Vous pouvez rédiger votre copie en français ou en anglais.

When giving reductions or algorithms, explain the algorithm at a high level of abstraction but do not forget corner cases or use vague notation. When arguing the correctness of an algorithm, make a precise claim about your construction before proving said claim. In the proof, you may abstain from proving trivial observations only if you do not miss the harder parts of the proof.

Exercise 1 : Time linear in machine size

We choose some way of describing nondeterministic Turing machines $M$, $M'$, etc. via some encoding $\lfloor M \rfloor$, $\lfloor M' \rfloor$, ... over some alphabet $A$. Such an encoding just lists the control states of machine $M$, its alphabet, its transition rules, etc., in some natural way. In particular, it is easy (i.e., logspace computable) to decide if a given string $x \in A^*$ is the encoding of some machine. The length $|\lfloor M \rfloor|$ is the “size” of the machine.

The problem LINEARHALT asks, given $\lfloor M \rfloor$ of size $n$, whether $M$ accepts the empty string in time at most $n$.

1. Show that LINEARHALT is NP-complete.

Solution:
Recall that a universal machine $U$ can read $\lfloor M \rfloor$ and simulate $M$ in polynomial-time. When $M$ is non-deterministic, the behaviour of $U$ will be non-deterministic too. To show that LINEARHALT is in NP we use a version of $U$ that first computes $n$ and then simulates $M$ for at most $n$ steps.

To see that LINEARHALT is NP-hard, we can show, for example, SAT $\leq$ LINEARHALT. Let us pick a fixed non-deterministic machine $M_S$ that solves SAT in some polynomial-time $p(n)$, where $n$ is the size of the formula at hand. We can assume that $M_S$ first copies its input tape on a worktape and then never reads its input tape again.

With a formula $\phi$ our reduction associates a machine $M_{S,\phi}$ that behaves like $M_S$ except that it first writes $\phi$ on the worktape instead of reading the input tape. Thus $\phi$ is hard-coded in the control states of $M_{S,\phi}$. After this phase, $M_{S,\phi}$ behaves like $M_S$ and decides whether $\phi \in$ SAT. The reduction now adds $p(n)$ dummy control states to $M_{S,\phi}$ in order to guarantee that $\lfloor M_{S,\phi} \rfloor$ has size at least $p(n)$. These extra states do not change the behaviour of $M_{S,\phi}$, they just make its encoding longer.

The time taken by $M_{S,\phi}$ is $p(n)$ where $n = |\phi|$ since the preparatory work done by $M_{S,\phi}$ is exactly similar to the first phase of $M_S$ where it copies its input tape. Therefore $M_{S,\phi}$ is in LINEARHALT iff $\phi \in$ SAT.

To see that the reduction is logspace, we note that it only has to generate the encoding of a version of $M_S$ (a fixed machine) to which where $O(n)$ extra states and rules have been added (for hardcoding $\phi$ in the control states of $M_{S,\phi}$) together with $p(n)$ dummy states.
Exercise 2 : Counting complexity

We recall that a parsimonious reduction between two counting problems $F : A^* \rightarrow \mathbb{Z}$ and $G : B^* \rightarrow \mathbb{Z}$, is a logspace-computable mapping $r : A^* \rightarrow B^*$ such that $F(x) = G(r(x))$ for all $x \in A^*$. We use $\leq_{par}$ to denote reducibility via parsimonious reductions.

We also recall that $\#\text{SAT}$ is the problem, given a set of boolean variables $X = \{x_1, \ldots, x_n\}$ and a boolean formula $\phi$ over $X$ (not necessarily in clausal form), to compute the number of valuations of $X$ that satisfy $\phi$. We write $\#\text{3SAT}$ for $\#\text{SAT}$ restricted to formulas in clausal form where each clause has at most three literals.

$\text{CIRCUITSAT}$ is the satisfiability problem for circuits like the following example.

