Problem 1: Reductions

Consider the following languages:

- \text{formulaSAT} := \{f : f \text{ is a satisfiable Boolean formula}\}.
- \text{SAT} := \{f : f \text{ is a satisfiable CNF formula}\}.
- \text{CLIQUE} := \{(G,k) : G \text{ contains a k-clique}\}. A graph \(G\) is said to contain a k-clique if there is a set of \(k\) vertices in \(G\) such that each of these vertices has an edge to each other vertex (a complete subgraph of size \(k\)).
- \text{INDSET} as described in the lecture (independent set problem; party problem).
- \text{co-TAUTO} := \{f : f \text{ is not a tautology}\}. A tautology is a Boolean formula that is true for any assignments of truth-values to the variables.

For each pair \(A, B \in \{\text{formulaSAT}, \text{SAT}, \text{CLIQUE}, \text{INDSET}, \text{co-TAUTO}\}\), show that \(A \leq_p B\). (I.e., that \(A\) is polynomial-time Karp reducible to \(B\).)

\textbf{Note:} The fact \(\text{SAT} \leq_p \text{INDSET}\) will be shown in the practice on Tuesday, Sep 13.

\textbf{Note:} None of these reductions need an elaborate construction like we use in the practice for showing that \(\text{SAT} \leq_p \text{INDSET}\). Each proof should just be a few lines. You are allowed to use facts we already showed, e.g., \(\text{SAT} \leq_p \text{INDSET}\) and \(\text{SAT}\) is NP-complete.

\textbf{Solution.} We know that \(\text{SAT}\) is NP-complete and thus NP-hard. By definition of NP-hardness, \(A \in \text{NP}\) implies \(A \leq_p \text{SAT}\). Since all the languages listed in this problem are easily seen to be in \(\text{NP}\), we have:

\[ \forall A \in \{\text{formulaSAT, CLIQUE, INDSET, co-TAUTO}\} : A \leq_p \text{SAT} \quad (1) \]

Any CNF formula is also a Boolean formula (the CNF formulas are a special case of the Boolean formulas). Thus \(\text{SAT}\) can be reduced to \(\text{formulaSAT}\) using the following reduction: \(f(\varphi) := \varphi\) if \(\varphi\) is a CNF formula\(^2\) and \(f(x) := B_0\) if \(x\) is not a syntactically valid CNF formula. Here \(B_0\) is a syntactically invalid Boolean formula. Obviously \(f(x) \in \text{formulaSAT}\) iff \(x \in \text{SAT}\). \(f\) is polynomial-time computable. Hence:

\[ \text{SAT} \leq_p \text{formulaSAT} \quad (2) \]

\(^1\)Certificates are: satisfying assignment for \(\text{formulaSAT}\), a clique for \(\text{CLIQUE}\), an independent set for \(\text{INDSET}\), and an assignment that makes \(f\) false for \(\text{co-TAUTO}\).

\(^2\)If the encoding of CNF-formulas is different, then \(f(\varphi)\) should be the re-encoding of \(\varphi\) as a Boolean formula.
A Boolean formula $B$ is satisfiable iff $B$ is not always false iff $\neg B$ is not always true iff $\neg B$ is not a tautology. Thus with $f(B) := \neg B$ (and $f(x) := x$ for syntactically invalid formulas $x$), we have $B \in \text{formulaSAT}$ iff $f(B) \in \text{co-TAUTO}$. $f$ is polynomial-time computable. Hence:

$$\text{formulaSAT} \leq_p \text{co-TAUTO} \quad (3)$$

The following was shown in the practice session:

$$\text{SAT} \leq_p \text{INDSET} \quad (4)$$

A set $S$ is an independent set in a graph $G$ iff $S$ is a clique in the graph $\tilde{G}$. Here $\tilde{G}$ denotes the graph that has an edge between $v, w$ iff $G$ does not have an edge between $v, w$. Let $f(G, k) := (\tilde{G}, k)$. (And $f(x) = x$ for syntactically invalid inputs.) Then $x \in \text{INDSET}$ iff $f(x) \in \text{CLIQUE}$. $f$ is polynomial-time computable. Hence:

$$\text{INDSET} \leq_p \text{CLIQUE} \quad (5)$$

The following graph summarizes all those reductions. $\xrightarrow{A} \xrightarrow{B}$ stands for $A \leq_p B$.

