COMPUTATIONAL COMPLEXITY HW-2

SIDDHANT CHAUDHARY, AMIK RAJ BEHERA
BMC201953 , BMC201908

Problem 1. In class, we defined the polynomial heirarchy using quantifiers. A language L is in Σ_2^p if there exists a polynomial time TM M and a polynomial q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} M(x, u_1, u_2) = 1$$

Show that $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$.

Solution. We know that SAT is an **NP**-complete problem, and hence $\mathbf{NP}^{\text{SAT}} = \mathbf{NP}^{\mathbf{NP}}$. So, it is enough to show that

$$\Sigma_2^p = \mathbf{NP}^{\text{SAT}}$$

and infact this is what we will show.

First, suppose $L \in \Sigma_2^p$. By definition, there is a polynomial time TM M and a polynomial q such that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} M(x, u_1, u_2) = 1$$

Consider the language L' defined as

$$\langle x, u_1 \rangle \in L' \iff u_1 \in \{0,1\}^{q|x|} \text{ and } \forall u_2 \in \{0,1\}^{q|x|} M(x, u_1, u_2) = 1$$

Now, look at the statement

$$\forall u_2 \in \{0,1\}^{q|x|} M(x, u_1, u_2) = 1$$

This is clearly a statement in **co-NP**. Since SAT is **NP**-complete, the truth of this statement can be determined by oracle access to SAT. This means that the membership in the language L' can be solved by oracle access to SAT. Also, observe that

$$x \in L \iff \exists u_1 \in \{0,1\}^{q|x|} \text{ such that } \langle x, u_1 \rangle \in L_1$$

Now, consider the following non-deterministic algorithm.

(1) On input x, non-deterministically generate a string u_1 such that $u_1 \in \{0,1\}^{q|x|}$.
(2) Using the oracle SAT, determine whether $\langle x, u_1 \rangle \in L'$ or not. Return the corresponding answer.

Clearly, this is a non-deterministic polynomial time algorithm that accepts the language L. So, it follows that $L \in \mathbf{NP}^{\text{SAT}}$, and hence $\Sigma_2^p \subseteq \mathbf{NP}^{\text{SAT}}$.

Conversely, suppose $L \in \mathbf{NP}^{\text{SAT}}$. Suppose we have a polynomial time NDTM M that has oracle access to SAT and which accepts the language L. Suppose x is the input to M. Being non-deterministic, M will make some choices in its computation, and suppose the choices it makes are encoded by bits $c_1, ..., c_m \in \{0,1\}$. Also during its computation, M will query the oracle several times, and the oracle returns a bit, i.e either 0 or 1 for each query. Suppose in the computation for x, the machine gets answers $a_1, ..., a_k$, where each $a_k \in \{0,1\}$. Suppose for

Date: March 10 2021.

$1 \leq i \leq k$, the boolean formula φ_i represents the i^{th} query that M asks the oracle. Then, observe that $x \in L$ if and only if the following conditions are true:

(1) M reaches the accepting state q_{accept} on input x using the choices $c_1, ..., c_m$.
(2) Each a_i is a correct answer to *some* query. This can be made formal as follows:

$$a_i = 1 \iff \exists \text{ assignment } u_i \text{ for } \varphi_i \text{ such that } \varphi_i(u_i) = 1$$
$$a_i = 0 \iff \forall \text{ assignments } v_i \text{ of } \varphi_i, \varphi_i(v_i) = 0$$

All of this can be written as follows:

$x \in L \iff \exists c_1, ..., c_m, a_1, ..., a_k, u_1, ..., u_k \forall v_1, ..., v_k$
M accepts x on making choices $c_1, ..., c_m$ and getting answers $a_1, ..., a_k$ and for each $1 \leq i \leq k$ if $a_i = 1$ then $\varphi_i(u_i) = 1$ and if $a_i = 0$ then $\varphi_i(v_i) = 0$

Clearly, this is a statement representing a language in Σ_2^p, i.e $L \in \Sigma_2^p$ and hence $\mathbf{NP}^{\text{SAT}} \subseteq \Sigma_2^p$. This shows that $\mathbf{NP}^{\text{SAT}} = \Sigma_2^p$, and completes the proof of the claim. ■

Problem 2. Show that **SPACE**$(n) \neq$ **NP**.