![Circuit Diagram]

A circuit $C$ is an acyclic directed graph with input gates $x_1, \ldots, x_m$ and nand-gates $g_1, \ldots, g_r$. The input gates have in-degree 0 and the nand-gates can have any number of inputs. Given a valuation $v : \{x_1, \ldots, x_m\} \rightarrow \{\top, \bot\}$ of the input gates, the boolean value (written $v(g)_C$) of any gate $g$ in $C$ is determined in the usual way. In the above example, the value of $g_3$ under $v$ is $\land(v(x_1), v(x_2), v(x_4))$, i.e., $\neg(v(x_1) \land v(x_2) \land v(x_4))$, and the value of $g_4$ is $\neg \land \emptyset$, i.e., $\bot$.

Formally, $\text{CIRCUITSAT}$ asks, given a circuit $C$ and a designated output gate $g$, whether there is a valuation with $v(g)_C = \top$, and $\#\text{CIRCUITSAT}$ is the counting version, asking how many valuations yield $v(g)_C = \top$.

2. Show that $\#\text{CIRCUITSAT}$, $\#\text{SAT}$ and $\#\text{3SAT}$ are equivalent under parsimonious reductions.

Solution:

Obviously $\#\text{3SAT} \leq_{par} \#\text{SAT}$ since $\text{3SAT}$ is just a special case of $\text{SAT}$. Similarly, $\text{SAT}$ can be seen as a tree-shaped circuit with inner gates performing $\land$, $\lor$ and $\neg$ operations. These operations can be encoded in $\land$ perhaps by adding extra intermediary gates. E.g. $\land(g, g')$ is realized as $\land(\land(g, g'))$ with one extra gate, $\lor(g, g')$ is realized as $\land(\land(g, \lor(g', \neg g)))$ with two extra gates, and $\neg(g)$ is replaced by a unary $\neg(g)$. This reduction from $\text{SAT}$ to $\text{CIRCUITSAT}$ is obviously a parsimonious reduction witnessing $\#\text{SAT} \leq_{par} \#\text{CIRCUITSAT}$.

For the reduction $\#\text{CIRCUITSAT} \leq_{par} \#\text{3SAT}$, we first simplify the problem by ensuring that every nand gate in $C$ has at most two inputs. This can be guaranteed by a parsimonious reduction that transforms every gate $g = \land(z_1, z_2, \ldots, z_k)$ with $k > 2$ into $g = \land(z_1, \land(z_2, \land(z_3, \ldots \ldots, z_k)))$, introducing $2k - 4$ extra gates.

Now that $C$ has input degree at most two, and given a target gate $g$, we associate a formula $\phi_{C,g}$ that uses $Z = \{x_1, \ldots, x_m, g_1, \ldots, g_r\}$ as boolean variables : that is, we associate a boolean variable with every gate of $C$, not just the input gates. We define $\phi_{C,g} \overset{\text{def}}{=} g \land \psi_1 \land \cdots \land \psi_{\ell}$ where each $\psi_i$ is a conjunction of 3-clauses that can only be satisfied by some $v$ if the variable $g_i$ gets the value $v(g_i)_C$ of the corresponding gate. The $\psi_i$’s are defined by :

- For a nand gate $g_i$ with no input, we let $\psi_i \overset{\text{def}}{=} \neg g_i$ ;
- For a nand gate $g_i$ with one input $z$ (can be some $x_j$ or some $g_j$), we let $\psi_i \overset{\text{def}}{=} (g_i \lor z) \land (\neg g_i \lor \neg z)$,
- For a nand gate $g_i$ with two input $z$ and $z'$ we let $\psi_i \overset{\text{def}}{=} (g_i \lor z) \land (g_i \lor z') \land (\neg g_i \lor \neg z \lor \neg z')$.

It is clear that a valuation on $Z$ satisfies $\bigwedge_{i=1}^{\ell} \psi_i$ if and only if it assigns to any $g_i$ that value $v(g_i)_C$. Thus the valuations on $Z$ that satisfy $\phi_{C,g}$ are in one-to-one correspondence with the valuations on $\{x_1, \ldots, x_m\}$ that make $v(g)_C$ true.
Recall that, for a square matrix $M$, its permanent $P(M)$ is defined as $P(M) \triangleq \sum_{\sigma \in \text{Sym}(n)} \prod_{i=1}^{n} M_{i,\sigma(i)}$ where $\text{Sym}(n)$ is the group of all permutations of the set $\{1, 2, \ldots, n\}$.

