter can always be converted to the former by giving it a "finite lookup lable": Suppose $n_0$ is the constant so that $M(x) \downarrow$ within $t(n)$ steps whenever $n = |x| \geq n_0$. For the finitely many $x$ of length below $n_0$, we put the yes/no answers into tabular form as a binary tree and encode that as extra states that govern the first up-to-$n_0$ steps of $M(x)$ on any $x$. Since we assume $t(n) \geq n+1$ for any running-time function, the new machine $M'$ runs in time $t(n)$ strictly while accepting the same language.

**Definition 2**: A class $\mathcal{C}$ is **bounded** if there is a computable function $t(n)$ such that $\mathcal{C} \subseteq$ DTIME$[t(n)]$.

**Definition 3**: $\mathcal{C}$ is **closed under finite variations** (**c.f.v.**) if for all $A \in \mathcal{C}$ and $B$ such that the symmetric difference $A \triangle B$ is finite, $B \in \mathcal{C}$.

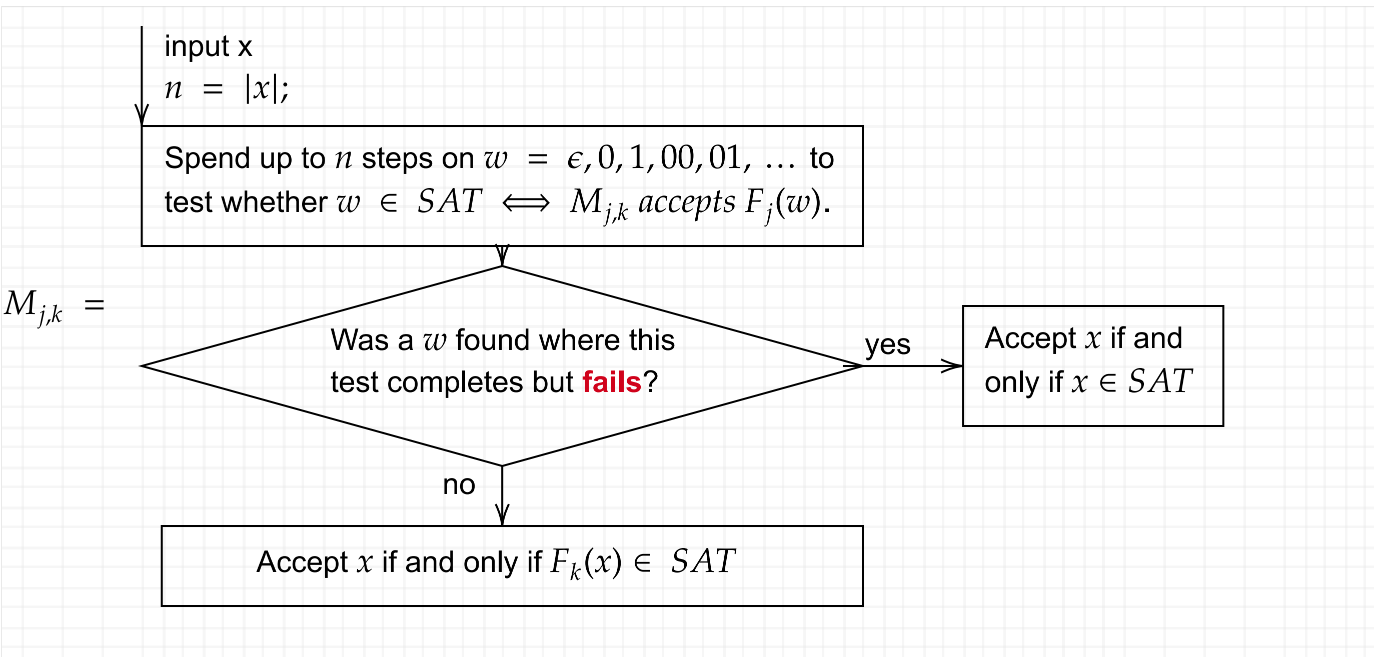
**Lemma 3**: Every r.p. class is bounded.

