Toda's Theorem, following Lance Fortnow

Advanced Complexity Test

This test is fairly long. Go fast: say what the essential ideas are; but do the computations right, and do not miss any subtle issue!

To give an idea of the relative difficulties, I'm giving the number of lines of my solution to each question.

1 Relativizations

Recall that an oracle machine is a multi-tape machine with a specific query tape, three extra control states Q, YES and NO. Let A be a language. The semantics of the machine with oracle A is as usual, except that when the machine reaches state Q, it then proceeds to state YES if the contents of the query tape is in A, and to NO otherwise. The *relativized* classes $\mathbf{P}^A$, $\mathbf{NP}^A$, $\mathbf{BPP}^A$, etc., are obtained from their classical counterpart by changing the underlying Turing machine model to the corresponding oracle machine, with oracle A.

For classes $\mathcal{C}$, $\mathcal{C}'$ of languages, and assuming the notation $\mathcal{C}^A$ makes sense, the class $\mathcal{C}^{\mathcal{C}'}$ is $\bigcup_{L' \in \mathcal{C}'} \mathcal{C}^{L'}$. I.e., a language L is in $\mathcal{C}^{\mathcal{C}'}$ iff there is a language $L' \in \mathcal{C}'$ such that $L \in \mathcal{C}^{L'}$.

1. *(2 lines, 1.4%)* Assume $\mathcal{C}'$ has a complete language L' (under polynomial time reductions). Show that, for any $\mathcal{C}$ among $\mathbf{P}$, $\mathbf{NP}$, $\mathbf{BPP}$, $\mathbf{RP}$, $\mathcal{C}^{\mathcal{C}'} = \mathcal{C}^{L'}$.

   One can include the polynomial time reduction inside the computation of the $\mathbf{P}$*, resp.,* $\mathbf{NP}$*,* $\mathbf{BPP}$*,* $\mathbf{RP}$ *machine. An important point is that this included computation is polynomial in the size of the query, which is itself polynomial in the size of the input.*

2 The Zachos Lemma

1. *(7 lines, 4.8%)* Show that $\mathbf{NP}^{\mathbf{BPP}} = \mathbf{MA}$.

   Let $L \in \mathbf{NP}^{\mathbf{BPP}}$*. So there is language* L' *in* $\mathbf{BPP}$ *such that* $L \in \mathbf{NP}^{L'}$*. Let* $\mathcal{M}$ *be a non-deterministic Turing machine with oracle* L' *deciding* L*. We can assume all calls to the oracle to work with queries of a fixed polynomial size.*

Since $L' \in \mathbf{BPP}$, there is a polynomial-time language D such that if $x' \in L'$ then $Pr_r[(x', r) \in D] \geq 1 - 1/2^{p(n)}$ (for any fixed polynomial p, and n is the size of the input—not x'), and if $x' \notin L'$, then $Pr_r[(x', r) \in D] \leq 1/2^{p(n)}$. Build the following **MA** *game. Merlin guesses a trace of $\mathcal{M}$, including the answers of the oracle. Then Arthur checks all the guessed answers to the oracle using his* **BPP** *algorithm, and also checks that the trace given by Merlin is a valid trace and one that accepts. If $x \in L$, then Merlin can provide all the right answers to the oracle, making Arthur accept. If $x \notin L$, and assuming Merlin's trace is valid and accepts, Merlin must have cheated and given a wrong answer to at least one of the oracle calls. The probability that he goes undetected is then at most $\sum_{k=1}^{m} 1/2^{p(n)}$, where there are m calls to the oracle. Since m is itself polynomial, this can be made asymptotically smaller that $1/n^{\ell}$ for any $\ell \geq 1$. So $L \in \mathbf{MA}$.*

An alternative argument works by letting Merlin guess, not a trace of $\mathcal{M}$, but a sequence of $q(n)$ bits, where $q(n)$ is a polynomial exceeding the time complexity of $\mathcal{M}$. Then Arthur simulates $\mathcal{M}$, resolving the i guess it has to make by reading Merlin's ith bit. The error bounds are as above.

The converse inclusion is clear: from D, define L' as the language of pairs (x, y) such that $Pr_r[(x, y, r) \in D] \geq 1/2$, and run the above argument in reverse to obtain the desired error bounds.

2. *(13 lines, 9%)* Using this, show that if $\mathbf{NP} \subseteq \mathbf{BPP}$, then $\mathbf{PH} \subseteq \mathbf{BPP}$. This is the *Zachos Lemma*.