Solution. We will prove it by contradiction. Assume that **SPACE**(n) = **NP**. From Space Hierarchy Theorem, we know that **SPACE**$(n) \subsetneq$ **SPACE**(n^2). Thus there exists a language $A \in$ **SPACE**(n^2) and $\notin$ **SPACE**(n). Let M be the Turing machine which decides A in $O(n^2)$ space. Define A-PAD as follows:

$$\text{A-PAD} = \{x\#1^{|x|^2} \mid x \in A\}.$$

We will now show that A-PAD $\in$ **SPACE**(n) by the following algorithm:

Algorithm 1 CHECK:A-PAD(w)

```
Find the first occurrence of # and mark it as P
Denote the string upto P as x
Run M(x) {Runs in O(|x|^2) space}
if M(x) == True then
  if String after P has exactly |x|^2 1's == True then
    Accept w {Runs in O(|x|^2) space}
  else
    Reject w
  end if
else
  Reject w
end if
```

Thus we have shown that A-PAD $\in$ **SPACE**(n), and by our assumption, this implies that A-PAD $\in$ **NP**. It is easy to see that we can reduce A to A-PAD in polynomial time. Thus $A \in$ **NP**. Again by our assumption, $A \in$ **SPACE**(n), which is a contradiction because we specifically chose A to not lie in **SPACE**(n). Hence our assumption that **SPACE**(n) = **NP** is false.

■

Problem 3. Show that if a sparse language is **NP**-complete, then **P** = **NP**.

Solution. We first define LSAT.

$\langle \phi, x \rangle \in$ LSAT if ϕ is a Boolean formula in n variables, $x \in \{0,1\}^n$ and there is a satisfying assignment of ϕ which is lexicographically at most x.

We can reduce SAT to LSAT as follows: SAT(ϕ) is True if and only LSAT$(\langle \phi, 1^n \rangle)$ is True, where ϕ is a Boolean formula on n variables. Since we know that SAT is **NP**-complete, we conclude that LSAT is **NP**-complete. Let A be a sparse language which is **NP**-complete. The by definition of **NP**-completeness, we get a polynomial time reduction f from LSAT to A. It means that $\langle \phi, x \rangle \in$ LSAT if and only if $f(\langle \phi, x \rangle) \in A$.

Since f is a polynomial time algorithm, the output of f is also polynomially bounded i.e. there exists a polynomial p such that on input x, $|f(x)| \leq p(x)$. By definition of sparse language, there exists a polynomial q such that $|A \cap \{0,1\}^n| \leq q(n)$. Given a Boolean formula ϕ with n variables, let $M = q(p(|\phi| + n))$. M upper bounds the number of elements in A with length $p(|\phi| + n)$, which implies that M upper bounds the number of elements in A such that $f(\langle \phi, x \rangle) \in A$. Note that M is poly$(|\phi| + n)$.

Now we will describe a polynomial time algorithm for solving SAT.