**Proof**: For all $n$, define $t(n)$ to be the maximum of the time taken by $Q_k(x)$ for all $k \leq n$ and $x$ of length (up to) $n$. Because each $Q_k$ is total, this is a computable function. For all $k$, the running time of $Q_k$ on inputs $x$ of length $k$ and higher is bounded by $t(|x|)$ by definition. Thus it meets the "lax" definition of running in time $t(n)$. As discussed above, the results for $x$ of length up to $k-1$ can be stored in a finite table to create a machine $Q'_k$ that accepts the same language as $Q_k$ and runs in time $|x|+1 \leq t(|x|)$ for those $x$. This not only tells us that every language in $\mathcal{C}$ belongs to DTIME$[t(n)]$, but because the change from $Q_k$ to $Q'_k$ is *effective*, it tells us that the recursive presentation can be changed to $[Q'_k]_{k=1}^{\infty}$ so that each machine obeys the $t(n)$ time bound strictly. ☒

When a class is c.f.v., then we don't even have to care about doing the finite-table patch. Now this comes in handy to show that the NP-complete languages are recursively presentable. Take the presentation $[F_k]$ of FP from above. Our first thought might be to define for each $k$ the machine

$M_k(x)$: compute $y = F_k(x)$ and accept $x$ if and only if $y \in SAT$.

Then $L(M_k) \leq_m^p SAT$ via the polynomial-time function computed by $F_k$. So $[M_k]$ captures the class of languages that polynomial-time many-one reduce to $SAT$---which is just another way to get a recursive presentation of NP. We need to intersect the logic with the condition that $SAT$ reduces to the language in turn. For each pair $j,k$ define $M_{j,k}$ to run as follows:




### Hierarchy Operations and Recursive Presentations

This idea readily translates into something more general: Suppose that $\mathcal{C}$ and $\mathfrak{D}$ are r.p. classes with presentations $[Q_j]$ and $[R_k]$ and that, crucially, $\mathcal{C} \cap \mathfrak{D}$ contains some language $A_0$ *together with all of its finite variations*. Then we can build $M_{j,k}$ on input $x$ to first spend $n$ steps looking for a witness $w = \epsilon, 0, 1, 00, 01, \ldots$ that $Q_j(w) \neq R_k(w)$, so that $L(Q_j) \neq L(R_k)$. If it finds and verifies one within $n$ steps, then $M_{j,k}$ accepts $x$ if and only if $x \in A_0$. Thus $L(M_{j,k})$ becomes a finite variation of $A_0$, but that's OK---it is still a language in $\mathcal{C} \cap \mathfrak{D}$. Now let any language $L$ in $\mathcal{C} \cap \mathfrak{D}$ be given. Then there are machines $Q_j$ and $R_k$ such that $L(Q_j) = L = L(R_k)$. Then the new machine $M_{j,k}$ on whatever input $x$ never finds a bad witness $w$. So $M_{j,k}$, if it finds no bad witness within $n$ steps, is coded to return $Q_j(x)$. Thus, $L(M_{j,k}) = L$, and we conclude that $[M_{j,k}]$ is a recursive presentation of $\mathcal{C} \cap \mathfrak{D}$.

To get a recursive presentation of $\mathcal{C} \cup \mathfrak{D}$, we can just merge together the original machines $Q_j$ and $R_k$ with no extra coding. The "merge" idea extends to infinite unions, provided we have an effective handle on the presentations for each class. Thus, *given* that the individual classes P, NP, $\mathsf{NP}^{\mathsf{NP}}$, ... are each r.p., it follows that their union PH is r.p. As for how to get the individual classes, the proof last week suggests and operator that we will use often.

**Definition 4**: For any class $\mathcal{C}$, define $\mathsf{NP}[\mathcal{C}]$ to be the class of languages $L$ such that for some polynomial $p$ and language $R \in \mathcal{C}$, $L = \left\{x : \left(\exists^p y\right)\langle x, y\rangle \in R\right\}$.

Then $\mathsf{NP} = \mathsf{NP}[\mathsf{P}]$. The operator $\mathsf{NP}[\cdot]$ is *idempotent*, a fancy term for saying $\mathsf{NP}[\mathsf{NP}[\mathcal{C}]] = \mathsf{NP}[\mathcal{C}]$ for every $\mathcal{C}$. This is basically because of how two adjacent $\exists$ quantifiers can be combined into one. But $\mathsf{NP}[\mathsf{co}\text{-}(\mathsf{NP}[\mathcal{C}])]$ gives us something different: by the proof of the equivalence between quantifiers and oracle levels in the "weak PH theorem," it gives $\sum_2^p$. Then iterating the $\mathsf{NP}[\cdot]$ and co- operations gives all of the polynomial hierarchy. The missing pieces we need to show it all to be recursively presentable are:

**Lemma 4**: If $\mathcal{C}$ is r.p. then so are $\mathsf{NP}[\mathcal{C}]$ and co-$\mathcal{C}$.

**Proof**: Let $[R_k]$ present $\mathcal{C}$ by total machines. Then for each $k$, let $Q_k$ be a TM that on any input $x$ tries all $y$ such that $|y| \leq p(|x|)$ (where $p$ is the polynomial length bound in the application of the operator $\mathsf{NP}[\cdot]$) and accepts $x$ if and when $R_k$ accepts $\langle x, y\rangle$. Then, although $Q_k$ will likely run in exponential time, it is still total, and every language in $\mathsf{NP}[\mathcal{C}]$ is captured as $L(Q_k)$ for some $k$, so $[Q_k]$ is a recursive presentation of $\mathsf{NP}[\mathcal{C}]$. [But one thing we can say in-passing is that if $R_k$ runs in polynomial space, then so does $Q_k$, because it needs only the additional space for $y$. This tells us that every level of the polynomial hierarchy stays within PSPACE.]

And for co-$\mathcal{C}$, because each $R_k$ is a total machine, we can complement its accepting and rejecting states to make $R_k'$ so that $L(R_k')$ is the complement of $L(R_k)$. ☒

However, it does not follow that the complement of the class $\mathcal{C}$ itself is r.p. Well, when $\mathcal{C}$ is bounded, the complement of $\mathcal{C}$ (in RE, say) is unbounded. But even if we do a difference $\mathcal{E} = \mathfrak{D} \setminus \mathcal{C}$ of r.p. classes, which stays bounded, we will see that in general $\mathcal{E}$ is **not** r.p. It does, however, still have index set $I_{\mathcal{E}}$ belonging to $\sum_3^0$. The reason is a knock-on effect. Whereas

$L(M) \notin \mathcal{C} \equiv (\forall k) L(M) \neq L(R_k) \equiv (\forall k)(\exists x)[(\exists c)T(M, x, c)\ XOR\ (\exists d)T(R_k, x, d)]$

yields no better than ∀∃∀ in prenex form, we get a leg up by the fact of also defining $L(M) \in \mathfrak{D}$. Let $[R_k]$ present $\mathcal{C}$ and let $[Q_j]$ present $\mathfrak{D}$. Then:

$$L(M) \in \mathfrak{D} \setminus \mathcal{C} \equiv (\exists j)[L(M) = L(Q_j) \wedge (\forall k) L(Q_j) \neq L(R_k)].$$

Now because $Q_j$ and $R_k$ are both total, the $L(Q_j) \neq L(R_k)$ part becomes

$$(\exists j) \ldots (\forall k)(\exists x)[Q_j(x) \neq R_k(x)]$$

where the part in [ … ] is now *decidable*. Since we have already seen that $(\exists j)[L(M) = L(Q_j)$ is a $\Sigma_3$-predicate, this becomes the conjunction of two $\Sigma_3$-predicates, which is a $\Sigma_3$-predicate.

## Structure of PH and a Possible Non-R.P. Class

The $\mathsf{NP}[\cdot]$ operator has one immediate utility: it speeds the proof of the "collapse lemma":

**Lemma 5**: For any $k$, if $\Sigma_k^p = \Pi_k^p$ then $\mathsf{PH} = \Sigma_k^p \cap \Pi_k^p$.

In particular ($k = 1$), if **NP** = **co-NP**, then the whole polynomial hierarchy "collapses" to **NP** ∩ **co-NP.** And of course, if **NP** = **P** then it all collapses to **P.**

**Proof**: For $k = 1$, we start with $\mathsf{NP}[\mathsf{co}\text{-}(\mathsf{NP}[\mathsf{P}])] = \mathsf{NP}[\mathsf{co}\text{-}\mathsf{NP}]$ and apply our hypothesis to make that $= \mathsf{NP}[\mathsf{NP}] = \mathsf{NP}$. So $\Sigma_2^p = \Sigma_1^p$, and we already hypothesized $\Sigma_1^p = \Pi_1^p$ so it all equals $\Sigma_1^p \cap \Pi_1^p$. Further use of co- and $\mathsf{NP}[\cdot]$ just winds up trying to build on the same quicksand. The full proof just replaces "1" by "$k$" here. ☒