With $\text{PERM}_{0, 1}$ we denote the problem of computing the permanent of a square matrix whose entries are all among 0 and 1.

3. Give a reduction showing that $\text{PERM}_{0, 1} \leq_{\text{par}} \#\text{SAT}$.

**Solution:**

With $M$ of dimension $n \times n$ a possible reduction associates a formula $\phi_M^P$ over $2n^2$ boolean variables: the $(s_{i,j})_{1 \leq i,j \leq n}$ that encode a permutation of $1, \ldots, n$ and the $(m_{i,j})_{1 \leq i,j \leq n}$ that encode the values inside $M$. Specifically, $\phi_M^P$ is some $\phi_s \land \phi_M \land \phi_X$ given by

$$\phi_M \triangleq \bigwedge_{1 \leq i,j \leq n} \neg m_{i,j} \land \bigwedge_{1 \leq i,j \leq n} m_{i,j}$$

$$\phi_S \triangleq \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} s_{i,j} \land \bigwedge_{j=1}^{n} \bigwedge_{i=1}^{n} s_{i,j} \land \bigwedge_{1 \leq j < j' \leq n} \neg s_{i,j} \lor \neg s_{i,j'}$$

$$\phi_X \triangleq \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} s_{i,j} \land m_{i,j}$$

Thus $\phi_M$ has $n^2$ clauses with 1 literal each : they can only be satisfied by assigning to the $m_{i,j}$’s the boolean values that encode the content of $M$. We also see that $\phi_S$ start with $2n$ clauses with $n$ literals each (requiring that for any $i$ there is at least one $s_{i,j}$ set to true and for any $j$ there is at least one $s_{i,j}$ set to true) and continue with $\frac{1}{2}n^2(n-1)$ clauses with 2 literals each (requiring that for any $i$ there cannot be two distinct $s_{i,j}$’s set to true) : the conjunction of all these clauses can only be satisfied by choosing a permutation $\sigma$ and setting all $s_{i,j}$ true if and only if $j = \sigma(i)$. There are thus exactly $n!$ valuations that satisfy $\phi_S \land \phi_M$. Finally the last part of $\phi_M^P$ is $\phi_X$ that is only satisfied by some valuation $v$ if $v$ corresponds to a permutation $\sigma$ such that all $m_{i,\sigma(i)}$’s are set to $\top$, i.e., a permutation such that $M_{i,\sigma(i)} = 1$ for all $i$. Finally, the number of valuations satisfying $\phi_M^P$ is exactly the permanent of $M$.

We conclude by observing that the reduction is evidently logspace : it only needs three counters bounded by $n$, hence by the size of $M$, when generating the formula $\phi_M^P$ from $M$.

**Exercise 3 : SPARSE languages.**

We fix an alphabet $\Sigma$ with at least two letters. $\Sigma^n$ is the set of words of length $n$, and $\#E$ denotes the number of elements in the finite set $E$. For simplicity, we shall assume that $\Sigma = \{0, 1\}$, although this is not strictly necessary.

A language $L \subseteq \Sigma^*$ is sparse if and only if there is a polynomial $p$ such that $\#(L \cap \Sigma^n) \leq p(n)$ for all $n \in \mathbb{N}$. One also say that $L$ “has polynomial density”. Let SPARSE be the class of all sparse languages.

4. For $k \in \mathbb{N}$, we define $L_k = \{u \in \{0, 1\}^* : |u|_1 = k\}$, where $|u|$ is the length of $u$ and $|u|_1$ is the number of times the letter 1 occurs in $u$. Show that $L_k \in \text{SPARSE} \cap \text{L}$ for any $k \in \mathbb{N}$.

**Solution:**

$L_k$ is regular hence in L. It is in $\text{SPARSE}$ because for any $n$, $\#L_k \cap \Sigma^n = \binom{n}{k} \leq n^k$.