(1) Let ϕ be the input to SAT and let α be the lexicographically smallest satisfying assignment of ϕ. Our goal is to determine α.
(2) Choose $M + 1$ *evenly spaced* assignments $y_1, \ldots, y_{M+1}$, arranged in lexicographically ascending order.
(3) Compute $f(\langle \phi, y_1 \rangle), \ldots, f(\langle \phi, y_{M+1} \rangle)$.
(4) Now there are two cases based on the above computed outputs:
    (a) $f(\langle \phi, y_i \rangle) = f(\langle \phi, y_j \rangle)$, for some $1 \leq i < j \leq M + 1$. If $f(\langle \phi, y_i \rangle) \in A$, this means that $\alpha \leq y_i, y_j$. In other words, $\alpha \notin (y_{i+1}, \ldots, y_j)$. Else $f(\langle \phi, y_i \rangle) \notin A$, which means that $\alpha > y_i, y_j$. In this case also, $\alpha \notin (y_{i+1}, \ldots, y_j)$. In either of the case, we have removed approximately $1/M$ possibilities for α from the assignment space (i.e. all possible assignments for ϕ).
    (b) All $f(\langle \phi, y_i \rangle)$ are distinct. Since all $|f(\langle \phi, y_i \rangle)|$'s are bounded by length $p(|\phi| + n)$, it means that there is atleast one $f(\langle \phi, y_k \rangle) \notin A$. In other words, $\alpha > y_k \geq y_1 \Rightarrow \alpha \notin (0, \ldots, y_1)$. Again, we have removed approximately $1/M$ possibilities for α from the assignment space.
(5) Repeating from Step 2 again (excluding the *removed assignments* from Step 4.a and 4.b), for $O(nm)$ times.
(6) We will be left with poly(n) of assignments. Now we can check one by one whether α is contained in these polynomial number of possibilities.
(7) If we find α, then we return saying that ϕ has a satisfying assignment, otherwise ϕ has no satisfying assignment.

So we have described a polynomial time algorithm solving SAT. Since SAT is **NP**-complete, this concludes that **P** = **NP**. ■

Problem 4. Prove the following:

(1) **RP** and **BPP** are closed under union and intersection.

Solution. Let L_1 and L_2 be two languages in **RP** and let M_1 and M_2 be the probabilistic Turing machines respectively.

(a) **RP** is closed under union: Let $L = L_1 \cup L_2$. The following UNION-RP will decide L:

Algorithm 2 UNION-RP(x)

```
Flip a fair coin
if Heads appears then
  return M_1(x)
else
  return M_2(x)
end if
```

It is clear to see that UNION-RP is a probabilistic Turing Machine and runs in polynomial time.

If $x \in L$, then $Pr[\text{UNION-RP accepts } x] \geq \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}$.

If $x \notin L$, then $Pr[\text{UNION-RP accepts } x] \geq \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0 = 0$.

Thus $L \in$ **RP**.

(b) **RP** is closed under intersection: Let $L = L_1 \cap L_2$. The following INTERSECT-RP will decide L:

Algorithm 3 INTERSECT-RP(x)

```
Run M_1(x)
if M_1(x) == True then
  Run M_2(x)
  if M_2(x) == True then
    return Accept
  else
    return Reject
  end if
else
  return Reject
end if
```

It is clear to see that INTERSECT-RP is a probabilistic Turing Machine and runs in polynomial time.

If $x \in L$, then $Pr[\text{INTERSECT-RP accepts } x] \geq \frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9}$. Note that any constant greater than 0 will work for $Pr[\text{INTERSECT-RP accepts } x]$, since by error reduction (i.e. running several M_1 and M_2's simultaneously and taking the majority's result) the probability can be increased to atleast $\frac{1}{2}$.

If $x \notin L$, then $Pr[\text{INTERSECT-RP accepts } x] \geq \frac{2}{3} \cdot 0 + 0 = 0$. x is rejected when either $x \notin L_2$ or $x \notin L_1$.

Thus $L \in$ **RP**.

Let L_1 and L_2 be two languages in **BPP** and let M_1 and M_2 be the probabilistic Turing machines respectively.

(a) **BPP** is closed under union: Let $L = L_1 \cup L_2$. The following UNION-BPP will decide L:

Algorithm 4 UNION-BPP(x)

```
Flip a fair coin
if Heads appears then
  return M_1(x)
else
  return M_2(x)
end if
```

It is clear to see that UNION-BPP is a probabilistic Turing Machine and runs in polynomial time.

If $x \in L$, then $Pr[\text{UNION-BPP accepts } x] \geq \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}$.

If $x \notin L$, then $Pr[\text{UNION-BPP rejects } x] \geq \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}$.

Thus $L \in$ **BPP**.

(b) **BPP** is closed under intersection: Let $L = L_1 \cap L_2$. We showed above that **BPP** is closed under union and in Problem 4.b, we showed that **BPP** is closed under complement. Since

$$L = (L_1^c \cup L_2^c)^c,$$

it follows that **BPP** is closed under intersection.

