CS 6743 **Lecture 7** **Fall 2007**

1 Examples of polynomial-time reductions

1.1 Propositional satisfiability problem

One of the most important (classes of) problems in complexity theory is the propositional satisfiability problem. Here, we will define it for CNFs, in the form convenient for NP-completeness reductions.

Definition 1 (Conjunctive normal form). *A propositional formula (that is, a formula where all variables have two values true/false, also written as 1/0) is in CNF if it is a conjunction (AND) of disjunctions (OR) of literals (variables or their negations). A formula is in kCNF, e.g., 3CNF, if the maximal number of variables per disjunction is k. The disjunctions are also called "clauses", so we talk about a formula as a "set of clauses". We use the letter n to be the number of variables (e.g., $x_1 \dots x_n$), and m for the number of clauses.*

Example 2. *The following formula is in 4CNF:* $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4 \vee \neg x_5)$

A *truth assignment* is an assignment of values true/false (1/0) to the variables of a propositional formula. A *satisfying assignment* is a truth assignment that makes the formula true. For example, in the formula above one possible satisfying assignment is $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0$. In this case, the first clause is satisfied (made true) by either of its literals, the second clause by either of x_1 or $\neg x_2$, and the last by $\neg x_5$.

Definition 3.

$$SAT = \{\phi | \phi \textit{ is a propositional formula which has a satisfying assignment}\}$$

Respectively, $3SAT \subset SAT$ *contains only formulae in 3CNF form.*

Example 4. *The following formula is* unsatisfiable*, that is, false for all possible truth assignments:* $(\neg x \vee negy) \wedge x \wedge y$.

1.2 Examples of reductions

Here we will give several examples of reductions. Later, when we show that SAT is NP-complete, these reductions will give us NP-completeness for the corresponding problems.

The problem Independent Set is defined as follows $IndSet = \{< G, k > | \exists S \subseteq V, |S| = k, \forall u, v \in S \neg E(u, v)\}$. That is, there are k vertices in the graph such that neither of them is connected to another (of course, they can be connected to vertices outside the set).

Example 5. *IndSet* $\leq_p$ *Clique. We need polynomial-time computable function* f*, with* $f(< G, k >) =< G', k' >$ *such that* $< G, k >\in Clique \iff < G', k' >\in IndSet$. *Let* f *be such that* $k' = k$ *and* $G' = \bar{G}$*, that is, taking* G *to complement of* G *and leaving* k *the same. This function is computable in polynomial time since all it does is taking a complement of the relation* E. *Now, suppose that there was a clique of size* k *in* G. *Call it* S. *Then the same vertices, in* S*, will be an independent set in* G'. *For the other direction, suppose that* G' *has an independent set of size* k. *Then* G *must have had a clique on the same vertices.*

In general, proving NP-completeness of a language L by reduction consists of the following steps.

1. Show that the language L is in NP
2. Choose an NP-complete L language from which the reduction will go, that is, $L' \leq_p L$.
3. Describe the reduction function and argue that it is computable in polynomial time.
4. Argue that if an instance x was in L', then $f(x) \in L$.
5. Argue that if $f(x) \in L$ then $x \in L'$.

Example 6. *Here we will show that although 3SAT is a subset of SAT, it has the same complexity:* $SAT \leq_p 3SAT$. *We will start with a formula in CNF form and obtain a 3CNF formula. Consider one clause of a formula, for example,* $(\neg x_1 \vee x_3 \vee x_4 \vee \neg x_5)$. *We want to convert this clause into a 3CNF formula. The idea is to introduce a new variable* y *encoding the last two literals of the clause. Then, the formula becomes* $(\neg x_1 \vee x_3 \vee y) \wedge (y \iff (x_4 \vee \neg x_5))$. *Now, opening the* $\iff$ *and bringing it into CNF form, we obtain* $(\neg x_1 \vee x_3 \vee y) \wedge (\neg y \vee x_4 \vee \neg x_5) \wedge (y \vee \neg x_4) \wedge (y \vee x_5)$. *Now apply this transformation to the formula until every clause has at most 3 literals. You can check the equivalence of the two formulae as an exercise. Now it remains to be shown that the transformation can be done in polynomial time. For every extra literal in a clause there needs to be a* y *introduced. Therefore, since there can be as many as* n *literals in a clause, there can be as many as* $n-2$ *such replacements per clause, for* m *clauses, and each introducing 3 new clauses. Therefore, the new formula will be of size at most* $4nm$*, which is definitely a polynomial in the size of the original formula. Moreover, each step takes only constant time. Therefore,* f *is computable in polynomial time.*

Example 7. *We show that* $3SAT \leq IS$.

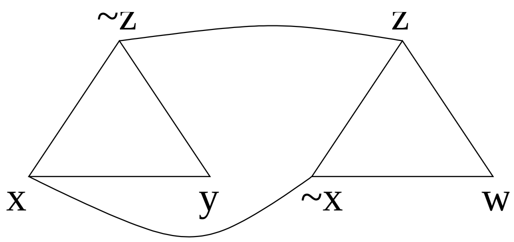
Given

$$\phi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots ()$$

with m *clauses, produce the graph* G_ϕ *that contains a triangle for each clause, with vertices of the triangle labeled by the literals of the clause. Plus, add an edge between any two complementary literals from different triangles. Finally, set* $k = m$.

In our example, we have triangles on $x, y, \neg z$ *and on* $\neg x, w, z$*, plus the edges* $(x, \neg x)$ *and* $(\neg z, z)$ *(see Figure 7).*

Figure 1: Reduction from 3SAT to IS.



It remains to be shown that ϕ is satisfiable iff G_ϕ has an independent set of size at least k.

Proof. We need to prove two directions. First, if ϕ is satisfiable, then G_ϕ has an independent set of size at least k. Secondly, if G_ϕ has an independent set of size at least k, then ϕ is satisfiable. (Note that the latter is the contrapositive of the implication "if ϕ is not satisfiable, then G_ϕ does not have an independent set of size at least k".)

For the first (easy) direction, consider a satisfying assignment for ϕ. Take one true literal from every clause, and put the corresponding graph vertex into a set S. Observe that S is an independent set of size k (where k is the number of clauses in ϕ).

For the other direction, take an independent set S of size k in G_ϕ. Observe that S contains exactly one vertex from each triangle (clause), and that S does not contain any conflicting pair of literals x and $\bar{x}$ (since any such pair of conflicting literals are connected by an edge in G_ϕ). Hence, we can assign the value True to all these literals in the set S, and thereby satisfy the formula ϕ. □

Note that in the proof above, we say things like "take a satisfying assignment to ϕ" or "take an independent set of G_ϕ". These are not efficient steps! But, we don't need them to be efficient, as they are done only as part of the analysis of the reduction (i.e., the proof of correctness of our reduction), and they are not part of the reduction itself. So the point is that the reduction *must be efficient, whereas the* analysis *of correctness of the reduction may involve inefficient (or even non-computable) steps.*