5. Show that $\text{SPARSE}$ contains some undecidable languages (Indication : Only look for very simple examples).

**Solution:**

Take the language $L$ of strings $1^n$ such that $n$ is the Gödel number of a halting Turing machine (or of any string from some undecidable language). Clearly $L$ is undecidable. It is sparse because $\#(L \cap \Sigma^n) \leq 1$ for any $n$. 

Page 3
6. Given \( L, L_1, L_2 \in \text{NL} \cap \text{SPARSE} \), do we have:
   (1) \( L_1 \cdot L_2 \in \text{NL} \)? (2) \( L_1 \cdot L_2 \in \text{coNL} \)? (3) \( L_1 \cdot L_2 \in \text{SPARSE} \)?
   (4) \( L^* \in \text{NL} \)? (5) \( L^* \in \text{coNL} \)? (6) \( L^* \in \text{SPARSE} \)?

Recall that \( L_1 \cdot L_2 = \{ uv : u \in L_1, v \in L_2 \} \) and that \( L^* = \{ \epsilon \} \cup L \cup L \cdot L \cup L \cdot L \cdot L \cup \cdots \)

**Solution:**

3. Yes: if \( L_1 \) and \( L_2 \) both have density bounded by \( p \) then \( \#(L_1 \cdot L_2 \cap \Sigma^n) \leq \sum_{i=0}^{n} p(i)p(n-i) \) which is in \( O(n \cdot p(n)) \).

6: No, e.g., \( \Sigma \) is sparse but \( \Sigma^* \) is not.

1: Yes. The NL algorithm just has to guess where the input string must be split in two and then check each part. The same reasoning applies to 4.

2 & 5: Yes since \( \text{NL} = \text{coNL} \).

We say that a language belongs to UNARY if it is included in \( \{1\}^* \).

7. Show that \( \text{UNARY} \subset \text{SPARSE} \).

**Solution:**

A language in UNARY contains at most one word for each length \( n \).

8. Show that if \( \text{SPACE}(2^{O(n)}) = \text{TIME}(2^{O(n)}) \), then \( \text{PSPACE} \cap \text{UNARY} \subset \text{P} \) and \( \text{NPSPACE} \cap \text{UNARY} \subset \text{P} \).

**Solution:**

Take a language \( L \) in \( \text{PSPACE} \cap \text{UNARY} \). With \( L \) we associate \( V = \{ n : 1^n \in L \} \). Note that \( V \subseteq \{0,1\}^* \) contains the binary encodings of the lengths of the words in \( L \). Now \( V \in \text{SPACE}(2^{O(n)}) \) since, given some binary \( n \), we can compute \( 1^n \) and call the \( \text{PSPACE} \) algorithm for \( L \) on the exponentially long input \( 1^n \). By hypothesis, we thus have \( V \in \text{TIME}(2^{O(n)}) \). But then we can decide \( L \) by the following method: on input \( x \), check that \( x \) has the shape \( x = 1^n \), compute \( n \) (of size \( \log |x| \)) and use the \( \text{TIME}(2^{O(n)}) \) algorithm for \( V \) on \( n \). This takes a time polynomial in \( |x| \). Hence \( L \in \text{P} \).

We conclude that \( \text{PSPACE} \cap \text{UNARY} \subset \text{P} \), and of course \( \text{NPSPACE} \cap \text{UNARY} \subset \text{P} \) as \( \text{PSPACE} = \text{NPSPACE} \).

We write \( L \leq_P L' \) when \( L \) is polynomial-time reducible to \( L' \).

9. Show that, if \( P = \text{NP} \), then \( \text{SAT} \) is polynomial-time reducible to some non-empty sparse language.

**Solution:**

In that case, \( \text{SAT} \) is in \( P \). The reduction first decides the \( \text{SAT} \) instance \( x \) in polynomial time. If \( x \) is satisfiable, then we return, say, 1, otherwise something else, say 0. This is a polynomial-time reduction showing \( \text{SAT} \leq_P \{1\} \).