Precept Outline
- Search-to-Decision Reductions
- **NP**-Completeness

A. Search-to-Decision Reduction for SAT

In lecture, we defined general (Cook) reductions $X \preceq Y$, allowed to call an algorithm to solve instances of Y a polynomial number of times; but only saw examples of a special case, where the algorithm for Y is called exactly once. That sufficed for reductions from decision to search problems, and from search to search problems.

Your task is to show that we can also reduce search problems to decision problems: construct a (Cook) reduction from the search version of SAT (find a satisfying assignment for the boolean system of equations, or report that none exists) to the decision version of SAT (output "yes" if the system is satisfiable, and output "no" otherwise).

B. NP-completeness of 3SAT

In this problem, we will prove a well-known intractability result: that 3-SATISFIABILITY (3-SAT hereafter) is **NP**-complete. Before we do so, let's see/review some important definitions:

SAT: In the SATISFIABILITY problem we are given a system of m boolean equations in n variables as input. The goal is to determine whether there is an assignment to the variables that satisfies all equations. Here is an example of a SAT instance with m = 3 and $n = 4$:

$$\begin{aligned} x_1 \text{ or } x_2 \text{ or } x_3 &= \text{ true} \\ \neg x_1 \text{ or } \neg x_3 &= \text{ true} \\ x_1 \text{ or } \neg x_2 \text{ or } x_3 \text{ or } x_4 &= \text{ true} \end{aligned}$$

If we set $x_1 =$ false, $x_2 =$ true, $x_3 =$ false, and $x_4 =$ true, then all of the equations are satisfied, so this is a yes (satisfiable) instance of SAT.

3-SAT: In the 3-SAT problem we are given as input a system of m boolean equations, each of which contains exactly 3 literals (a variable or its negation) from n variables. The goal, as before, is to determine

if the system is satisfiable. Note that the example above is **not** a valid instance of 3-SAT since the second equation only has 2 literals and the last one has 4.

To show that 3-SAT is **NP**-complete we need to prove two separate things:

1. 3-SAT is in **NP** (i.e., there is a polynomial-time verification algorithm);
2. All problems in **NP** poly-time reduce to 3-SAT.

Start by showing that 3-SAT is in **NP** by describing a polynomial-time algorithm for verifying a candidate solution.

To show that all problems in **NP** poly-time reduce to 3-SAT, first recall that in lecture we saw that SAT is **NP**-complete, so finding a poly-time reduction from SAT to 3-SAT suffices. Prove (briefly) that every problem X in **NP** poly-time reduces to 3-SAT if such a reduction exists

To prove SAT $\preceq$ 3-SAT, we can construct (in polynomial time) an instance y of 3-SAT from an instance x of SAT such that y is satisfiable if and only if x is satisfiable. Note that x and y may have different sizes and/or different numbers of variables.

Given an instance x of SAT, describe a polynomial-time algorithm that constructs an instance y of 3-SAT with the above property.

Apply the above transformation to the example in the beginning of this section.

C. NP-completeness of Independent Set

In this problem, we will prove a well-known intractability result: that INDEPENDENT-SET (IND-SET hereafter) is **NP**-complete. Recall that, to do so, we need to—besides understanding what the IND-SET problem is—prove two separate things:

1. IND-SET is in **NP** (i.e., there is a polynomial-time algorithm for verifying a candidate solution);
2. Some **NP**-complete problem poly-time reduces to IND-SET.

We will pick SAT as the **NP**-complete problem to reduce from.

An instance of the IND-SET problem has two components: a graph G and a positive integer k. A valid witness to the instance (G, k) is a set S of $\geq k$ vertices such that none of the edges of G have both endpoints in S.

Prove IND-SET is in **NP**.

In order to reduce from SAT to IND-SET, we must construct an instance of IND-SET from an instance of SAT.

If the system of boolean equations has m equations and n variables, set $k = m$ and create a graph G with *one vertex for each appearance of a literal* in an equation.

Place an edge between two vertices if the literals they represent are

1. in the same equation; or

2. negations of each other.

Apply this transformation to the SAT instances below and list all independent sets of size at least k in the resulting graphs.

Instance 1:

$$\begin{aligned} x_1 \text{ or } \neg x_2 \text{ or } \neg x_3 \text{ or } x_4 &= \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } x_4 &= \text{true} \\ x_2 \text{ or } x_3 \text{ or } \neg x_4 &= \text{true} \end{aligned}$$

Instance 2:

$$\begin{aligned} \neg x_1 \text{ or } x_2 &= \text{true} \\ \neg x_2 \text{ or } x_3 &= \text{true} \\ x_1 \text{ or } \neg x_3 &= \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } \neg x_3 &= \text{true} \\ x_1 \text{ or } x_2 \text{ or } x_3 &= \text{true} \end{aligned}$$

We're still missing one step: post-processing the solution of the IND-SET instance to obtain a solution to the SAT instance. Describe an algorithm that does so.

Explain why the reduction is correct; that is, show that it

1. runs in polynomial time;
2. generates an unsatisfiable IND-SET instance when the SAT instance is unsatisfiable; and
3. generates a satisfiable instance when the SAT instance is satisfiable, where the solution obtained from the post-processing step is a satisfying truth assignment.