We show that $\Sigma_i^P \subseteq \mathbf{BPP}$ for every i, by induction on $i \geq 1$. The base case is the assumption. In the inductive case, let $L \in \Sigma_{i+1}^P = \mathbf{NP}^{\Sigma_i^P}$. Since $\Sigma_i^P \subseteq \mathbf{BPP}$ by induction hypothesis, $L \in \mathbf{NP}^{\mathbf{BPP}}$, so $L \in \mathbf{MA}$ by the previous question. By Babai's Theorem (Theorem 3.11 in the notes), $L \in \mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$. Now $\mathbf{BP}\cdot$ is a monotonic operator (Lemma 3.7 in the notes), and $\mathbf{NP} \subseteq \mathbf{BPP}$ by assumption. So $L \in \mathbf{BP} \cdot \mathbf{BPP}$. We have already noticed in the notes (see the proof of Theorem 3.18 in the notes, where we used a similar argument) that $\mathbf{BP} \cdot \mathbf{BPP} = \mathbf{BPP}$. Here we only notice that $x \in L$ is decided by testing whether $(x, r) \in L'$ for some language L' in $\mathbf{BPP}$, drawing r at random. We can assume the probability of error to be at most $1 - \sqrt{2/3}$. Then we decide whether $(x, r) \in L'$ by testing whether (x, r, r') is in some fixed polynomial-time language, with error at most $1 - \sqrt{2/3}$. The probability of not making an error in drawing both r and r' is then at least $2/3$, and we conclude.

Another solution consists in observing that, if $\mathbf{NP} \subseteq \mathbf{BPP}$, then $\Sigma_2^p = \Pi_2^p = \mathbf{PH}$ (Corollary 1.23 of the notes). So $\mathbf{PH} = \Sigma_2^p = \mathbf{NP}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{BPP}} = \mathbf{MA} \subseteq \mathbf{AM}$ (Babai's Theorem) $= \mathbf{BP} \cdot \mathbf{NP} \subseteq \mathbf{BP} \cdot \mathbf{BPP}$, and as above, $\mathbf{BP} \cdot \mathbf{BPP} = \mathbf{BPP}$.

3 The Class $\oplus$P (Parity-P)

Let us recall the $\mathbf{BP}\cdot$ operator from the lectures: for any complexity class $\mathcal{C}$, $\mathbf{BP}\cdot\mathcal{C}$ is the class of languages L such that there is a randomized polynomial time Turing machine $\mathcal{A}'$ and a language $D' \in \mathcal{C}$ such that, on input x (of size n):

- If $x \in L$, then $Pr_r[\mathcal{A}'(x,r) \in D'] \geq 2/3$;
- If $x \notin L$, then $Pr_r[\mathcal{A}'(x,r) \in D'] \leq 1/3$.

where probabilities are taken on random strings r of size $q(n)$, for some polynomial q in n.

We define another operator $\oplus\cdot$ ("parity") as follows: $L \in \oplus\cdot\mathcal{C}$ iff there is a language L' in $\mathcal{C}$, and a polynomial $p(n)$, such that:

- $x \in L$ iff the number of strings y of size $p(n)$ such that $(x,y) \in L'$ is *odd*.

I.e., $\oplus\cdot\mathbf{P}$ is the class of language decidable on a non-deterministic Turing machine by letting all the branches of computation have the same length $p(n)$ (this is important), and then accepting iff the number of accepting branches is odd.

1. *(11 lines, 7.6%)* Define the relativization $(\oplus\cdot\mathbf{P})^A$ as $\oplus\cdot(\mathbf{P}^A)$. Show that $(\oplus\cdot\mathbf{P})^{\oplus\cdot\mathbf{P}} = \oplus\cdot\mathbf{P}$. (Hint: count modulo 2.) This is the *Papadimitriou-Zachos Lemma*.

   One has $\mathcal{C} \subseteq \oplus\cdot\mathbf{P}^{\mathcal{C}}$: run a deterministic polynomial time Turing machine with oracle $\mathcal{C}$; either it has one accepting branch, or zero. But one is odd, and zero is even.

   The converse inclusion is harder. One can observe that $\oplus\cdot\oplus\cdot\mathbf{P} \subseteq \oplus\cdot\mathbf{P}$ (hence the two classes are equal). If L is a language in $\oplus\cdot\oplus\cdot\mathbf{P}$, then there is a language $L' \in \oplus\cdot\mathbf{P}$ such that $x \in L$ iff the number of strings y (of the right size) such that $(x,y) \in L'$ is odd. Write $\#\{y \mid (x,y) \in L'\}$ this number. There is a further, polynomial time language L'' such that $(x,y) \in L'$ iff the number of strings (of the right size) z such that iff $((x,y),z) \in L''$ is odd. Now $\#\{(y,z) \mid ((x,y),z) \in L''\} = \sum_y \#\{z \mid ((x,y),z) \in L''\}$, and modulo 2, this is also $\sum_{y/\#\{z\mid((x,y),z)\in L''\}=1 \mod 2} 1$, since the ys for which $\#\{z \mid ((x,y),z) \in L''\} = 0 \mod 2$ cancel out. In turn, this is $\#\{y \mid \#\{z \mid ((x,y),z) \in L''\} = 1 \mod 2\} = \#\{y \mid y \in L'\}$. So $\#\{(y,z) \mid ((x,y),z) \in L''\}$ is odd iff $x \in L$. Hence $L \in \oplus\cdot\mathbf{P}$.