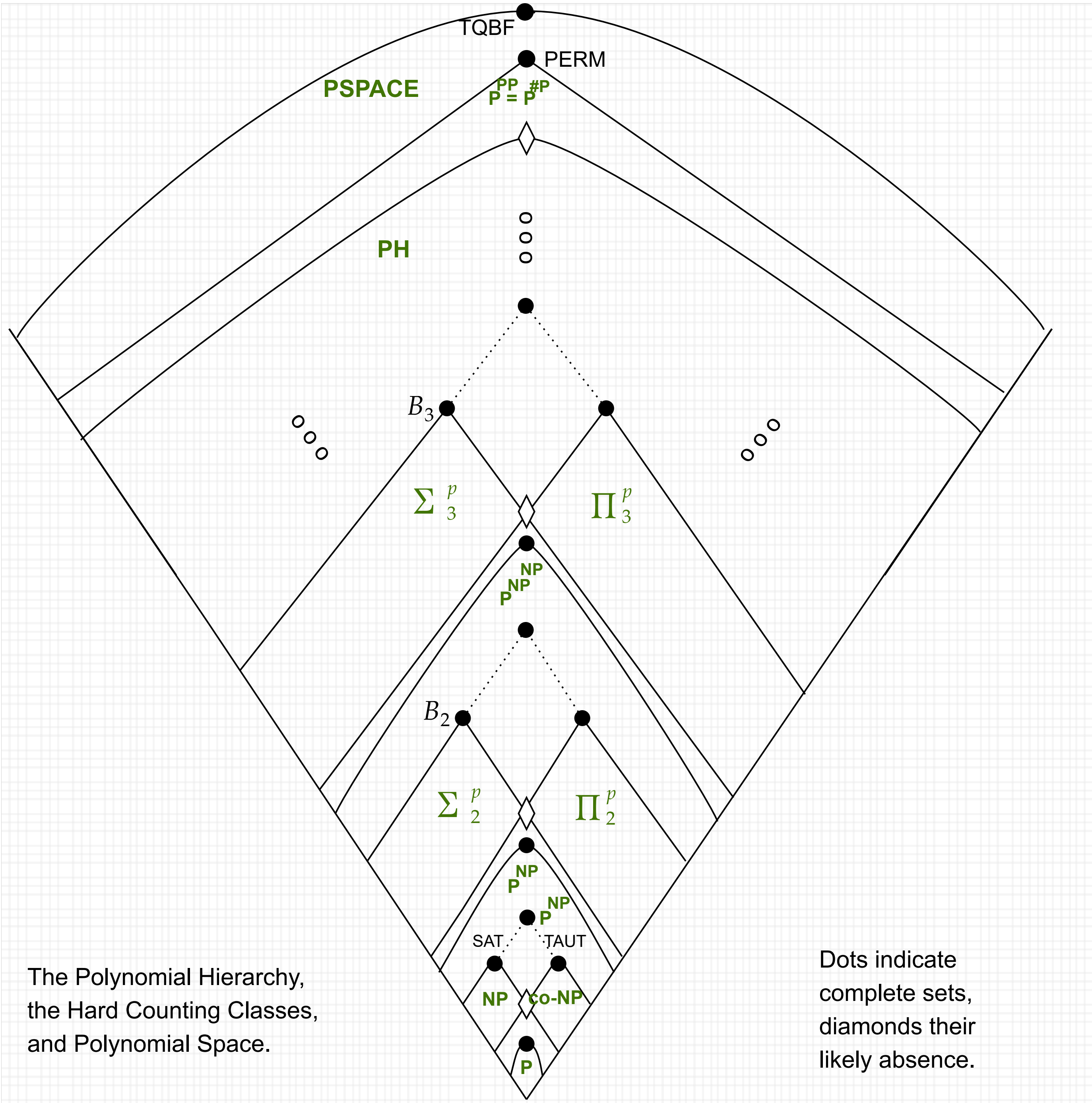
**Corollary 6**: If $\mathsf{PSPACE} = \mathsf{PH}$, then for some $k$, $\mathsf{PSPACE} = \mathsf{PH} = \Sigma_k^p \cap \Pi_k^p$.

Proof: If the TQBF language belongs to PH, then it belongs to $\Sigma_k^p$ for some finite $k$. But TQBF is PSPACE-complete and mapping-reduces to its complement, so $\Sigma_k^p = \Pi_k^p$ follows. ☒

**Hall Of Mirrors Effect**: For any oracle $A$, if $\Sigma_k^{p,A} = \Pi_k^{p,A}$ then $\mathsf{PH}^A = \Sigma_k^{p,A} \cap \Pi_k^{p,A}$.