\begin{center}
\begin{tikzpicture}
  \node[shape=circle,draw=black] (A) at (0,0) {SAT};
  \node[shape=circle,draw=black] (B) at (2,0) {CLIQUE};
  \node[shape=circle,draw=black] (C) at (0,-2) {INDSET};
  \node[shape=circle,draw=black] (D) at (2,-2) {co-TAUTO};
  \node[shape=circle,draw=black] (E) at (0,-4) {formulaSAT};

  \draw[->] (A) -- (B);
  \draw[->] (A) -- (C);
  \draw[->] (A) -- (D);
  \draw[->] (A) -- (E);
  \draw[->] (B) -- (C);
  \draw[->] (B) -- (D);
  \draw[->] (C) -- (D);
  \draw[->] (C) -- (E);
  \draw[->] (D) -- (E);
\end{tikzpicture}
\end{center}

We can see that from any node to any other node, there is a chain of reductions. Since $A \leq_p B \leq_p C$ implies $A \leq_p C$ (this case be easily seen from the definition of Karp reductions), it follows that each language in the graph has a reduction to each other language.

In particular, all those languages are NP-complete, because they are in NP and all can be reduced to SAT.

Problem 2: NP with unbounded certificates

An important condition in the definition of NP is that the certificate has polynomially-bounded length. Without that condition, the definition would look as follows (the following is not an established class!):
**Definition 1 (NP with unbounded witnesses)** A language $L$ is in $\text{hugeNP}$ iff there exists a polynomial-time Turing machine $M$ such that for all $x$ it holds that:

\[ x \in L \iff \exists u \in \{0,1\}^\ast : M(x,u) = 1. \tag{6} \]

Note: $u$ is not restricted in its length in this definition.

Show that $\text{HALT} \in \text{hugeNP}$ where $\text{HALT}$ is the Halting Problem.

**Hint:** A certificate for $(m,\alpha) \in \text{HALT}$ could be the string $1^n$ where $n$ is the number of steps that $M_m(\alpha)$ runs. The notation $1^n$ means a bitstring consisting of $n$ ones (e.g., $1^{10} = 1111111111$). So you just need to construct the TM $M$ from **Definition 1** and explain why it is polynomial-time.

**Solution.** Let $M(x,u)$ do the following:

- If $x$ is not a pair $(m,\alpha)$ consisting of a TM description $m$ and a TM input $\alpha$, return 0.
- If $u$ is not a string of the form $1^n$, return 0.
- If $x = (m,\alpha)$ and $u = 1^n$, then run $M_m(\alpha)$ for $n$ steps. If $M_m(\alpha)$ halts within $n$ steps, return 1. Else return 0.

If $(m,\alpha) \notin \text{HALT}$, then $\nexists u \in \{0,1\}^\ast : M(x,u) = 1$, since $M_m(\alpha)$ never stops and hence $M((m,\alpha),u)$ cannot output 1. If $(m,\alpha) \in \text{HALT}$, then with $u := 1^n$ where $n$ is the running time of $M_m(\alpha)$, we have $M(x,u) = 1$. So (6) holds for all $x$.

It remains to show that $M$ is polynomial-time. Checking the first two conditions from the definition of $M$ is obviously in polynomial-time. The time $T$ required to run $M_m(\alpha)$ for $n$ steps is polynomial in $n$ and the sizes of $m,\alpha$. Thus $T \leq p(n + |m| + |\alpha|) = p(|u| + |m| + |\alpha|) = p(|((m,\alpha),u)|)$. Thus the TM $M$ always runs in polynomial-time in the length of its input, as required.

Hence $\text{HALT} \in \text{hugeNP}$.

This shows that the definition of $\text{hugeNP}$ is probably not what we wanted. In fact, with some extra effort we can show that $\text{hugeNP}$ actually coincides with $\text{RE}$, the class of “recursively enumerable” languages.

---

3We call $M$ polynomial-time iff there exists a polynomial-time $p$ such that for all $x,u$, the running time of $M(x,u)$ is bounded by $p(|x| + |u|)$.

4That is, $\text{HALT} = \{(m,\alpha) : M_m(\alpha) \text{ terminates}\}$ where $M_m$ is the TM with description $n$. 

---