   However, what we really need to prove was $\oplus\cdot\mathbf{P}^{\oplus\cdot\mathbf{P}} \subseteq \oplus\cdot\mathbf{P}$. (I had forgotten this in a previous version of the solution, whence the low estimation on the required number of lines.) Using the above, it is enough to show $\mathbf{P}^{\oplus\cdot\mathbf{P}} \subseteq \oplus\cdot\mathbf{P}$, since then $\oplus\cdot\mathbf{P}^{\oplus\cdot\mathbf{P}} \subseteq \oplus\cdot\oplus\cdot\mathbf{P} \subseteq \oplus\cdot\mathbf{P}$.

   Let L be a language in $\mathbf{P}^{\oplus\cdot\mathbf{P}}$. So there is a language $L' \in \oplus\cdot\mathbf{P}$ such that L is in $\mathbf{P}^{L'}$. Let $\mathcal{M}$ be a polynomial-time machine with oracle deciding L' that decides L. Call a partial trace *$T = T_0T_1\ldots T_m$ of $\mathcal{M}$ any sequence of polynomially many configurations of $\mathcal{M}$, starting from the initial configuration $C_0(x)$, ending in an*

accepting configuration, and such that whenever T_k is not in the query state Q, then T_{k+1} is obtained by running $\mathcal{M}$ for one step from T_k. Also, we require that if T_k is in the query state, then T_{k+1} is in either the YES or the NO state, and nothing else changes compared with T_k. In other words, a partial trace is an accepting trace of $\mathcal{M}$ on input x, except that we don't check whether the oracle answered correctly.

One easily checks that a given word is a partial trace, in polynomial time (even logarithmic space). Given a partial trace T, we check that the oracle answers are correct as follows. Assume T made k calls to the oracle, with queries $z_1, \ldots, z_k$, and that the guessed answers were $b_1, \ldots, b_k$ (from the set $\{YES, NO\}$). Encode YES and NO as tape letters, say, equate YES with 1, NO with 0. Note that the correct answer to z_1 is YES iff $\#\{y \mid (z_1, y) \in L'\}$ is odd, where y is of the correct length.

Let L'_0 be a language that has exactly one more satisfying value y compared to L'. For example, let L'_0 be the language of all pairs (z, ay) (where a is a letter) such that either $a = 0$ and $(z, y) \in L'$, or $a = 1$ and y is a string of the form 1^ℓ for some length ℓ. Let also L'_1 be the language of all pairs (z, ay) such that $a = 0$ and $(z, y) \in L'$. So the correct answer to z_1 is YES iff $\#\{ay \mid (z_1, ay) \in L'_0\}$ is even, iff $\#\{ay \mid (z_1, ay) \in L'_1\}$ is odd. Let L'' be the language of those pairs (z, b) (where $b \in \{0, 1\}$) such that $\#\{ay \mid (z, ay) \in L'_b\}$ is odd. This is the language of pairs (z, b) such that either $b = 0 = NO$ and the correct answer to z is NO, or $b = 1 = YES$ and the correct answer to z is YES. I.e., L'' is the language of pairs (z, b) such that b is the correct answer to z.

Let now L''' be the language of all tuples $(x, (T, a_1y_1, \ldots, a_ky_k))$ where: (1) T is partial trace with k queries, as above; (2) for each query z_i and guessed answer b_i, a_i is a letter and y_i is a string of the right length; (3) such that $(z_i, a_iy_i) \in L'_{b_i}$ for each i. Since the product of k numbers is odd iff all are odd, $x \in L$ iff there is an odd number of strings $(T, a_1y_1, \ldots, a_ky_k)$ satisfying the latter conditions. L''' is clearly polynomial-time decidable, so $L \in \oplus \cdot \mathbf{P}$.

An alternative, and seemingly easier argument, consists in inlining the calls to the $\oplus \cdot \mathbf{P}$ oracle in a $\oplus \cdot \mathbf{P}^{\oplus \cdot \mathbf{P}}$ machine $\mathcal{M}$. I.e., when we enter the query state with query y, we execute the machine deciding membership in $L' \in \oplus \cdot \mathbf{P}$ (i.e., whether $\#\{z \mid (y, z) \in L''\}$ is odd, for some polynomial-time language L'') by guessing z and deciding whether $(y, z) \in L''$. Say $\mathcal{M}''$ decides L'' in polynomial-time. When we reach a final state of $\mathcal{M}''$, we proceed with the computation of $\mathcal{M}$. But, first, we need to proceed whether $\mathcal{M}''$ accepted or not. Second, we need to remember whether $\mathcal{M}''$ accepted: one may think that the modified $\mathcal{M}$ machine should accept in the end if and only if all inlined calls to $\mathcal{M}''$ accepted on the given branch. However, this is wrong: there are some branches in $\mathcal{M}$ that need to proceed provided the call to the L' oracle rejected. *So we have to duplicate the final states of $\mathcal{M}''$, depending on whether we expect $\mathcal{M}$ to get a YES or NO answer. Then we need to fiddle around to ensure that the number of accepting branches of*

the inlined machine $\mathcal{M}''$ is odd if and only if the expected answer is the correct one. This can be done as above (where we added one spurious accepting branch), or by modifying $\mathcal{M}''$ so that its total *number of branches is odd. The latter implies that either the number of accepting or the number of rejecting branches is odd, but not both.*

2. *(21 lines, 14.5%)* Let $\Sigma = \mathbb{Z}/2\mathbb{Z}$ in this Section. Recall that a linear hash function $h : \Sigma^m \to \Sigma^{m'}$ is a linear map from $\mathbb{Z}/2\mathbb{Z}^m$ to $\mathbb{Z}/2\mathbb{Z}^{m'}$.

