Exercise 1: Machines with certificates

We consider a variant of Turing Machines that are like usual deterministic TMs except that they also carry an extra read-only input tape (not a worktape) called a certificate tape that is read-once (on this tape the reading head may only move rightwards or stay motionless). We write $M(x, u)$ for the output of such a machine $M$ started with $x$ on input tape and $u$ on the certificate tape. We say $M$ runs in space $f(n)$ if for any input $x$ of size $n$ and any certificate $u$, the machine halts and uses at most $f(n)$ cells on its worktapes.

We now consider the class $C$ of all languages $L$ such that there exists a polynomial $p$ and a deterministic TM-with-certificate running in logspace such that

$$x \in L \iff \exists u : |u| \leq p(|x|) \text{ and } M(x, u) \text{ accepts.}$$

(*) 1. Show that $C$ coincides with $NL$.

**Solution:**

$(C \subseteq NL)$: Let $L \in C$ as witnessed by $M$ and $p$. Consider $M'$ that is like $M$ instead that it has no certificate tape. Instead of reading the next character from $u$, $M'$ relies on nondeterminism and guesses that character (it also count the guesses and will guess at most $p(n)$ characters). This $M'$ is a nondeterministic TM running in logarithmic space and it accepts $x$ iff $M$ accepts it for some $u$. Hence $L \in NL$.

$(NL \subseteq C)$: Let $L \in NL$ as witnessed by $M$, a nondeterministic logspace standard TM. Assume w.l.o.g. that $M$ has only one worktape and that exactly two rules are applicable in each of its nondeterministic states. There is a polynomial $p$ such that $M$ runs in $p(n)$ time, hence makes $p(n)$ nondeterministic choices, for $n \overset{\text{def}}{=} |x|$.

Let now $M'$ be the TM with certificate that behaves like $M$ except that it expects that $u$ contains a list $(c_t)_{0 \leq t < p(n)}$ of boolean values used to decide which rule to apply...
in each nondeterministic step: \( M' \) is a deterministic TM with certificate that accepts \( x \) for some \( u \) iff \( M \) accepts it (\( M' \) rejects when \( u \) is not of the expected shape). The length of \( u \) is bounded by \( p(n) \). Hence \( L \in \mathcal{C} \).

(Another approach is to show that GAP —or any other NL-complete problem— is in \( \mathcal{C} \) but this also requires showing that \( \mathcal{C} \) is closed under logspace reductions.)

2. Let us now assume that TMs with certificate are allowed to move leftwards on the certificate tape (hence \( u \) can be read several times but the certificate tape is still read-only) and define \( \mathcal{C} \) as previously.

Prove that \( \mathcal{C} \) coincides with one of \( \text{NL}, \text{PTIME}, \text{NP} \) or \( \text{PSPACE} \) (and tell which one).

**Solution:**

For these new machines \( \mathcal{C} = \text{NP} \).

\( (\mathcal{C} \subseteq \text{NP}) : \) We proceed as in question 1: \( M' \) guesses the contents of \( u \) instead of reading it. The difference is that these guesses must be stored on an extra worktape in case \( M \) revisits some cells, thus \( M' \) is no longer logspace. The running time is exactly as in \( M \) except for an extra initial phase of guessing and storing \( u \) but since \( |u| \leq p(n) \) this only adds a polynomial overhead and \( M' \) runs in polynomial time like \( M \). Finally \( L \) is indeed in \( \text{NP} \).

\( (\text{NP} \subseteq \mathcal{C}) : \) Let \( L \) be accepted by a nondeterministic TM \( M \) that runs in time \( p(n) \). Compared to question 1, \( M \) is not necessarily logspace.

Let \( M' \) be a TM with certificate that behaves like \( M \) but assumes that it is provided with a certificate \( u \) that resolves all the nondeterministic choices (as in the previous question) and that also contains a prediction \((a_{i,t})_{0 \leq i,t \leq p(n)}\) of the contents of each working memory cell at each computation time. Now \( M' \) simulates \( M \) on \( x \) and instead of reading a working memory cell, it fetches it on \( u \) (remember that \( u \) can be read multiple times), and instead of writing a character in memory at step \( t \), it checks that the character is indeed what is predicted by \( u \) for the given cell at step \( t + 1 \). If at some point the contents of \( u \) is not consistent with a run of \( M \) on \( x \), \( M' \) rejects. This \( M' \) is a deterministic TM with certificate that runs in logspace. If \( M' \) accepts \( x, u \) then \( u \) describes a valid accepting run of \( M \) on \( x \). If \( M \) has a run that accepts \( x \), there is a contents for \( u \) that will allow \( M' \) to accepts \( x \).