■

(2) **BPP** is closed under complement. Is **RP** closed under complement?

Solution. Let $L \in$ **BPP** and let M be the probabilistic Turing machine which decides L. Then COMPLEMENT-BPP decides L^c:

Algorithm 5 COMPLEMENT-BPP(x)

```
Run M(x)
return Negation of M(x)
```

It is clear to see that COMPLEMENT-BPP is a probabilistic Turing Machine and runs in polynomial time.

If $x \in L^c \Rightarrow x \notin L$, then $Pr[\text{COMPLEMENT-BPP accepts } x] \geq \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}$.

If $x \notin L^c \Rightarrow x \in L$, then $Pr[\text{COMPLEMENT-BPP rejects } x] \geq \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}$.

Thus $L^c \in$ **BPP**.

Complement of **RP** is **coRP**, and it strongly believed that **RP** $\neq$ **coRP**. Thus **RP** is believed not to be closed under complement. ■

(3) There is a decidable language that is in **P/poly** but not in **P**.

Solution. We will describe a decidable language in P/poly and not in P in the following steps:

Step 1: From Time Hierarchy Theorem, we know that DTIME$(2^{n^k}) \subset$ DTIME(2^{2^n}). Let $A \in$ DTIME(2^{2^n}) and $\notin$ DTIME(2^{n^k}).

Step 2: A is decidable because it is in DTIME(2^{2^n}). Let M_A be a deterministic Turing machine which decides A, and runs in $O(2^{2^n})$ time and not in $O(2^{n^k})$ time.

Step 3: Define the *unary language* U_A *of* A as follows:

$$1^n \in U_A \Leftrightarrow (n)_2 \in A, \quad \text{where } (n)_2 \text{ represents the binary representation of } n.$$

Step 4: U_A is decidable because A is decidable.

Step 5: U_A is in DTIME(2^{n^k}): On input 1^n, we can convert n to $(n)_2$, where $(n)_2$ takes log n space. Now $M_A((n)_2)$ runs in $O(2^{2^{\log\, n}}) = O(2^{n^k})$ time.

Step 6: U_A is not in **P**: Assume on contrary that $U_A \in$ **P**. Then there exists a deterministic polynomial time Turing machine M' which decides U_A. Let $x = (n)_2$ be input to M_A i.e. we want to decide whether $x = (n)_2 \in A$. It is equivalent to deciding whether $1^n \in U_A$, which we can do by running $M'(1^n)$ in $O(2^{|x|^k})$ time, since $|1^n| = 2^{|x|}$. This means we can decide A in DTIME(2^{n^k}), which is a contradiction.

Step 7: $U_A \in$ **P/poly**: Since every unary language is in **P/poly**.

Therefore we have showed that U_A is a decidable language in **P/poly** but not in **P**. ■

(4) If **NP** $=$ **P**$^{\text{SAT}}$ then **NP** $=$ **co-NP**.

Solution. Let $L \in$ **NP**. From hypothesis, $L \in$ **P**$^{\text{SAT}}$. This means there exists a polynomial time Turing machine M with oracle SAT which decides L. Let M' be as follows: On input x, run $M(x)$ and return the complement of $M(x)$. It's clear that M' decides L^c and it follows that $L^c \in$ **P**$^{\text{SAT}}$. Again, from hypothesis, $L^c \in$ **NP**. By definition, $L \in$ **coNP**. Therefore, **NP** $\subseteq$ **coNP**. Analogously, we can prove that **coNP** $\subseteq$ **NP**. Hence **NP** = **coNP**. ■

(5) If **NP** $\subseteq$ **BPP** then **NP** $=$ **RP**.