   Let F be a propositional formula, built on propositional variables $x_1, \ldots, x_m$, say. Let X be the set of environments (mappings from the propositional variables $x_1, \ldots, x_m$ to truth-values) ρ that satisfy F (in notation, $\rho \models F$). Let m' be a natural number such that $2^{m'-2} \leq |X| \leq 2^{m'-1}$, where $|X|$ is the cardinality of X. Identify each environment ρ with the obvious vector in Σ^m. Show that:

$$Pr_{h,b}[\exists! \rho \in \Sigma^m \cdot \rho \models F \text{ and } h(\rho) = b] \geq \frac{1}{8}$$

   where h is drawn at random uniformly among all linear hash functions from Σ^m to $\Sigma^{m'}$, and b is drawn at random uniformly, and independently, in $\Sigma^{m'}$. We write $\exists!$ for "there exists a unique". (Hint: given a fixed ρ, find a lower bound for the probability of the event $C_\rho(h, b)$, defined as holding whenever $h(\rho) = b$ but $h(\rho') \neq b$ for every $\rho' \in X$ such that $\rho' \neq \rho$.)

   Given a fixed $\rho \in X$:

$$\begin{aligned} Pr_{h,b}[C_\rho(h,b)] &= Pr_{h,b}[h(\rho) = b \text{ and } \rho \text{ is not a collision for } h \text{ in } X] \\ &= Pr_{h,b}[h(\rho) = b \mid \rho \text{ is not a collision for } h \text{ in } X] \\ &\quad .Pr_{h,b}[\rho \text{ is not a collision for } h \text{ in } X] \end{aligned}$$

   by Bayes' law for example. The probability that $h(\rho) = b$ knowing that ρ is not a collision for h in X (assuming such h and b exist) is exactly the mean of all probabilities over b that $b = b_0$, for various values of b_0, i.e., $1/2^{m'}$.
   We now use the technique we used to show Sipser's first coding lemma (Lemma 3.13 in the notes) with $\ell = 1$. The probability over h and b (hence in fact over h alone) that ρ is a collision for h in X is:

$$\begin{aligned} Pr_h[\exists \rho' \in X \cdot \rho' \neq \rho \text{ and } h(\rho') = h(\rho)] &\leq \sum_{\rho' \neq \rho} \frac{1}{2^{m'}} \\ &\leq \frac{2^{m'-1} - 1}{2^{m'}} \leq \frac{1}{2} \end{aligned}$$

   since $|X| \leq 2^{m'-1}$. So the probability that ρ is not a collision for h in X is at least $1/2$.

We therefore obtain:

$$Pr_{h,b}[C_\rho(h,b)] \;\geq\; \frac{1}{2^{m'+1}}$$

Knowing this,

$$\begin{aligned} Pr_{h,b}[\exists!\rho \in \Sigma^m \cdot \rho \models F \text{ et } h(\rho) = b] &= Pr_{h,b}[\exists \rho \in X \cdot C_\rho(h,b)] \\ &= \sum_{\rho \in X} Pr_{h,b}[C_\rho(h,b)] \end{aligned}$$

since the events $C_\rho(h,b)$ are disjoint as ρ varies. Since $2^{m'-2} \leq |X|$ and $Pr_{h,b}[C_\rho(h,b)] \geq \frac{1}{2^{m'+1}}$, this is greater than or equal to $2^{m'-2}.\frac{1}{2^{m'+1}} = \frac{1}{8}$.

3. *(17 lines, 11.7%)* Deduce a probabilistic polynomial time algorithm that takes as input a propositional formula F, and outputs propositional formulae $F_1, \ldots, F_k$ (for some polynomial k) such that:

   - If F is unsatisfiable, then so are $F_1, \ldots, F_k$;
   - If F is satisfiable, then with high probability, some F_i has a *unique* satisfying assignment (formally, $Pr[F_1$ is uniquely satisfiable or $\ldots$ or F_k is uniquely satisfiable$] \geq 2/3$).

   Given a hash function h and a vector b in $\Sigma^{m'}$, one can build a formula $F(h,b)$ whose satisfying assignments ρ are those that satisfy F and such that $h(\rho) = b$. Representing h as a 0-1 matrix, this is a conjunction of parity tests of the form $x_{i_1} + \ldots + x_{i_p} = 1$ (or $= 0$), where $+$ is exclusive-or.

   If one finds the right m' (since $m' \geq 2$, $|X| \geq 1$, which shows that F must be satisfiable for this m' to exist), the previous question shows that we obtain a formula $F(h,b)$ that has a unique satisfying assignment, with probability at least $1/8$, this way.

   We do not know m'. But we enumerate all its possible values, between 2 and $m+1$. For each value of m', we build a formula $F(h_{m'}, b_{m'})$ by drawing $b_{m'} \in \Sigma^{m'}$ and h linear from Σ^m to $\Sigma^{m'}$. With probability at least $1/8$, at least one of these m formulae will be uniquely satisfiable (assuming F satisfiable).

