MIRI, MAMME, MPAL – Computational Complexity – Homework 2

Exercise 1: Prove that $\mathbf{BPP}^{\mathbf{BPP}} = \mathbf{BPP}$.

Solution: The inclusion $\mathbf{BPP} \subseteq \mathbf{BPP}^{\mathbf{BPP}}$ is clear. Let $L \in \mathbf{BPP}^{\mathbf{BPP}}$ and let M be a probabilistic Turing machine that decides if $x \in L$ in time $p(|x|)$, with error probability at most α (both if $x \in L$ and if not); with access to an oracle for $A \in \mathbf{BPP}$. Furthermore, let N be a probabilistic Turing machine that decides if $y \in A$ in time $q(|y|)$, with error probability at most β (both if $y \in A$ and if not).

Let M' be the probabilistic Turing machine that simulates M, but instead of querying the oracle, it simulates N on the input to the oracle. On an input x, The number of simulated queries is at most $p(|x|)$ (one for each step of M). Because M must write each query to the oracle, the inputs to N are of length at most $p(|x|)$. Therefore, the total time of M' is bounded by

$$q(p(|x|)) \cdot p(|x|),$$

which is polynomial in $|x|$. The probability of M' giving the correct answer is

$$\begin{aligned}
\gamma \geq & \mathbb{P}(\text{all simulated queries give the correct answer and } M' \text{ gives the correct answer}) \\
= & \mathbb{P}(\text{all simulated queries are correct}) \cdot \mathbb{P}(M' \text{ gives the correct answer} \mid \text{all simulated queries are correct}) \\
\geq & \mathbb{P}(\text{all simulated queries are correct}) \cdot (1-\alpha) \\
\geq & (1-\beta)^{p(|x|)} \cdot (1-\alpha).
\end{aligned}$$

The two last inequalities come from the fact that the randomness in the computational path of the simulated M and the simulated oracle calls are all independent, and that M' perfectly simulates M whenever all the simulated oracle calls give the correct answer.

We have seen in class that the error rates of probabilistic Turing machines can be made arbitrarily small while keeping the runtime of the machines polynomial. In fact, we saw that they can be made less than

$$\frac{1}{2^{t(|x|)}}$$

for any polynomial t and input x. For us to prove that $L \in \mathbf{BPP}$, it suffices to show that $\gamma \geq \frac{3}{4}$. For example, setting

$$\alpha \leq \frac{1}{2^3} = \frac{1}{8}$$

and

$$\beta \leq \frac{1}{2^{10p(|x|)}} \leq \frac{1}{10p(|x|)},$$

we obtain

$$\gamma \geq \left(1 - \frac{1}{10p(|x|)}\right)^{p(|x|)} \left(1 - \frac{1}{8}\right) \geq \frac{9}{10} \cdot \frac{7}{8} > \frac{3}{4},$$

where the middle inequality comes from the fact that the function

$$f(n) = \left(1 - \frac{1}{10n}\right)^n$$

is increasing and $f(1) = \frac{9}{10}$.

Exercise 2: Describe the original definition of the advice class $\mathbf{P}|\text{poly}$ and give a high level explanation of why it is equivalent to the class $\mathbf{SIZE}(\text{poly})$.

Solution: The advice classes $C|A$ are defined [1] as the set of languages L for which there exists a language $L' \in C$ and a function $f_L : \mathbb{N} \to \{0,1\}^*$ such that the function $n \mapsto |f_L(n)|$ is in A (i.e., for $A = \text{poly}$, $|f_L(n)|$ is polynomially bounded in n) satisfying

$$L = \{x \in \{0,1\}^* \mid (x, f_L(|x|)) \in L'\}.$$

Intuitively, the language L becomes a problem in C by giving it some advice $f_L(|x|)$, which depends only on the length of the input x and whose length $|f_L(|x|)|$ is bounded by a function in A. This is reminiscent of the definition of $\mathbf{SIZE}(\mathcal{C})$, in which the logic gates can depend on the input size. The definition, while perhaps intricate, precisely defines L based on $L' \in C$ and the pre-determined advice $f_L(|x|)$. For any input x, its membership in L is determined by whether $(x, f_L(|x|))$ is in L'; the advice $f_L(|x|)$ is an integral part of this decision process for each input length.

In our case, we have $C = \mathbf{P}$ and $A = \text{poly}$, which means that the language L can be decided by a polynomial-time Turing machine given some advice f_L of length bounded by a polynomial in the length of the input. In this case, the classes $\mathbf{P}|\text{poly}$ and $\mathbf{SIZE}(\text{poly})$ coincide. Let us sketch the proof of this by double inclusion.

The inclusion $\mathbf{P}|\text{poly} \subseteq \mathbf{SIZE}(\text{poly})$ is similar to the one we saw in class, where we showed that

$$\mathbf{P} \subseteq \mathbf{SIZE}(\text{poly})$$

by a reduction from a turing machine with a single tape, with extra states to simulate the read/write head, where the transformation was local in the sense that the sub-circuits representing each position of the tape were only connected to the sub-circuits representing the next and previous positions. In that case, the circuit just became bigger for each input length, as the number of positions we needed to simulate grew polynomially with the input length. Let $n \in \mathbb{N}$ be a fixed input length. To simulate the advice string $f_L(n)$, we "hard-code" the input positions of the advice string into the circuit S' corresponding to L' for input size $n + |f_L(n)|$. These positions will behave just like the input positions of the tape but will no longer be connected to the actual input. We can do this because the advice function only depends on the length of the input, and not on the input itself. One way to hard-code the advice string into the circuit is to, for example, let p be the first position of the actual input, and codify each position corresponding to the ith bit of $f_L(n)$ (that is, the $(n+i)$th position of the simulated input where $1 \leq i \leq |f_L(n)|$) as

$$\begin{aligned} p \text{ OR } \quad (\neg p), \text{ if } f_L(n)_i = 1 \\ p \text{ AND } (\neg p), \text{ if } f_L(n)_i = 0 \end{aligned}$$