Solution. We will first show that **RP** $\subseteq$ **NP**. Let $L \in$ **RP** and let M be the probabilistic Turing machine which decides L. Construct a non-deterministic Turing machine M' as follows:

On input x, M' guesses a computation path. Now there are two cases:

(a) $x \in L$: By definition of **RP**, $2/3^{rd}$ of the computation paths of $M(x)$ will accept. $M'(x)$ accepts since $M'(x)$ requires only one accepting path to accept.
(b) $x \notin L$: By definition of **RP**, none of the computation paths of $M(x)$ will accept. $M'(x)$ rejects, since $M'(x)$ requires all paths to be rejecting paths.

Therefore, M' decides L. This shows that **RP** $\subseteq$ **NP**.

Since **RP** is closed under poly-time reductions, to show that **NP** $\subseteq$ **RP**, it suffices to show that SAT $\in$ **RP**. By assumption, SAT $\in$ **BPP**. Let $M \in$ **BPP** be a probabilistic Turing machine which decides SAT. By error reduction (i.e. running several M's simultaneously and taking the majority's result), we can assume that M gives the correct output with probability atleast $\frac{1}{2^k}$, where k is the size of the input (size of Boolean formula). Let ϕ be a Boolean formula of n variables, $\{x_1, \ldots, x_n\}$. The following algorithm will show that SAT $\in$ **RP**:

Algorithm 6 SAT-RP(ϕ)

```
Run M(φ)
if M accepts then
  for i in range {1, ..., n} do
    Assign x_i = 1
    Run M(φ)
    if M accepts then
      Assign x_i = 1 permanently from now onwards i.e. in the further iterations of this for loop
    else
      Assign x_i = 0 permanently from now onwards i.e. in the further iterations of this for loop
    end if
  end for
  Check whether the assigned values actually satisfies φ
  if the assignment satisfies φ then
    return Accept φ
  else
    return Reject φ
  end if
else
  return Reject φ
end if
```

Now we will show that SAT-RP is in **RP**:

(a) Suppose ϕ is satisfiable: We will show that SAT-RP rejects ϕ with probability at most $1/2$. To reject ϕ, either SAT-RP would have to return "reject" for a variable assignment which is a part of satisfying assignment or reject in the beginning only. Thus there are at

most $(n+1)$ calls to M, which means the probability of rejection is $\frac{n+1}{2^{|\phi|}} < \frac{1}{2}$, since $n+1 < |\phi|$.

(b) Suppose ϕ is not satisfiable: In this case, it is clear from the algorithm that SAT-RP rejects ϕ with probability 1.

Hence SAT in **RP**. ■

(6) **BPP** $\subseteq$ **P/poly**.

Solution. Let $L \in$ **BPP**. This means there exists a polynomial time Turing machine M and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$, $Pr_{r\in\{0,1\}^{p(|x|)}}[M(x,r) = L(x)] \geq 1 - \frac{1}{2^{|x|}}$ (such a Turing machine exists by Error reduction). We will show that $L \in$ **P/poly** by giving a family $\mathcal{C}_n$ of polynomial sized circuits.

Let $x \in \{0,1\}^*$ with $|x| = n$. For a given x, we say a sequence of choices for M is "good" if $M(x,r) = L(x)$, "bad" otherwise. While running M with input x, there are in total $2^{p(n)}$ possible sequences of choices for M. From definition of **BPP**, there are strictly less than $2^{p(n)-n}$ sequence of "bad" choices. Summing over all $x \in \{0,1\}^n$, there are strictly less than $2^{p(n)}$ sequence of "bad" choices.

This means, there exists a sequence of "good" choice, of length $p(n)$. Call this sequence of "good" choice as α_n. Thus for all $x \in \{0,1\}^n$, $M(x,\alpha_n) = L(x)$. Then $\mathcal{C}_n$ is described as follows: On input x where $|x| = n$, we simulate $M(x,\alpha_n)$ i.e. α_n is hard-wired in the circuit. Clearly this is a polynomial sized circuit. Thus we have a family $\{\mathcal{C}_n\}$ of polynomial sized circuits deciding L. Hence $L \in$ **P/poly**. ■

Problem 5. Show the following.

(1) Prove that in the certificate definition of **NL** (section 4.4 .1 of Arora-Barak) if we allow the verifier machine to move its head back and forth on the certificate, then the class being defined changes to **NP**.