   Let us do the construction 6 times: we obtain $6m$ formulae, in polynomial time. When F is satisfiable, the probability that at least one of the formulae is uniquely satisfiable is at least $6/8$, which is greater than or equal to $2/3$.

   On the other hand, if F is unsatisfiable, clearly none of the formulae thus constructed is satisfiable.

4. *(12 lines, 8.3%)* Conclude that $\mathbf{NP} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}}$.

SAT is **NP***-complete. Do the above construction. Given an input clause set (or more generally, propositional formula) F, either F is satisfiable, and with high probability one of the obtained formulae F_1, ..., F_k at least has a unique satisfying assignment (... and* 1 *is odd).*

I.e., we use a randomized machine to compute F_1, ..., F_k. For each, we call an oracle deciding $\oplus$SAT (i.e., whether a given formula has an odd number of satisfying assignments: this is trivially in $\oplus \cdot \mathbf{P}$). If at least one call to the oracle returns YES, we accept, otherwise we reject.

If F is satisfiable, we will then accept with probability at least $2/3$, otherwise we will definitely reject.

Since $\mathbf{BPP}^{\oplus \cdot P}$ is stable under polynomial time reductions, we conclude. We have in fact shown that $\mathbf{NP} \subseteq \mathbf{RP}^{\oplus \cdot \mathbf{P}}$.

4 Relativizing Fagin's Theorem, and SAT

All of our results in the lectures used multi-tape Turing machines. It will be easier to reason with single-tape Turing machines here. Recall that we can encode all multi-tape Turing machines as single-tape Turing machines, encoding all tapes B_1, ..., B_m (each with some symbol representing the control state at the position of the head) as the concatenation $B_1 \# \ldots \# B_m$ where $\#$ is some fresh separator symbol. Let Q be the alphabet of control states. We shall assume the query tape is encoded as B_1, and that the head is at the beginning of B_1 when the machine enters state Q.

1. *(19 lines,* 13.1*%)* Remember that (Fagin's Theorem) the languages in **NP** are exactly the languages definable by existential second-order formulae. See the Appendix for the precise statement, and for a proof.

   We shall need the following second-order construction. Given a formula F, and first-order variables (occurring or not) x_1, ..., x_k in F, a *predicate expression* (not to be confused with a predicate variable P) is an expression of the form $\lambda x_1, \ldots, x_k \cdot F$. The latter is of *arity* k. The notation may be clearer if you write $F(x_1, \ldots, x_k)$ instead of F: $\lambda x_1, \ldots, x_k \cdot F(x_1, \ldots, x_k)$ denotes the predicate of k values that is true exactly when the formula F is true of these values.

   We shall consider a *third-order* predicate symbol $A^?$. A *relativized* second-order formula F_2 is a second-order formula with an additional formula former construct $A^?(E_1, \ldots, E_m)$ where E_1, ..., E_m are predicate expressions: i.e., the third-order predicate symbols are applied to (second-order) predicate expressions.

   The semantics is now defined by giving oneself the denotation of $A^?$ as a fixed m-ary relation among relations (denoting E_1, ..., E_m) of the relevant arities. (Formally, $A^?$ comes with a fixed *signature* $(k_1, \ldots, k_m)$, and we require each E_i to be of arity k_i.)

   This is probably hard to understand. Let us illustrate this in the case that will interest us. This uses notions from the proof of Fagin's Theorem we have already seen, see the

Appendix. If A is a language, and the tape alphabet is $a_1, \ldots, a_p$, so that $T_{a_i}(\vec{t}, \vec{x})$ encoded the positions $\vec{x}$ where one could find letter a_i at time t, the predicate expressions $\lambda\vec{x} \cdot T_{a_i}(\vec{t}, \vec{x})$, $1 \leq i \leq p$, altogether encode the contents of the tape at time $\vec{t}$. So one can use $A^?(\lambda\vec{x} \cdot T_{a_1}(\vec{t}, \vec{x}), \ldots, \lambda\vec{x} \cdot T_{a_p}(\vec{t}, \vec{x}))$ as a formula testing whether the contents of the tape is in language $QA\#\Sigma^*$ (i.e., control state, then a word in A, separator, and all the other tapes).

Show that the languages in $\mathbf{NP}^A$ are exactly those definable by relativized existential second-order formulae.

First, to evaluate a given relativized second-order formula F_2 on a model with n elements, we guess the interpretations of each existentially quantified predicate variable, as in the non-relativized case, in polynomial time. We then evaluate the formula in this model, in polynomial time again. The only new case is that of formulae of the form $A^?(E_1, \ldots, E_m)$. Given any $E_i = \lambda x_1, \ldots, x_{k_i} \cdot F_i$, we evaluate F_i on each k_i-tuple of values: the result is the truth-table of E_i, and can be built in polynomial time. Of course, not every m-tuple of tables encodes a language (e.g., we need that no two letters are at the same position, and each position has at least one letter), but we can check these conditions in polynomial time, build the corresponding tape, and decide by making a call to the oracle deciding membership in A. (Before this, we prepend some state in Q, and append a # sign.)