The number of gates in S' (the circuit for L') is a polynomial in $n + |f_L(n)|$ (the total length of the actual input x and the advice string $f_L(n)$), which in turn is bounded by a polynomial in n because $|f_L(n)|$ (the length of the advice) is polynomially bounded in n. The number of extra gates needed to do the hard-coding of $f_L(n)$ is linear in $|f_L(n)|$, as we codify each bit independently. Therefore, we obtain a circuit S whose size is bounded by a polynomial in n and outputs 1 for an input x of length n if and only if

$$(x, f_L(n)) \in L' \iff x \in L.$$

Now we show the other inclusion, $\mathbf{SIZE}(\text{poly}) \subseteq \mathbf{P}|\text{poly}$. Let $L \in \mathbf{SIZE}(\text{poly})$. This means that for each input length n, there exists a circuit S_n that decides L for all inputs of length n, and the number of gates of S_n (denoted $|S_n|$) is bounded by a polynomial in n. We want to find a polynomial-time Turing machine M and an advice string family $f_L(n)$ whose length is polynomially bounded in n, such that for all inputs x of length n, $M(x, f_L(n))$ accepts if and only if $x \in L$, that is, if S_n accepts x. The idea is to encode a description of the circuit S_n into the advice string $f_L(n)$ (which can be done because $|S_n|$ is polynomially bounded in n), and then let M simulate S_n on the input x. We need to find a succinct description $\langle S_n \rangle$ of S_n and describe a polynomial-time Turing machine M that simulates S_n given the input x and $f_L(n) = \langle S_n \rangle$.

We assume gates are topologically sorted $g_1, g_2, \ldots, g_{|S_n|}$, meaning if gate g_k inputs to g_j, then $k < j$. The first n gates, $g_1, \ldots, g_n$, correspond to the input bits $x_1, \ldots, x_n$. Gates $g_{n+1}, \ldots, g_{|S_n|}$ are internal logic gates. One gate (e.g., $g_{|S_n|}$) is the output gate.

The advice string $f_L(n) = \langle S_n \rangle$ can specify:

(a) **Total number of gates:** The value $|S_n|$, encoded in binary ($\mathcal{O}(\log |S_n|)$ bits).

(b) **Gate Information:** For each logic gate g_j ($j \in \{n+1, \ldots, |S_n|\}$):

- Its type (e.g., integer code for INPUT, AND, OR, NOT; $\mathcal{O}(1)$ bits).
- Indices of its input gate(s) (e.g., (k_1, k_2) for binary gates, k_1 and a dummy value for unary, where $k_1, k_2 < j$). Each index takes $\mathcal{O}(\log |S_n|)$ bits.

The total length of $\langle S_n \rangle$ is $\mathcal{O}(|S_n| \log |S_n|)$. Since $|S_n| \in \text{poly}(n)$, $\log |S_n| = \mathcal{O}(\log n)$, so the length of $\langle S_n \rangle$ is $\mathcal{O}(\text{poly}(n) \log n)$, which is polynomially bounded in n.

The language L' is defined as

$$\textbf{SIM} = \{(x, \langle S \rangle) \mid S \text{ has } |x| \text{ inputs and accepts } x\}.$$

Therefore,

$$L = \{x \in \{0,1\}^* \mid S_{|x|} \text{ accepts } x\} = \{x \in \{0,1\}^* \mid (x, f_L(|x|)) \in \textbf{SIM}\}.$$

If we show that $\textbf{SIM} \in \textbf{P}$, we will have shown that $L \in \textbf{P}|\text{poly}$. We now provide pseudocode for the Turing machine $M_{\textbf{SIM}}$ that simulates S on x given $(x, \langle S \rangle)$.

Algorithm 1 Circuit Simulation by $M_{\textbf{SIM}}$

```
procedure SimulateCircuit(x, ⟨S_n⟩)
    N_gates, GATES ← ⟨S_n⟩                ▷ Parse the circuit description into a list of gates
    n ← |x|.
    Declare GateValues[1 ... N_gates]      ▷ Initialize an array to store the values of all gates
    for i ← 1 to n do
        GateValues[i] ← x_i                ▷ x_i is the i-th bit of x
    end for
    for j ← n + 1 to N_gates do
        (t, k_1, k_2) ←GATES[j].           ▷ Get the type and input indices of gate j
        if t is NOT then
            GateValues[j] ←¬GateValues[k_1]
        else if t is AND then
            GateValues[j] ←GateValues[k_1] ∧ GateValues[k_2]
        else if t is OR then
            GateValues[j] ←GateValues[k_1] ∨ GateValues[k_2]
        end if
    end for
    return GateValues[N_gates]
end procedure
```

The algorithm clearly runs in polynomial time, so we have shown that $\textbf{SIM} \in \textbf{P}$, concluding our proof that

$$\textbf{SIZE}(\text{poly}) \subseteq \textbf{P}|\text{poly}.$$

References

[1] Richard Karp. Turing machines that take advice. *Enseign. Math.*, 28:191–209, 1982.