CS 121: Introduction to Theoretical Computer Science Fall 2019

Section 7

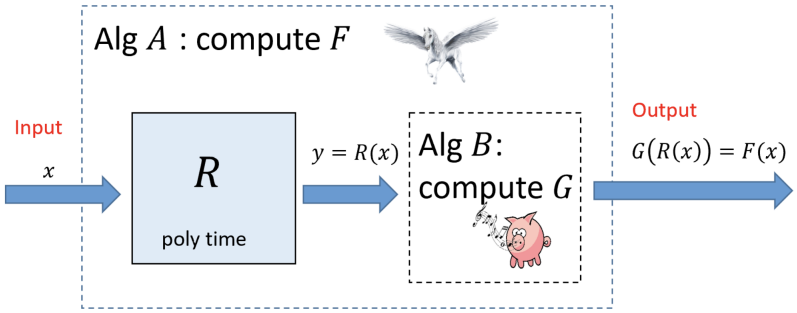
Prof. Boaz Barak

1 Polynomial Time Reductions

Recall how we have been using reductions to prove the uncomputability of functions. We now extend that idea to incorporate notions of time complexity. Polynomial-time reductions are important because they allow us to relate the computational complexity of seemingly unrelated problems (for example, 3SAT and the Longest Path problem seem completely different, but through reductions we can show that they are both computationally "hard" and are in NP).

1.1 Intuition for a Polynomial Time Reduction

If $F \leq_p G$ (i.e. F reduces to G), this means F is "no harder" than G. If we can compute G, we can compute F.



As shown in the above graphic, we can transform a polynomial-time algorithm B that computes G into a polynomial time algorithm A that computes F. To compute F, we transform the input x into F using our polynomial-time reduction to get $R(x)$, and pass that as input into our algorithm that computes G. Because $G(R(x)) = F(x)$, we have a way to compute F if we can compute G.

1.2 Definition

Let $F, G : \{0,1\}^* \to \{0,1\}$. We say that F reduces to G, denoted by $F \leq_p G$ if there is a polynomial-time computable R such that for every $x \in \{0,1\}^*$,

$$F(x) = G(R(x))$$

This is equivalent to saying that F is "easier" than G, as we can efficiently compute F so long as we have a way to efficiently compute G. Remember that "efficient" implies polynomial time.

Exercise: Show that for every $F, G, H : \{0,1\}^* \to \{0,1\}$, if $F \leq_p G$ and $G \leq_p H$, then $F \leq_p H$.

1.3 Example Polynomial Time Reduction

Suppose we want to show that there is no efficient polynomial-time algorithm to compute the *Zero-One Linear Equations Problem.*

To prove this, we can reduce a known NP-Hard problem (i.e. 3SAT) to Zero-One Linear Equations to show $3SAT \leq_p 01EQ$.

If we can show $3SAT$ reduces to $01EQ$, then $3SAT$ is no harder than $01EQ$, and if we can compute $01EQ$ in polynomial time then we can compute $3SAT$. Because our reduction shows that 3SAT is "easier than, or equally as hard as" $01EQ$, if we cannot compute $3SAT$ in polynomial time (and we know there is no such polynomial-time algorithm), we can conclude there exists NO polynomial-time algorithm for $01EQ$.

Understanding the Zero-One Linear Equations Problem
The *Zero-One Linear Equations Problem* corresponds to the function $01EQ : \{0,1\}^* \to \{0,1\}$ where the input is a collection $= E$ of linear equations in variables $x_0, ...x_{n-1}$, and the output is 1 iff $\exists$ assignment $x \in \{0,1\}^n$ of 0/1 values to the variables that satisfies all the equations. So if E encodes the equations $x_0 + x_1 + x_2 = 2, x_0 + x_2 = 1, x_1 + x_2 = 2$ then $01EQ = 1$ because there exists an assignment to satisfy this ($x = 011$).

Reduction from 3SAT to 01EQ
A reduction from F to G should transform the input of F into the input to G in poly-time. We want to define an algorithm R that takes in input x to $3SAT$, and transforms it into input to $01EQ$.
Step 1: Describe an algorithm R for mapping an input φ for 3SAT into an input E for $01EQ$.