Conversely, we adapt the proof in the Appendix. We change COMPUTE into COMPUTE′, which is the conjunction of $\neg S_Q(\vec{t}) \Rightarrow$ COMPUTE (if not in state Q, compute as before) and $S_Q \Rightarrow$ ORACLE, where ORACLE is the conjunction of:

- $T_a(\vec{t}, \vec{x}) \wedge \vec{t'} = \vec{t} + 1 \Rightarrow T_a(\vec{t'}, \vec{x})$ *for all* $a \in \Sigma$ *(the tape does not change);*
- $H(\vec{t}, \vec{x}) \wedge \vec{t'} = \vec{t} + 1 \Rightarrow H(\vec{t'}, \vec{x})$ *(the head does not move);*
- $A^?(\lambda\vec{x} \cdot T_{a_1}(\vec{t}, \vec{x}), \ldots, \lambda\vec{x} \cdot T_{a_p}(\vec{t}, \vec{x})) \wedge \vec{t'} = \vec{t} + 1 \Rightarrow S_{YES}(\vec{t'})$;
- $\neg A^?(\lambda\vec{x} \cdot T_{a_1}(\vec{t}, \vec{x}), \ldots, \lambda\vec{x} \cdot T_{a_p}(\vec{t}, \vec{x})) \wedge \vec{t'} = \vec{t} + 1 \Rightarrow S_{NO}(\vec{t'})$.

2. *(29 lines, 20%)* From this deduce that FORM-SAT^A is $\mathbf{NP}^A$-complete under polynomial-time reductions, where FORM-SAT^A is defined as follows. This is a variant on FORM-SAT, the problem of deciding whether an input propositional formula is satisfiable. (SAT further restricts the formula to be in clausal form, but FORM-SAT will be easier for our purposes.) FORM-SAT^A is as FORM-SAT, except we now also accept atomic formulae of the form $A^!(x_1, \ldots, x_q)$ where $x_1, \ldots, x_q$ are propositional variables. We say that $A^!(x_1, \ldots, x_q)$ is true iff the bit-string $x_1, \ldots, x_q$ is an encoding (in a sense that you shall make precise) of a tape contents in the language $QA\#\Sigma^*$.

Given any language L in $\mathbf{NP}^A$, find a relativized formula F_2 as in the previous question defining L. We define the following reduction to SAT^A. On input x, of size n, for each existentially quantified predicate variable P, of arity k, create k^n propositional formulae (a polynomial number, since k is fixed). We write

them simply $P(v_1, \ldots, v_n)$, where v_1, …, v_n are values in the n-element model. Translate $F_2 = \exists P_1, \ldots, P_m \cdot \forall \vec{t}, \vec{t'}, \vec{x}, \ldots \cdot G$, where G is quantifier-free, as the conjunction of all the instances of G obtained by replacing the first-order variables $\vec{t}$, $\vec{t'}$, $\vec{x}$, …, by all possible tuples of values. Since there are a fixed number of variables, this takes polynomial time.

Also, replace $A^?(E_1, \ldots, E_m)$, where each E_i is of arity k_i, by $A^!(\vec{x}_1, \ldots, \vec{x}_m) \wedge \bigwedge_{i=1}^m def_{E_i}(\vec{x}_i)$, where:

- *for each i, $\vec{x}_i$ is a k_i^n-tuple of fresh variables $x_{v_1,\ldots,v_{k_i}}$, meant to denote the k_i^n-tuple of truth-values taken by E_i on each of the k_i^n possible k_i-tuples $(v_1, \ldots, v_{k_i})$ of arguments to E_i;*
- *$def_E(\vec{x})$, where $E = \lambda y_1, \ldots, y_k \cdot F(y_1, \ldots, y_k)$, states the semantics of E: this is the conjunction over all k-tuples of values v_1, …, v_k of $x_{v_1,\ldots,v_k} \Leftrightarrow F(v_1, \ldots, v_k)$.*

Finally, take the conjunction of all the above with the encoding of the input: the conjunction of all clauses $X_a(\vec{x})$ (on all positions $\vec{x}$ where there is a letter a), $\neg X_a(\vec{x})$ (on all other positions), $a \in \Sigma$.

This all takes polynomial time. So FORM-SAT^A is $\mathbf{NP}^A$-hard. Since we apply this to the case $E_i = \lambda \vec{x} \cdot T_{a_i}(\vec{t}, \vec{x})$ (with T_{a_i} of arity $2k$), the desired encoding x_1, …, x_q of the tape is an mk^n-tuple of fresh propositional variables which denote, for each letter and each position on the tape, whether this letter is at this position on the tape.

Conversely, we decide an instance of FORM-SAT^A by guessing the values of all propositional variables in polynomial time. We evaluate the resulting formula by calling A as an oracle on the tape that its arguments encode if they do, and returning an arbitrary value otherwise.

