Exercise 1 (Too Fast). Show that $\text{ATIME}(\log n) \neq \mathcal{L}$.

Solution 1. When considering $\text{ATIME}(\log n)$, we do not even have the time to read the full input. So any language which is in $\mathcal{L}$ and needs for the input to be completely read will give the result. For instance, one may use the palindromes language, or $0^n$ on a two letter alphabet, or $0^{2k}$ on a one letter alphabet.

Exercise 2 (A translation result.). Show that if $P = \text{PSPACE}$, then $\text{EXPTIME} = \text{EXPSPACE}$.

Solution 2. In any case, we have $\text{EXPTIME} \subseteq \text{EXPSPACE}$. Let us assume that $P = \text{PSPACE}$ and let us show that $\text{EXPSPACE} \subseteq \text{EXPTIME}$. Let $L_1 \in \text{EXPSPACE}$ be a language accepted by a Turing machine $M_1$ running in space $2^{n^c}$, for some $c \geq 1$. We define:

$$L_2 = \{(x, 1^{\lfloor |x|^c \rfloor}) \mid x \in L_1\}$$

A Turing machine $M_2$ which launches $M_1$ on $x$ for an input $w = (x, 1^{\lfloor |x|^c \rfloor})$ (after checking the size of $w$) accepts $L_2$ and runs in space $O(|w|)$. Hence, $L_2 \in \text{PSPACE} \subseteq P$. Therefore, there exists a Turing machine $M_3$ running in polynomial time accepting $L_2$ and runs in exponential time. That is, $L_1 \in \text{EXPTIME}$ and $\text{EXPSPACE} \subseteq \text{EXPTIME}$.

Exercise 3 (Closure under morphism). Given a finite alphabet $\Sigma$, a function $f : \Sigma^* \rightarrow \Sigma^*$ is a morphism if $f(\Sigma) \subseteq \Sigma$ and for all $a = a_1 \cdots a_n \in \Sigma^*$, $f(a) = f(a_1) \cdots f(a_n)$ ($f$ is uniquely determined by the value it takes on $\Sigma$).

Show that $P = \text{NP}$ if and only if $P$ is closed under morphism.

Solution 3. 

- Assume that $P = \text{NP}$. Consider $f$ a morphism and $L \in P = \text{NP}$. Let us show that $f(L) \in \text{NP} = P$. We consider a non-deterministic Turing machine $M$ that, on an input $w \in \Sigma^*$, guesses a word $a \in \Sigma^*$ such that $|a| = |w|$ and then checks that $f(a) = w$ and that $a \in L$ in polynomial time. It follows that $f(L) \in \text{NP} = P$ and $P$ is closed under morphisms.

- Now, assume that $P$ is closed under morphism. We show that $\text{SAT} \in P$, which proves that $\text{NP} \subseteq P$ since $\text{SAT}$ is $\text{NP}$-complete for logspace reductions and $P$ is closed under logspace reductions. Consider the following language:

$$L = \{ (\phi, v) \mid v \text{ is a valuation satisfying } \phi \}$$

We have that $L \in P$ as one can check in polynomial time that a valuation satisfies a boolean formula. Furthermore, we can assume that the alphabet $\Sigma$ is equal to the disjoint union $\Sigma_\phi \cup \Sigma_v$ and the symbols used to encode $\phi$ (resp. $v$) are in $\Sigma_\phi$ (resp. $\Sigma_v$).
Then, if we consider the morphism \( f \) that ensures \( f(a) = a \) for all \( a \in \Sigma_\phi \) and \( f(a) = 0 \) for all \( a \in \Sigma_v \). Then,

\[
f(L) = \{(\phi, 0^n) \mid \phi \text{ has } n \text{ variables and is satisfiable}\}
\]

By closure under morphism, it follows that \( f(L) \in \mathbb{P} \). Since, an instance of SAT can be reduced in polynomial time (in fact, in logarithmic space) to an instance of \( f(L) \), it follows that SAT \( \in \mathbb{P} \). Hence, \( \mathbb{P} = \mathbb{NP} \).

**Exercise 4 (P-choice).** A language \( L \) is said to be \( \mathbb{P} \)-peek, written \( L \in \mathbb{P}_p \), if there is a function \( f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \), computable in polynomial time, such that \( \forall x, y \in \Sigma^* : \)

1. \( f(x, y) \in \{x, y\} \),
2. if \( x \in L \) or \( y \in L \) then \( f(x, y) \in L \).

In that case, \( f \) is called the peeking function for \( L \).

1. Show that \( \mathbb{P} \subseteq \mathbb{P}_p \).
2. Show that \( \mathbb{P}_p \) is closed under complementation.
3. Show that if there exists a \( \mathbb{NP} \)-hard language in \( \mathbb{P}_p \) then \( \mathbb{P} = \mathbb{NP} \).

**Solution 4.**

1. Consider \( L \in \mathbb{P} \). We build \( f \) such that, on an input \( w = (x, y) \), \( f \) checks in polynomial time if \( x \in L \). If so it returns \( x \), otherwise, it returns \( y \). Then, we have \( L \in \mathbb{P}_p \).