Proof idea: A constraint $x_2 \vee \bar{x_5} \vee x_7$ can be rewritten as $x_2 + (1 - x_5) + x_7 \geq 1$.
Making it an equation: Since the sum of the left hand side is an inequality but can be at most 3, we add *auxiliary variables* to make it an equality.
Dealing with negated variables: We also add another variable x_i' to correspond to the negation of x_i and include the equation $x_i + x_i' = 1$.
Thus we have transformed

$$x_2 \vee \bar{x_5} \vee x_7 \Longrightarrow x_2 + x_5' + x_7 + y + z = 3$$

More generically, we transform:

$$x_1 \vee x_2 \vee x_3 \Longrightarrow x_1 + x_2 + x_3 + u + v = 3$$

Formally, the algorithm for this transformation can be described by the pseudocode:

```
Input: CNF input to 3SAT with n variables x_0...x_n-1 and m clauses
Output: set E of linear equations over 0/1
def R(CNF):
    for each of n variables in CNF:
        add equation x_i+x_i'=1 to E representing its negation
    for current clause j in m clauses in CNF:
        let the literals making up the clause be a,b,c
        for each literal in the clause:
```

```
        map the literal to a variable
        (if a is a negation, let its corresponding variable in 01EQ be x_a= a')
        (if a is not negated, let it be x_a = a)
    add the equation x_a + x_b + x_c + y_j + z_j = 3 to E
return E
```

Step 2: Show reduction R runs in polynomial time R clearly runs in polynomial time because it makes an initial loop of n steps, each taking constant time, and another loop of m steps, each step also taking constant time to create our final set of equations. This is poly-time runtime overall.

Step 3a: Show completeness: If $3SAT(\varphi) = 1$ **then** $01EQ(R(\varphi) = 1$

This is the first part of our proof of correctness. Suppose that $3SAT(\varphi) = 1$, then there is an assignment w to satisfy φ. Every clause in φ has form $w_1 \vee w_2 \vee w_3$ so because $w_1 + w_2 + w_3 \geq 1$ we can represent it as $w_1 + w_2 + w_3 + y + z = 3$ where y and z are between 0 and 1. If we let $x'_i = 1 - x_i$ for each variable i, the assignment $x_0 \cdots x_{n-1}, x'_0 \cdots x'_{n-1}, y_0 \cdots y_{m-1}, z_0 \cdots z_{m-1}$ satisfies $E = R(\varphi)$ so $01EQ(R(\varphi)) = 1$.

Step 3b: Show soundness: If $01EQ(R(\varphi) = 1$ then $3SAT(\varphi) = 1$

This is the second part of our proof of correctness. Suppose that $01EQ(R(\varphi) = 1$. Then there must be some assignment $x_0 \cdots x_{n-1}, x'_0 \cdots x'_{n-1}, y_0 \cdots y_{m-1}, z_0 \cdots z_{m-1}$.

Based on the way we did our transformation R, we know that x'_i is the negation of x_i for all $i \in [n]$. Because we defined $y_j, z_j \in [0, 1]$, $y_j + z_j \leq 2$ for all j in $[m]$. Thus, for every clause C_j in φ of the form $w_1 \vee w_2 \vee w_3$, we have $w_1 + w_2 + w_3 \geq 1$. This means the assignment $x_0 \cdots x_{n-1}$ satisfies φ and thus $3SAT(\varphi) = 1$.

This completes our reduction, and proves there is no polynomial-time algorithm to compute $01EQ$. On your homework, it is important you show all 3 bolded steps (including both directions of your proof of correctness!). Besides the practice problems on the next page, it will be helpful to consult the example reductions in Chapter 12 of the book. There are also a lot of example reductions online!

1.4 Practice

Problem 1 The vertex cover of size k over a graph is a set of k vertices such that each edge in the graph has an endpoint in the set. VERTEXCOVER(G, k) returns 1 if there is a vertex cover of size at most k in the graph G, and 0 otherwise. Prove that 3SAT $\leq_p$ VERTEXCOVER.

Problem 2 A "half cover" of size k over a graph is a set of k vertices such that at least half the edges of the graph have an endpoint in the set. HALFCOVER(G, k) returns 1 if there is a half cover of size at most k in the graph G, and 0 otherwise. Prove that VERTEXCOVER $\leq_p$ HALFCOVER.

Problem 3 A 3 coloring of a graph is an assignment of a color $\in \{R, B, G\}$ to every vertex of the graph such that no two vertices connected by an edge are assigned the same color. Prove that 3SAT $\leq_p$ 3COLOR.

Problem 4 Let SINGLE2SAT take in a 2CNF formula, and returns 1 if there is exactly one solution, and 0 otherwise. Similarly, let DOUBLE2SAT take in a 2CNF formula, and returns 1 if there is exactly two solutions, and 0 otherwise. Show that SINGLE2SAT $\leq_p$ DOUBLE2SAT.