3. *(7 lines, 4.8%)* Show that the result $\mathbf{NP} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}}$ from Part 3 relativizes (see the precise statement below). To this end, we define $\mathbf{BPP}^{\oplus \cdot \mathbf{P}}$ relativized to oracle A so that both the **BPP** and the $\oplus \cdot \mathbf{P}$ machines have access to the oracle. Since the **BPP** machine can already use a call to a $(\oplus \cdot \mathbf{P})^A$ oracle to obtain answers to queries to A, we define $(\mathbf{BPP}^{\oplus \cdot \mathbf{P}})^A$ as $\mathbf{BPP}^{(\oplus \cdot \mathbf{P})^A} = \mathbf{BPP}^{\oplus \cdot \mathbf{P}^A}$.

   Then show that $\mathbf{NP}^A \subseteq (\mathbf{BPP}^{\oplus \cdot \mathbf{P}})^A$.

   Use FORM-SAT^A instead of SAT. The construction of F_1, …, F_k from F is the same, since each is obtained by taking F, unchanged, in conjunction with a formula encoding $h(\rho) = b$. The machine then needs to call an oracle to test the parity of the number of assignments that satisfy F_i, for each i. The is easily done with an $\oplus \cdot \mathbf{P}^A$ oracle, since checking whether an assignement satisfies a relativized formula is in $\mathbf{P}^A$.

   We conclude again since $\mathbf{BPP}^{\oplus \cdot \mathbf{P}^A}$ is stable under polynomial time reductions.

4. *(3 lines, 2%)* Show that, if $\mathbf{NP}^A \subseteq \mathbf{BPP}^A$, then $\mathbf{PH}^A \subseteq \mathbf{BPP}^A$. We define the relativized polynomial hierarchy $\mathbf{PH}^A$ as the union of Σ_i^A, $i \geq 1$, where $\Sigma_1^A = \mathbf{NP}^A$, $\Sigma_{i+1}^A = \mathbf{NP}^{\Sigma_i^A}$. (I am not expecting a full, detailed proof, but at least a good roadmap.)

   We prove this as the Zachos Lemma, observing that everything relativizes. To this end, we must also check that the $\mathbf{MA} \subseteq \mathbf{AM}$ *theorem relativizes. This is tedious, but clear.*

5. *(4 lines,* 2.8*%)* Deduce the first of Toda's theorems: $\mathbf{PH} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}}$.

   Since $\mathbf{NP}^A \subseteq (\mathbf{BPP}^{\oplus \cdot \mathbf{P}})^A$*, taking* $A = \oplus \cdot \mathbf{P}$*,* $\mathbf{NP}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}^{\oplus \cdot \mathbf{P}}}$*. By the Papadimitriou-Zachos Lemma, this is included in* $\mathbf{BPP}^{\oplus \cdot \mathbf{P}}$*. We can now apply the relativized Zachos Lemma and conclude that* $\mathbf{PH}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}}$*.*
   By induction on i *in* Σ_i^P *and* Σ_i^A*,* $\Sigma_i^P \subseteq \Sigma_i^A$*. So* $\mathbf{PH} \subseteq \mathbf{PH}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{BPP}^{\oplus \cdot \mathbf{P}}$*.*

5 Toda's Theorem

The second of Toda's Theorems is $\mathbf{PP}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{P}^{\oplus \cdot \mathbf{P}}$. This would fill in itself another advanced complexity test. Since $\mathbf{BPP}^A \subseteq \mathbf{PP}^A$ (which is easy), and $\mathbf{P}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{PSPACE}$ (easy as well) we conclude:

Toda's Theorem: $\mathbf{PH} \subseteq \mathbf{P}^{\oplus \cdot \mathbf{P}} \subseteq \mathbf{PSPACE}$

In particular, the power of counting modulo 2 is very high: somewhere between **PH** and **PSPACE**.

(This part does not contain any question.)

A Fagin's Theorem

We have seen the following version of Fagin's Theorem in the homework assignment:

> The languages in **NP** are exactly those languages L such that there is an existential second-order formula F_2, with no free first-order variable, and only $=$, P and X_a, $a \in \Sigma$, as free predicate variables, such that for every input x of size n, x is in L iff $I_n[(X_a \mapsto D_a)_{a\in\Sigma}, P \mapsto D_<] \models F_2$. Here D_a is the set of positions in x where one finds the letter a (i.e., X_a, $a \in \Sigma$, encode the input x), and $I_n[(X_a \mapsto D_a)_{a\in\Sigma}, P \mapsto D_<]$ is the unique equational structure in standard representation with domain of cardinality n that maps X_a to D_a for each $a \in \Sigma$ and P to the strict ordering $1 < 2 < \ldots < n$.

In one direction, we show that evaluating any existential second-order formula over a model can be done in non-deterministic polynomial time in the size of the model (the formula is fixed).

Conversely, if $\mathcal{M}$ is a non-deterministic polynomial time machine deciding L, we build F_2 as follows. Assume that $\mathcal{M}$ takes time and space bounded by n^k where n is the size of the input. We create the following predicate variables:

- S_q, for each control state q; $S_q(\vec{t})$ should hold exactly when $\mathcal{M}$ is at control state q at time (denoted by) $\vec{t}$;
- T_a, for each tape letter $a \in \Sigma$; $T_a(\vec{t}, \vec{x})$ should hold exactly when the symbol at position (denoted by) $\vec{x}$ at time (denoted by) $\vec{t}$ is a;
- H: $H(\vec{t}, \vec{x})$ should hold exactly when the head is at position $\vec{x}$ at time $\vec{t}$.

