

CMPT 710/407 - Complexity Theory

Lecture 3: Reductions and Time Complexity

Valentine Kabanets

September 11, 2007

1 Reductions

For two languages L_1 and L_2 (over an alphabet Σ), we say that L_1 is *reducible* to L_2 , denoted by $L_1 \leq L_2$, if there is an *efficiently* computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that

$$x \in L_1 \Rightarrow f(x) \in L_2,$$

$$x \notin L_1 \Rightarrow f(x) \notin L_2.$$

For now, “efficiently” means “in polytime”.

Suppose that $L_1 \leq L_2$ (via a reduction f) and that L_2 is decided by some TM M_2 . Then the following is a TM deciding L_1 :

“On input x ,

1. compute $y = f(x)$, and
2. simulate M_2 on y , outputting whatever M_2 outputs.”

Observe that if the reduction f is efficient and if algorithm M_2 is efficient, then the described algorithm for L_1 is also efficient. That is, if L_2 is easy, then so is L_1 . The contrapositive statement is that if L_1 is hard, then so is L_2 .

In other words, we have the following two views of the reduction $L_1 \leq L_2$.

Interpretations of $L_1 \leq L_2$:

- **[algorithm design technique]** L_1 is at most as hard as L_2 : can solve L_1 , given an algorithm for L_2 .
- **[lower bound technique]** L_2 is at least as hard as L_1 : if L_1 is known to be hard, then the reduction $L_1 \leq L_2$ yields that L_2 is also hard.

Example of a reduction:

- $3SAT = \{\phi \mid \phi \text{ is a satisfiable 3CNF formula}\}$ (Recall that a 3CNF is a conjunction of clauses, where each clause is a disjunction of three literals; a literal is a variable or its negation.)
- $IS = \{(G, k) \mid G \text{ is a graph with an independent set of size } \geq k\}$ (Recall that an independent set is a subset of vertices such that no two vertices in the subset are connected by an edge.)

We show that $3SAT \leq IS$.

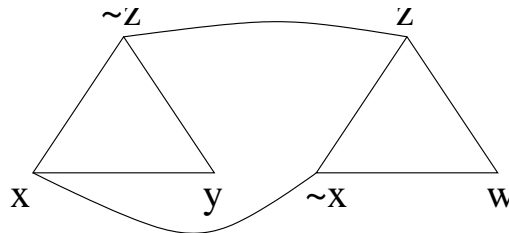
Given

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

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 1).

Figure 1: Reduction from 3SAT to IS.



Theorem 1. 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 . \square

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.

Exercise 2. *Try to show that $IS \leq 3SAT$. (Note that the input graph G can be arbitrary; it is not necessarily a collection of triangles with some extra edges, as was the case in the reduction from $3SAT$ to IS done above.)*

2 Time complexity

Consider some (multi-tape) TM M deciding a language L . For an input x , we define the running time

$$T_M(x) = \text{the number of transitions made by } M \text{ on } x \text{ before halting.}$$

For an input length n , we define the running time of M on inputs of size n as

$$\max_{x:|x|=n} \{T_M(x)\};$$

that is, we take the *worst-case* point of view: bound the running time of M by its running time on the worst possible input of size n .

Finally, for a function $t(n)$ (think of t as a time bound), we say that a language L is decided in time $t(n)$ if there is a TM M deciding L (i.e., M always halts, and M accepts x if and only if $x \in L$) such that, for all sufficiently large n , we have $T_M(n) \leq t(n)$.

Remark 3. *By our definition, a language L is decided in time $t(n)$ if there is a TM M deciding L that asymptotically runs in time at most $t(n)$; that is, there is some natural number n_0 such that for all $n \geq n_0$ the running time of M on inputs of size n is at most $t(n)$. Note that such an algorithm may take arbitrary amount of time on finitely many inputs of size at most n_0 . Still, since the number of such “bad” inputs is finite, we can hard-code into the description of our Turing machine the table of answers for $x \in L$ for all such x of length at most n_0 . This will give us a new Turing machine M' that does the table lookup for inputs x of length at most n_0 , and simulates M on inputs of length greater than n_0 . The running time of the new Turing machine will be at most $t(n)$ on all inputs (assuming that $t(n) \geq n$), since the table lookup can be done in linear time.*

The remark above also suggests that if we have a fast algorithm that is correct on all but finitely many inputs, then we can always “patch it up” by adding to the description of the algorithm the table of correct values on the finitely many “bad” inputs. This will yield a new, efficient algorithm that is correct on *all* inputs.