(An variant construction would have \( u \) contain a list of the configurations of \( M \) on an accepting run on \( x \). One has to argue that \( u \) has polynomial size and that a deterministic logspace \( M' \) can check the consistency of \( u \) on \( x \). Another approach would be to show that that \( \mathcal{C} \) is closed under reductions and that \( \text{SAT} \) can be solved by a logspace deterministic TM with multiple-read certificate.)

**Exercise 2 : Odd satisfiability?**

Recall that \( \text{ParitySat} \) asks, given a list \( I = \langle \varphi_1, \ldots, \varphi_m \rangle \) of boolean formulas, whether the number of satisfiable \( \varphi_i \)'s is odd. (The size \( n \overset{\text{def}}{=} |I| \) of an instance is \( \sum_{i=1}^m |\varphi_i| \).)

We are interested in a variant problem: \( \text{ParityFirstSat} \) asks, given \( I \) as above, whether the smallest index \( i \in \{1, \ldots, m\} \) such that \( \varphi_i \) is satisfiable exists and is odd.

(*) 3. Show \( \text{ParityFirstSat} \leq \text{ParitySat} \). (As usual “\( \leq \)” denotes logspace reducibility.)
Solution:
Note that, for both problems, adding an extra \( \bot \) formula at the end of an instance \( I \) does not change its value, so w.l.o.g. we may always assume that \( m \) is odd.

Let us now define \( r \) via

\[
r \left( \langle \phi_1, \ldots, \phi_m \rangle \right) \overset{\text{def}}{=} \varphi_1 \lor \varphi_2 \lor \varphi_3 \lor \cdots \lor \varphi_i \lor \cdots \lor \varphi_m.
\]

If the first satisfiable formula in some instance \( I \) has index \( i \) then \( r(I) \) has exactly \( m + 1 - i \) satisfiable formulas. And if \( I \) contains no satisfiable formulas, then \( r(I) \) has exactly 0 satisfiable formulas. Assuming that \( m \) is odd, \( r(I) \in \text{ParitySat} \) iff \( I \in \text{ParityFirstSat} \). Since \( r \) is logspace, this shows \( \text{ParityFirstSat} \leq \text{ParitySat} \).

4. (*) Does there exists a total function \( \pi \) which, given a pair \( (I, k) \) where \( I \) is a list \( \langle \phi_1, \ldots, \phi_m \rangle \) of boolean formulas and \( k \in \mathbb{N} \) is some number written in binary, returns a boolean formula \( \pi(I, k) = \psi \) such that \( \psi \) is satisfiable if, and only if, at least \( k \) formulas in \( I \) are satisfiable?

Solution:
Obviously such a function \( \pi \) exists, e.g., by defining \( \pi(I, k) = \top \) if \( I \) has \( k \) or more satisfiable formulas, and \( \pi(I, k) = \bot \) otherwise. This is a correct answer.

(Later it will be useful to know that a logspace computable \( \pi \) exists : see question 5. The \( \pi \) above is not defined with any concern about computability. A more clearly computable \( \pi \) could be given via, e.g.,

\[
\pi(I) \overset{\text{def}}{=} \bigvee_{S \subseteq \{1, \ldots, m\}, |S| = k} \bigwedge_{i \in S} \varphi_i,
\]

but this is not logspace since the output has exponential size.)

5. (**) Show \( \text{ParitySat} \leq \text{ParityFirstSat} \).

Solution:
For this we note that telling whether at least \( k \) formulas in \( I \) are satisfiable is obviously in \( \text{NP} \) (just check that \( k \leq m \), then guess \( k \) formulas from \( I \) and guess valuations for them) so that problem reduces to any \( \text{NP} \)-complete problem, in particular it reduces to \( \text{SAT} \). The corresponding reduction is a logspace-computable function \( \pi \) that is another correct answer to question 4.

With this \( \pi \), define now \( r(I) \overset{\text{def}}{=} \langle \pi(I, m), \pi(I, m-1), \ldots, \pi(I, 0) \rangle \). In \( r(I) \) the first satisfiable formula is \( \pi(I, k) \) where \( k \) is the exact number of satisfiable formulas in \( I \). If we assume that \( m \) is odd, the index of this \( \pi(I, k) \) formula is odd iff \( k \) is odd. Thus \( r \) is a correct reduction from \( \text{ParitySat} \) to \( \text{ParityFirstSat} \). It is logspace-computable since \( \pi \) is.

6. (*) Show that \( \text{ParitySat} \) and \( \text{ParityFirstSat} \) are in \( \text{P}^\text{NP[O(\log n)]} \) (or in \( \text{P}_{||}^\text{NP} \) if you prefer : we saw that these two classes coincide).