F_2 is the existential quantification over all predicate variables S_q, T_a, and H of the first-order formula φ defined as the universal closure (i.e., add $\forall x$ for every free first-order variable in front) of $Z(\vec{0}) \wedge L(\ell) \Rightarrow$ CONSISTENCY $\wedge$ START $\wedge$ COMPUTE $\wedge$ END, where the latter are defined below.

$\vec{0}$ is a k-tuple of first-order variables, and $Z(\vec{0})$ is the first-order formula $\forall \vec{x} \cdot \neg P(\vec{x}, \vec{0})$, stating that $\vec{0}$ encodes time 0. $L(\ell)$ is the first-order formula $\forall x \cdot \neg P(\ell, x)$ stating that variable ℓ should be mapped to n.

- CONSISTENCY states that at each time, the head is always in at most one position, the machine is in at most one control state, and the contents of the tape is unique. This is the conjunction of:
  - $\neg(H(\vec{t}, \vec{x}) \wedge H(\vec{t}, \vec{x}') \wedge \vec{x} \neq \vec{x}')$ (which we prefer to the more natural $H(\vec{t}, \vec{x}) \wedge H(\vec{t}, \vec{x}') \Rightarrow \vec{x} = \vec{x}'$, as the transition to II.6 will be more natural this way), where $\vec{x} \neq \vec{x}'$ is the disjunction of all $x_i \neq x_i'$, $1 \leq i \leq k$;
  - for each pair of distinct control states q, q', $\neg(S_q(\vec{t}) \wedge S_{q'}(\vec{t}))$;
  - for each pair of distinct letters a, b, $\neg(T_a(\vec{t}, \vec{x}) \wedge T_b(\vec{t}, \vec{x}))$.

- START states what the tape contains initially. This is the conjunction of:
  - $S_{q_0}(\vec{0})$ (initially, $\mathcal{M}$ is in the initial state q_0),
  - $H(\vec{0},\vec{0})$ (initially, the head is at position 0),
  - writing $\vec{0}[x]$ for the vector of variables $\vec{0}$, except the last one is replaced by variable x, the conjunction over all letters $a \in \Sigma$ of $X_a(\vec{0}[x]) \Rightarrow T_a(\vec{0}[x])$ (the first n letters on the tape at time 0 are given by the input);
  - $\vec{0}[\ell] <^k \vec{x} \Rightarrow T_{\sqcup}(\vec{x})$, where $\sqcup$ is the blank (all other letters on the tape are blank).
- COMPUTE is more complex, but states the expected transition relation. It is the conjunction of:
  - What part of the tape does not change: the conjunction over $a \in \Sigma$ of $T_a(\vec{t},\vec{x}) \wedge H(\vec{t},\vec{y}) \wedge \vec{x} \neq \vec{y} \wedge \vec{t'} = \vec{t}+1 \Rightarrow T_a(\vec{t'},\vec{x})$; the formula $\vec{t'} = \vec{t}+1$ was defined in II.3.
  - What changes: the conjunction over all control states q and all letters $a \in \Sigma$ of $\mathrm{PRE}(q,a) \Rightarrow \bigvee_{q',b,dir} \mathrm{POST}(q',b,dir)$, where (q',b,dir) ranges over the triples of all possibles next control state (q'), letter to be written (b) and direction (dir, being 0, +1 or −1).
    We let $\mathrm{PRE}(q,a) = S_q(\vec{t}) \wedge H(\vec{t},\vec{x}) \wedge T_a(\vec{t},\vec{x}) \wedge \vec{t'} = \vec{t}+1$.
    And we let:

$$\begin{aligned}
\mathrm{POST}(q',b,dir) &= S'_{q'}(\vec{t'}) \wedge T_b(\vec{t'},\vec{x}) \wedge H(\vec{t'},\vec{x}) \quad \text{if } dir = 0 \\
\mathrm{POST}(q',b,dir) &= S'_{q'}(\vec{t'}) \wedge T'_b(\vec{t'},\vec{x}) \wedge (\vec{x'} = \vec{x}+1 \Rightarrow H(\vec{t'},\vec{x'})) \quad \text{if } dir = +1 \\
\mathrm{POST}(q',b,dir) &= S'_{q'}(\vec{t'}) \wedge T'_b(\vec{t'},\vec{x}) \wedge (\vec{x} = \vec{x'}+1 \Rightarrow H(\vec{t'},\vec{x'})) \quad \text{if } dir = -1
\end{aligned}$$

- END is the conjunction over all non-final states q of $t_1 = \ell \wedge \ldots \wedge t_k = \ell \Rightarrow \neg S_q(\vec{t})$ (at the last time instant $n^k - 1$, the control state is not non-final). (We may also take the disjunction over all final states q of $t_1 = \ell \wedge \ldots \wedge t_k = \ell \Rightarrow S_q(\vec{t})$.)