(2) Show that the following language is **NL**-complete.

$$\{\langle G\rangle \mid G \text{ is a strongly connected digraph}\}$$

Solution. Let us prove part (1) first. So suppose in our certificate definition of **NL**, we allow the verifier machine to move its head back and forth on the certificate. Now we know that 3SAT is an **NP**-complete problem. So, it is enough to show that 3SAT is acceptable by a machine which works as per this definition (because if this is true, then given a language $L \in$ **NP**, our certificate for an input x will simply be the formula ϕ that x is mapped to by the polynomial time Karp reduction from L to 3SAT).

Consider the following machine M that fits in our modified certificate definition of **NL** that accepts the language 3SAT.

(1) Suppose the input is ϕ, where ϕ is a formula in 3-CNF, and suppose ϕ contains k clauses. The certificate u for ϕ will simply be an assignment of values for each variable in the k clauses. If there are k clauses, then note that $|u| = 3k$, since each clause contains three variables.

(2) M scans u from left to right, and it puts three bits of u on its work tape at a time (clearly, this takes logarithmic space, because we are putting exactly three bits at a time). For the current three bits written on the work tape, M checks whether this assignment of values to the variables in the corresponding clause makes that clause 1 or 0. If the clause is 0, then we reject. If all of u is checked and and each clause is 1, then we move to step (3).
(3) In step (3), M checks whether u is a *valid* assignment for ϕ, i.e if each clause has the same value assigned to the same variable. To check this, M will move left and right on its certificate tape (and this is where the modification in the definition comes to play), and clearly this is done in polynomial time. If the assignment is valid, then accept. Otherwise reject.

It is easy to see that M is a polynomial time TM that takes log-space, and it fits with our modified certificate definition of **NL**. This shows that the modified definition is a definition for **NP**.

Now, we will prove part (2). Define the language

$$\text{STRONGLY-CONNECTED} := \{\langle G\rangle \mid G \text{ is a strongly connected digraph}\}$$

We have already proven in class that PATH is an **NL**-complete problem. So, to show that STRONGLY-CONNECTED is **NL**-complete, we will show that PATH is log-space reducible to STRONGLY-CONNECTED.

But first, let us show that STRONGLY-CONNECTED is in **NL**. Since **NL** $=$ **co-NL** (which was proven in class), it is enough to show that $\overline{\text{STRONGLY-CONNECTED}} \in$ **NL**. We do this as follows.

(1) Suppose the input is $\langle G\rangle$. Non-deterministically select two nodes a, b of G.
(2) Use the log-space algorithm for PATH on the input $\langle G, a, b\rangle$. If this algorithm rejects, then accept, because in that case G is not strongly connected. Otherwise reject.

Clearly, the above algorithm is a non-deterministic log space algorithm that correctly accepts $\overline{\text{STRONGLY-CONNECTED}}$, and hence STRONGLY-CONNECTED is in **NL**.

Finally, let us log-space reduce PATH to STRONGLY-CONNECTED. The reduction is quite simple, and works as follows:

(1) Suppose we have the input $\langle G, s, t\rangle$ to PATH. If the input is not of this form, then we just map the input string to a garbage string that does not lie in STRONGLY-CONNECTED.
(2) Copy all of G on the output tape. Clearly, this takes log-space, because we need to store one-bit at a time.
(3) Run a for loop from $i = 1$ to $i = |V|$, where $|V|$ is the number of vertices in G. For each i, write an edge from i to s on the output tape, and write an edge from t to i on the output tape. Again, this takes log-space, because we only need to store the value of the counter i.