**"Proof"**: Everything done in CSE596 and so far in this course relativizes!

It is, however, possible to have PH collapse, indeed to have $\mathsf{P} = \mathsf{NP}$, without having PSPACE collapse into it. Here's an attempt to make a picture of the polynomial hierarchy that is more suggestive of its "vital signs":




The Polynomial Hierarchy, the Hard Counting Classes, and Polynomial Space.

Dots indicate complete sets, diamonds their likely absence.

Now we introduce a curious attempt at a class that could have an index set that is not $\Sigma_3$-definable.

**Definition 5** [KWR, 1982]: $\mathsf{H}$ = the intersection of $\mathsf{P}^A$ over all oracles $A$ that make $\mathsf{P}^A = \mathsf{NP}^A$.

**Proposition 7**: $\mathsf{PH} \subseteq \mathsf{H} \subseteq \mathsf{PSPACE}$, and $\mathsf{H} \neq \mathsf{PH} \implies \mathsf{NP} \neq \mathsf{P}$, indeed that the polynomial hierarchy is infinite.

**Proof**: Because $\mathsf{P}^{TQBF} = \mathsf{NP}^{TQBF} = \mathsf{PSPACE}$, we get $\mathsf{H} \subseteq \mathsf{PSPACE}$. If $A$ is any oracle such that $\mathsf{P}^{A} = \mathsf{NP}^{A}$, then by the relativized collapse lemma ("hall of mirrors"), $\mathsf{PH}^{A} = \mathsf{P}^{A}$, so $\mathsf{PH} \subseteq \mathsf{P}^{A}$, so the unrelativized $\mathsf{PH}$ stays inside $\mathsf{P}^{A}$ for every such $A$, so $\mathsf{PH} \subseteq \mathsf{H}$. If the polynomial hierarchy collpases to $\sum_{k}^{p}$ (for any $k$), then the language $B_k$ becomes an oracle relatuive to which $\mathsf{NP}^{B_k} = \mathsf{P}^{B_k}$, so $\mathsf{H}$ equals $\sum_{k}^{p}$ equals $\mathsf{PH}$ in that case. Thus $\mathsf{H} \neq \mathsf{PH}$ makes the whole polynomial hierarchy infinite... ☒

...Which is what we believe, but does this idea help to prove it? I thought once maybe yes, but now we should not be so starry-eyed. [However, it may follow by building on stuff to come next that $\mathsf{H} = \mathsf{PH}$ without any hypothesis, which would kill my idea but would have other interests.]

Proving $\mathsf{H} = \mathsf{PH}$ would also imply that $\mathsf{H}$ is recursively presentable. That in turn would imply that the index set $I_{\mathsf{H}}$, in other words, the language $\{M: L(M) \in \mathsf{H}\}$, belongs to $\sum_{3}^{0}$. Can we give a $\Sigma_3$-definition for $I_{\mathsf{H}}$ without needing any hypothesis? This leads to a second "psych" observation about logic and the arithmetical hierarchy:

1. It is hard to find natural examples of languages that are not complete for some level of AH.
2. It is also hard to think of natural examples of languages that do not belong to $\sum_{3}^{0}$ or $\prod_{3}^{0}$. To (para-)quote Hartley Rogers, whose textbook *Elements of Recursion Theory* is a bellwether in that field, "It is hard for the human mind to grab more than three quantifier alternations at a time. Many lemmas in published mathematics are really ways of enabling the mind to get past a couple more quantifier alternations."

The Rogers quote tends toward $I_{\mathsf{H}}$ being $\Sigma_3$-definable even if $\mathsf{H} \neq \mathsf{PH}$. Even if $\mathsf{H}$ is not r.p., it is the next-best thing as an intersection of classes $\mathsf{P}^{A}$ that are individually r.p. In a topological sense, r.p. classes behave like closed sets---and an intersection of closed sets is closed. The key property of a closed set $C$ in a metric space is that if $a$ is a point not in $C$, then there is an open ball around $a$ that is disjoint from $C$. If $a$ does not belong to an intersection $\cap_i C_i$ of closed sets, then there is a single $C_i$ such that $a \notin C_i$. This means that the key hypotheses of the diagonalization theorem we will use to prove Ladner's Theorem will hold even if we replace an r.p. class by an intersection of them. So $\mathsf{H}$ is "as good as r.p." anyway.