2. Consider \( L \in \mathbb{P}_p \) and its peeking function \( f \). Let us consider \( f' \) such that \( f'(x, y) = x \) if \( f(x, y) = y \) and \( y \) otherwise. One can check that \( f' \) is a peeking function the complement of \( L \).

3. Consider a polynomial time reduction \( tr \) from SAT to \( L \in \mathbb{P}_p \) with the peeking function \( f \). We design the following algorithm that runs in polynomial time and decides SAT:

\[
a(s):\nonumber
\]

\[
\text{if } s = \text{True} : \nonumber
\text{accept; } \nonumber
\text{elseif } s = \text{False} : \nonumber
\text{reject; } \nonumber
\text{else : } \nonumber
\text{let } x \text{ in Var}(s); \nonumber
\text{if } f(tr(s[x <- \text{True}]), tr(s[x <- \text{False}])) = tr(s[x <- \text{True}]); \nonumber
\text{a(s[x <- \text{True}]); } \nonumber
\text{else : } \nonumber
\text{a(s[x <- \text{False}])}
\]

Computing \( f \) and \( tr \) can be done in polynomial time and there is \( |\text{Var}(s)| \) recursive call where \( \text{Var}(s) \) is the set of variables appearing in the formula \( s \). Hence the algorithm runs in polynomial time. The correction of the algorithm comes from the definition of a peeking function.
Exercise 5 (Dependency QBF). We have seen that the problem QBF is PSPACE-complete. Let us consider a version of this problem called DQBF that is \textsc{NEXPTIME}-complete.

An instance of DQBF basically consists in an instance of QBF where additionally we are given the possibility to indicate, for each existential variable, the set of universal variables on which it depends. Specifically, an instance of DQBF consists in quantified formula:

$$\Phi = \forall x_1, \ldots, \forall x_n, \exists y_1(D_1), \ldots, \exists y_k(D_k) : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

where, for all $1 \leq i \leq k$, we have $D_i \subseteq \{x_1, \ldots, x_n\}$ the set of universal variables on which the variable $y_i$ may depend.

1. Give formally the condition on which the DQBF-formula $\Phi$ is satisfiable;

2. Show that it is still \textsc{NEXPTIME}-complete if the sets $D_i$ are disjoint, i.e. for all $i, j \leq k$, we have $i \neq j \Rightarrow D_i \cap D_j = \emptyset$;

3. Show that it is in \textsc{PSPACE} if $\varphi$ is in CNF and all the sets $D_i$ are disjoint. Do you think it is \textsc{PSPACE}-complete?

Solution 5. 1. The formula $\Phi$ is satisfiable iff, for all $1 \leq j \leq k$ there exists a function $v_j : D_j \rightarrow \{\text{true, false}\}$ such that, for all valuations $v : \{x_1, \ldots, x_n\} \rightarrow \{\text{true, false}\}$:

$$\varphi(v(x_1), \ldots, v(x_n), v_1[D_1], \ldots, v_k[D_k])$$

is set to true.

2. This problem is still \textsc{NEXPTIME} as a subcase of DQBF. Consider now an arbitrary DQBF-formula:

$$\Phi = \forall x_1, \ldots, \forall x_n, \exists y_1(D_1), \ldots, \exists y_k(D_k) : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

Then we build the new DQBF-formula:

$$\Phi' := \forall x_1, \ldots, \forall x_n, \forall X_1, \ldots, \forall X_k, \exists y_1(X_1), \ldots, \exists y_k(X_k) : \varphi_{-\text{equiv}} \lor \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

where, for all $1 \leq j \leq k$, we have $X_j := \{x_j | x \in D_j\}$ a set of $|D_j|$ fresh variables and $\varphi_{-\text{equiv}}$ is such that:

$$\varphi_{-\text{equiv}} := \bigvee_{1 \leq j \leq k} \bigvee_{x \in D_k} (\neg x \Leftrightarrow x_j)$$

With this construction, if any variable in $D_j$ is not equivalent to its $X_j$ counterpart, then the logical formula trivially holds. If all variables in all $D_j$’s are equivalent to the in $X_j$, then they do not change how the $y$ variables can be chosen. It follows that $\Phi'$ is satisfiable iff $\Phi$ is. Finally, note that the set on which the $y$ variables depend are disjoint and that this transformation in polynomial time.

3. Consider such a DQBF-formula:

$$\Phi = \forall x_1, \ldots, \forall x_n, \exists y_1(D_1), \ldots, \exists y_k(D_k) : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

where, for all $i \neq j$, we have $D_i \cap D_j = \emptyset$. Let $X := \{x_1, \ldots, x_n\}$ and $D := X \setminus (\cup_{1 \leq j \leq k} D_j)$. Now, we have $X = \bigcup_{1 \leq j \leq k} D_j \cup D$. Furthermore, the formula $\varphi$ can be written in the following way:

$$\varphi = \bigwedge_{1 \leq i \leq m} C_j$$
Assume that $\Phi$ is satisfiable and let us denote by $(v_i)_{1 \leq i \leq k}$ the valuation functions of all the $y$ variables. We argue that for all clauses $1 \leq i \leq m$, there exists a pair $(D_i, v_i)$ (which could be $(D, \emptyset)$) such that for all valuations $v : D_i \rightarrow \{\text{true}, \text{false}\}$, we have: $(v(D_i), v_j(v(D_j))) \models C_j$. Indeed, if that is not the case, then there would exist a valuation $v : \{x_1, \ldots, x_n\} \rightarrow \{\text{true}, \text{false}\}$ that would not satisfy the clause $C_j$. In fact, solving the problem only amounts to guessing for each clause, which pair will satisfy it, and then solve the corresponding QBF decision problem. Note here that it may be possible that a given pair has a satisfy a conjunction if clauses, hence these clauses cannot be treated separately. We have the following (non-deterministic) algorithm running in polynomial space with as input the formula $\Phi = \wedge_{1 \leq i \leq m} C_j$:

$$\text{DQBF}(\Phi) :$$

for $1 \leq i \leq m$:

\begin{itemize}
  \item guess $b_i \in \{1, \ldots, k+1\}$
\end{itemize}

for $1 \leq j \leq k+1$:

\begin{itemize}
  \item $C := \text{true}$
  \item for $1 \leq i \leq m$:
    \begin{itemize}
      \item if $b_i = j$:
        \begin{itemize}
          \item then $C := C \wedge C_i$
        \end{itemize}
      \end{itemize}
  \item if $j \leq k$:
    \begin{itemize}
      \item $\text{QBF}(\forall D_{b_j}, \exists y_j : C)$
    \end{itemize}
  \item else:
    \begin{itemize}
      \item $\text{QBF}(\forall D, C)$
    \end{itemize}
\end{itemize}

Exercise 6 (Unary language). Recall that a unary language is any language over a one-letter alphabet. Prove that if a unary language is $\text{NP}$-complete, then $\text{P} = \text{NP}$.

Solution 6. Consider a unary language $L$ (say on the alphabet $\Sigma = \{1\}$) that is $\text{NP}$-complete and a polynomial time reduction $\text{tr}$ ensuring $\phi \in \text{SAT} \iff \text{tr}(\phi) \in L$. There is $a, c \geq 1$ such that we have $|\text{tr}(\phi)| \leq a \cdot |\phi|^c$ for all $\phi$. We design a polynomial time algorithm that solves SAT. Consider a SAT formula $\phi$. For a variable $x$ appearing in $\phi$, we denote by $\phi[x \rightarrow \text{true}]$ the (simplification of the) formula $\phi$ where $x$ is set to $\text{true}$ (and similarly $\phi[x \rightarrow \text{false}]$). Note that $|\phi[x \rightarrow \text{true}]| \leq |\phi|$ and $|\phi[x \rightarrow \text{false}]| \leq |\phi|$

We maintain a list $l$ of pairs $(\text{tr}(\phi), \varphi)$ such that $\phi$ is satisfiable if and only if one of the formula of $l$ is satisfiable while ensuring $|l| \leq 2 \times a \cdot |\phi|^c$ at all time. Initially, we set $l = \{(\text{tr}(\phi), \varphi)\}$. Then, we loop over the variables $x_1, \ldots, x_n$ of $\phi$ and, at each iteration dealing with a variable $x_i$ for some $1 \leq i \leq n$, we proceed in two steps:

- for every pair $p = (\text{tr}(\varphi), \varphi)$ in $l$, we add $(\text{tr}(\text{tr}(\varphi[x_i \rightarrow \text{true}]), \varphi[x_i \rightarrow \text{true}]), \varphi[x_i \rightarrow \text{false}])$ and $(\text{tr}(\text{tr}(\varphi[x_i \rightarrow \text{false}]), \varphi[x_i \rightarrow \text{false}]), \varphi[x_i \rightarrow \text{false}])$ and we remove $p$.
- for all $1 \leq k \leq a \cdot |\phi|^c$, we keep (at most) one pair of the shape $(1^k, \varphi)$ and remove the other from $l$.

By construction, at the end of each iteration, we have $|l| \leq a \cdot |\phi|^c$ because, for all formula $\varphi$ on which $\text{tr}$ is applied, we have $|\text{tr}(\varphi)| \in \{1^k \mid 1 \leq k \leq a \cdot |\phi|^c\}$. Therefore, $l$ is of size at most $2 \cdot a \cdot |\phi|^c$ (this upper bound may be achieved at the end of the first step). Furthermore, if at the beginning of an iteration we have the equivalence that $\phi$ is satisfiable if and only if one of the formula of $l$ is satisfiable, we still have it at the end of the iteration.
Indeed, it is straightforward that this holds at the end of the first step. Furthermore, if $\text{tr}(\varphi) = \text{tr}(\varphi')$ for two formulas $\varphi$ and $\varphi'$, then $\varphi \in \text{SAT} \iff \varphi' \in \text{SAT}$. It follows that the property still holds at the end of the second step and at the end of the iteration. Then, once these iterations are over, the final step consist in checking that the list $l$ contains a pair $(1^k, \text{True})$ for some $k$. The algorithm we described runs in polynomial time and decides $\text{SAT}$. Therefore $\text{SAT} \in \mathbb{P}$. 