We claim that this is the required reduction. Observe that if there is a path from s to t in G, then the resultant graph is strongly connected: if i, j are two distinct vertices, then paths from i to j and j to i respectively are $i \to s \to t \to j$ and $j \to s \to t \to i$ respectively. Similarly, if there is no path in G from s to t, then

the resultant graph will not be strongly connected either. This is because there will be no s to t path in the resultant graph, since the only edges we are adding are into s and out of t, so it cannot make any new s to t paths. ■

Problem 6. Let us define $\mathrm{Maj}_n : \{0,1\}^n \to \{0,1\}$ as

$$\mathrm{Maj}_n(x_1, ..., x_n) = \begin{cases} 1 & , \text{ if } \Sigma_i x_i \geq n/2 \\ 0 & , \text{ otherwise} \end{cases}$$

Prove that Maj_n can be computed by a circuit of size $O(n)$.

Solution. ■

Problem 7. Let's say a language $L \subseteq \{0,1\}^*$ is in **P/poly** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$, a sequence of strings $\{\alpha_n\}_{n\in\mathbb{N}}$ with $\alpha_n \in \{0,1\}^{p(n)}$, and a deterministic polynomial time TM M such that for every $x \in \{0,1\}^n$

$$x \in L \iff M(x, \alpha_n) = 1$$

Let us call α_n to be the *advice string* for all x of length n. Note that the advice string is *not* similar to a witness or certificate as used in the definition of **NP**. For example, all unary languages, even UHALT which is undecidable, are in **P/poly** because the advice string can be a single bit that tells us if the given unary string is in UHALT or not.

A set $S \subseteq \Sigma^*$ is said to be *sparse* if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for each $n \in \mathbb{N}$, the number of strings of length n in S is bounded by $p(n)$. In other words, $|S_{=n}| \leq p(n)$, where $S_{=n} \subseteq S$ contains all strings in S that are of length n.

(1) Give the definition of **P/poly** as given in the class using polynomial size circuit families. Briefly describe how does this definition and the one given using advice string define the same class.

Solution. Definition: **P/poly** is the class of languages that are decidable by polynomial-sized circuit families. In other words, let $\{\mathcal{C}_n\}$ be a circuit family, where each $\mathcal{C}_n$ has polynomial size (w.r.t. n i.e. the number of inputs). **P/poly** is the class of languages that are decidable by $\{\mathcal{C}_n\}$.

Now we will show that the two definitions define the same class.

Let L be a language decided by $\{\mathcal{C}_n\}$. For any input of size n, there exists $\mathcal{C}_n$ of polynomial size (w.r.t. n), such that it decides the input. Let α_n be the description of $\mathcal{C}_n$. α_n is of polynomial size. Describe Turing machine M as follows: On any input x of length n, simulate x on α_n. M is a deterministic polynomial Turing machine because it simulates $\mathcal{C}_n$. The advice string is α_n.
Let L' be a language decided by a Turing machine M' and advice string α'_n. Describe $\mathcal{C}'_n$ as follows: As the advice string α'_n is fixed, $\mathcal{C}'_n$ is a Boolean circuit that simulates M', with α'_n hard-wired into the circuit. Since M' is a polynomial Turing machine and α'_n is also polynomial, $\mathcal{C}'_n$ is also of polynomial size.

Hence we have showed that both the definitions are equivalent. ■

(2) Given $k \in \mathbb{N}$ sparse sets $S_1, ..., S_k$, show that there exists a sparse set S and a deterministic polynomial time TM M with oracle access to S such that given an input $\langle x, i \rangle$ the TM M will accept it if and only if $x \in S_i$. Define the set S (note that it need not be computable), and give the description of M with oracle S. Note that a TM M with oracle access to S can query whether $s \in S$ and get the correct answer in return in constant time.

Solution. We define our set S as

$$S := \{x01^i \mid x \in S_i \text{ for some } 1 \leq i \leq k\}$$

So, S is the string containing all strings x which belong to some S_i, along with the suffix 01^i which represents the index i. We claim that S is a sparse set. To show this, let $n \in \mathbb{N}$ be fixed such that $n \geq 2$ (note that S does not contain any string of size 0 or 1). We bound the number of strings of size n in S. For each $1 \leq i \leq k$, let p_i be the polynomial that bounds the number of strings of size $n - i - 1$ in the set S_i. So, observe that

$$|S_{=n}| = \sum_{i=1}^{k} |S_{=n-i-1}| \leq \sum_{i=1}^{k} p_i(n - i - 1)$$

Clearly, the last sum is a polynomial in n (because k is fixed and does not depend on n). So, it follows that S is a sparse set.