Solution:
For both problems it is much easier to show membership in \( \text{P}_{||}^\text{NP} \).

Given an instance \( I \), a possible algorithm is to use \( m \) queries to a \( \text{SAT} \) oracle to decide the satisfiability status of each \( \varphi_i, i = 1, \ldots, m \). These queries can be done
“in parallel” since they are independent (i.e., they can be provided before collecting any of the oracle answers).

Once the oracle has told us which of the \( \varphi_i \)'s are satisfiable, a simple deterministic computation will either count their number and check that this number is odd (for \( \text{ParitySat} \)), or find what is the first index of a satisfiable formula (for \( \text{ParityFirstSat} \)).

(Since we showed that the two problems are inter-reducible, it is possible to only prove membership in \( \mathbf{P}^\mathbf{NP} \| \) for one of them on the condition that one shows, or at least observes, that \( \mathbf{P}^\mathbf{NP} \) is closed under reductions.)

We now consider a new problem, \( \text{SeqSatProgram} \), where one has to solve a sequence of dependent satisfiability problems. Formally, an instance is a “list” of the form

\[
\begin{align*}
x_1 & := \exists Y_1 : \varphi_1(Y_1) \\
x_2 & := \exists Y_2 : \varphi_2(x_1, Y_2) \\
x_3 & := \exists Y_3 : \varphi_3(x_2, Y_3) \\
\vdots \\
x_m & := \exists Y_m : \varphi_m(x_{m-1}, Y_m)
\end{align*}
\]

where each \( Y_i \) is a set of boolean variables disjoint from \( X = \{x_1, \ldots, x_m\} \), and each \( \varphi_i \) is a boolean formula with all its variables in \( Y_i \cup \{x_{i-1}\} \). The meaning is that the \( x_i \)'s should be computed in turn. When computing \( x_i \), one has to solve a satisfiability problem that depends on the value of \( x_{i-1} \).

Seen as a decision problem, the question is whether \( x_m \), the last computed variable, evaluates to true or false. It is clear that \( \text{SeqSatProgram} \) is in \( \mathbf{P}^\mathbf{NP} \).

(*** ) 7. Show that \( \text{SeqSatProgram} \) is in \( \mathbf{P}^{\mathbf{NP}[O(\log n)]} \).

**Solution:**

It is easier to prove membership in \( \mathbf{P}^\mathbf{NP} \| \). Here is a possible algorithm. When \( m > 1 \), one transforms an instance like \( \dagger \) into the following equivalent instance:

\[
\begin{align*}
x'_1 & := \exists Y_1 : \varphi_1(Y_1) \\
x'_2 & := \exists Y_2 : \varphi_2(\top, Y_2) \\
x'_3 & := \exists Y_3 : \varphi_3(\top, Y_3) \\
\vdots \\
x'_m & := \exists Y_m : \varphi_m(\top, Y_m) \\
x''_m & := \exists Y_m : \varphi_m(\bot, Y_m)
\end{align*}
\]

\[
x_m := \exists z_1, \ldots, z_m : [z_m \land z_1 \equiv x_1 \land \land_{i=2}^m z_i \equiv (\text{if } z_{i-1} \text{ then } x'_i \text{ else } x''_i)]
\]

In \( \dagger \), the formula “ 1 1 i 1 1 if 1 then 1 else 1 ” is convenient notation for \( z \equiv z_{i-1} \land x'_i \lor \neg z_{i-1} \land x''_i \), and “ 1 1 1 1 1 1 1 1 1 ” is short for \( (z \lor \cdots) \land (\neg z \lor \cdots) \).

The intuition behind this construction is that, instead of waiting for the value of \( x_{i-1} \) and compute \( x_i \), one can compute two possible values \( x'_i \) and \( x''_i \) for \( x_i \), that correspond to the two cases for \( x_{i-1} \). Later, when we know the actual value of \( x_{i-1} \), we can pick the right value for \( x_i \) among \( x'_i \) and \( x''_i \). The cost is that we end up with twice the number of satisfiability problems but these are independent queries.

To prove the equivalence between \( \dagger \) and \( \ddagger \), the crux is to note that \( \exists z_1, \ldots, z_m : z_1 \equiv x_1 \land \land_{i=2}^m z_i \equiv \ldots \). is always satisfiable and in only one way: each \( z_i \) is assigned the value defined by the \( z_i \equiv \cdots \). expressions.
Finally, can be solved in $P^{NP}$: the first $2m - 1$ lines are independent and the last one is an easy boolean computation that does not require any oracle query (each $z_i$ is evaluated deterministically).