### Structure of Reductions

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

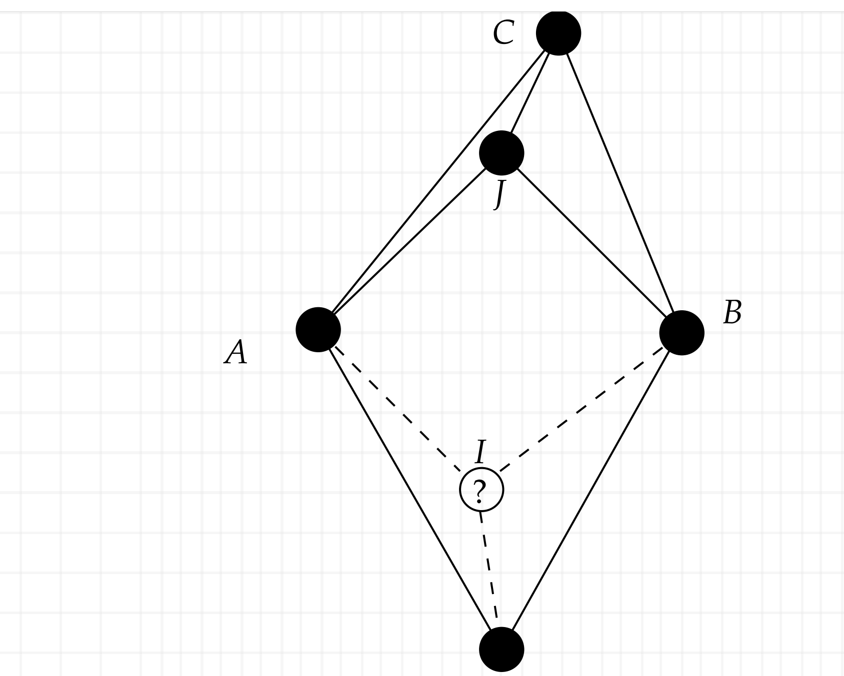
**Definition 6**: The **join** of two langaugaes $A$ and $B$ is $A0 \cup B1 = \{x0 : x \in A\} \cup \{y1 : y \in B\}$ .

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 8**: 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 9**: 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 $\mathsf{NP}^X \cap$ **co-**$\mathsf{NP}^X$ does not have complete sets, but $\mathsf{NP}^X$ 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 7**: 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 $E \in \mathsf{P}$ 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:

**Theorem 10**: Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be r.p. c.f.v. classes, and let $A, B$ be languages such that $A \notin \mathcal{C}_1$ and $B \notin \mathcal{C}_2$. Then we can find $E \in \mathsf{DTIME}[n+1]$ 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 $\mathsf{P} \neq \mathsf{NP}$. Then $A = SAT$ does not belong to $\mathcal{C}_1 = \mathsf{P}$ and $B = \emptyset$ does not belong to $\mathcal{C}_2 = \mathsf{NPC}$. Then the language $D = E||(A,B)$ is neither in $\mathsf{P}$ nor $\mathsf{NP}$-complete, but it $\leq_m^p$ reduces to $SAT \oplus \emptyset$, so it belongs to $\mathsf{NP}$. Hence $D$ is $\mathsf{NP}$-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 $y$ such that $A(y) \neq Q_1(y)$. By $A \notin \mathcal{C}_1$ there must be such an $y$, 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 $x$ 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 $y$ of length $t_1 + 1$ or higher such that $B(y) \neq R_1(y)$. 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 $y$ of length $t_1' + 1$ or higher such that $A(y) \neq Q_2(y)$ and lake $t_2$ to be the timestamp upon finding and verifying it. Then the search for $y$ such that $B(y) \neq R_2(y)$ 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 $\mathcal{C}_1 \cup \mathcal{C}_2$.

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. ☒

Next time, I will finish these notes and move on to section 3.5 of the Arora-Barak draft http://theory.cs.princeton.edu/complexity/book.pdf on the oracle $B$ making $\mathsf{P}^B \neq \mathsf{NP}^B$. Then I will go to sections 6.1 and 6.2. I will review how circuits were used to prove the Cook-Levin Theorem in CSE596 while also showing how circuits can have "oracle gates" to make this all work for "relativized SAT." From Week 4 onward, we will be occupied for awhile with chapter 7, plus section 9.1 and the first two pages of section 9.2.