Now, the deterministic polynomial time TM M with oracle access to S works as follows.

(a) On input $\langle x, i \rangle$, generate the string $x01^i$. This clearly takes polynomial time.
(b) Query the oracle S to check whether $x01^i \in S$. Return the corresponding answer.

Clearly, the TM M works operates in polynomial time and accepts the required language. This completes the proof. ■

(3) Let us define a variant of **P/poly** called $\textbf{P/poly}_{\text{det}}$ with a constraint that there should exist a polynomial time algorithm that can compute the advice string for any length $n \in \mathbb{N}$. In other words, there is a polynomial time algorithm A such that $\alpha_n = A(n)$. Is $\textbf{P} = \textbf{P/poly}_{\text{det}}$? Is $\textbf{NP} = \textbf{P/poly}_{\text{det}}$? Justify.

Solution. This is relatively straightforward. We claim that $\textbf{P/poly}_{\text{det}} = \textbf{P}$. It is clear that $\textbf{P} \subseteq \textbf{P/poly}_{\text{det}}$, and so we only need to prove the reverse inclusion. So, suppose $L \in \textbf{P/poly}_{\text{det}}$, and let $\{\alpha_n\}$ be the sequence of advice strings. By definition, we know that there exists a polynomial time TM M such that

$$x \in L \iff M(x, \alpha_{|x|}) = 1$$

Also, we know that there is a polynomial time TM M' that can compute the string α_n from the input n. So, consider a polynomial time algorithm for L that works as follows.

(a) On input x, first generate the string $\alpha_{|x|}$ using the polynomial time TM M'. This clearly takes time polynomial in $|x|$.
(b) Run the machine M on the input $\langle x, \alpha_{|x|} \rangle$. Return the corresponding answer. Again, this step takes time polynomial in $|x|$.

Clearly, the above polynomial time algorithm correctly decides the language L, and hence this shows that $L \in$ **P**, which means **P/poly**$_{\text{det}} \subseteq$ **P**. So, it follows that **P** $=$ **P/poly**$_{\text{det}}$.

The question of determining whether **NP** $=$ **P/poly**$_{\text{det}}$ is then the same as determining whether **P** $=$ **NP**. ■

(4) Let the language $L \in$ **P/poly**. Show that there exists a sparse set S_L and a deterministic polynomial time TM M with oracle access to S_L that can decide the language L.

Solution. From Problem 7.1, we know that there exists a deterministic Turing machine M' and a sequence of advice strings $\{\alpha_n\}$ such that for every $x \in \{0,1\}^n$:

$$x \in L \Leftrightarrow M'(x, \alpha_n) = 1.$$

Define S as follows:

$$S = \{1^n\#a \mid a \text{ is a prefix of } \alpha_n, n \in \mathbb{N}\}.$$

For any n, there are clearly polynomial number of strings of length n (there are $|\alpha_n|$ prefixes of α_n). Thus S is a sparse set. We now describe M^S as follows:

(a) Let input be x, with $|x| = n$.
(b) Initialise α_n as empty string. Our goal is to find the advice string α_n by making polynomial number of queries to oracle S, determining one bit at a time.
(c) Query oracle if $1^n\#0,\ 1^n\#1 \in S$. The result will determine the first bit of α_n. Assume that we have found till now that $1^n\#b \in S$ i.e. in other words b is a prefix of α_n. Then query the oracle if $1^n\#b0,\ 1^n\#b1 \in S$. The result will determine the next bit of α_n. Continue this until both the queries result in false, indicating that we have reached the end of the α_n.
(d) So now we have determined the advice string α_n. Run $M'(x, \alpha_n)$.

The above runs in polynomial time because: We made polynomial number of queries to S (each query takes constant time) and also $M'(x, \alpha_n)$ runs in poly-time. It is also clear that it decides L. Thus L is decided by M^S, where S is a sparse set. ■