

# CIS 511 Homework 5

Stephen Phillips, Dagaen Golomb

April 8, 2015

## Problem 1

The NP-Completeness proof for SUBSET-SUM works because the input is encoded in  $\log(n)$  bits, so solving the problem takes time exponential in the number of bits. Since complexity is measured against the size of the input, this places SUBSET-SUM  $\in$  NP.

However, for UNARY-SSUM, each number occupies length equal to its value. Therefore, we can convert each number into binary which can be done in polynomial time via successive division by two and checking any remainder (which becomes the next significant digit). This runs in polynomial time. We do this for each number. Now, we have the classic SUBSET-SUM problem. We can solve this in a brute-force manner by simulating the non-deterministic TM.

This simulation takes time exponential in the length of the [created] input, which is  $\log(n)$  of the original input. Therefore, this runs in time polynomial ( $O(e^{\log(n)}) = O(n)$ ) of the original input. Thus, UNARY-SSUM  $\in$  P.

## Problem 2

### Part a

We know that  $CNF_2 = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable CNF formula with each variable appears at most 2 times}\}$  is in  $P$  since we can make an algorithm for it.

**function**  $M(\phi)$

Check that all variables  $x_i$  appear at most twice as

Create  $\psi$ , a modifiable copy of  $\phi$

Create bit vector  $b$  for the final assignment

**for all**  $i \in \{1, \dots, n\}$  **do**

**if** There is one clause left **then**

        Accept

**else if** There is one variable left **then**

**if** Clauses  $x_i$  and  $\bar{x}_i$  both are in  $\phi$  **then**

            Reject

**else**

            Accept

**end if**

**else if** the literal  $x_i$  appears once **then**

$\psi \leftarrow \psi(x_i = 1)$  (and simplify  $\psi$ )

**else if** the literal  $\bar{x}_i$  appears once **then**

$\psi \leftarrow \psi(x_i = 0)$

**else if** the literal  $x_i$  appears twice **then**

$\psi \leftarrow \psi(x_i = 1)$

**else if** the literal  $\bar{x}_i$  appears twice **then**

$\psi \leftarrow \psi(x_i = 0)$

**else**

        Combine the clauses with  $x_i$  and  $\bar{x}_i$  in  $\psi$  by creating a new clause with all the other literals in the two clauses together

**end if**  
**end for**  
**end function**

We need to show that this works, so we analyze each case in the if else chain in the loop. We show that each branch either makes the problem smaller, or solves the problem.

- If there is only one clause, we can set any literal in the clause to true and we have satisfied the problem, so we are done.
- If there is one variable  $x$  left, since we know each variable only appears twice, it is unsatisfiable if and only if the clauses left are  $x$  and  $\bar{x}$ , so we accept accordingly.
- If a literal  $x$  only appears once in the formula, we can just set it to true and get rid of the clause that the literal is in. Similarly, if the literal  $\bar{x}$  is in only once, we can just set it to false to get rid of the clause.
- If the literal  $x$  appears twice in the formula (or equivalently  $\bar{x}$ ), then we can like before just set the literal to the appropriate value and get rid of the clauses that it occupies.
- The tricky case - if  $x$  and  $\bar{x}$  both appear in different clauses. Note however that:

$$(x + a)(\bar{x} + b) \iff \bar{x}a + xb + ab \iff a + b$$

Since we can choose  $x$  accordingly. So it is satisfiable if and only if  $a + b$  is true, so we combine the clauses and continue.

Since we reduce the problem by at least one variable each time, with simple boolean formula manipulations in each step, we can run this in polynomial time in the number of clauses and variables. Therefore, this is in  $P$ .

## Part b

We show that  $CNF_3 = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable CNF formula with each variable appears at most 3 times}\}$  is NP-Complete. We show how to change an instance of 3SAT to an instance in  $CNF_3$ .

Given a 3SAT we can increase the number of variables to make sure that no one variable shows more than 3 times. Let us say a variable  $x$  shows up in  $k$  clauses. Then we create new variables  $y_1, \dots, y_k$ . We create a ‘chain’ of implications:

$$y_1 \implies y_2 \implies y_3 \implies \dots \implies y_n \implies y_1$$

This means all these variables must have the same value. In CNF form, using  $x \implies y \iff (\bar{x} + y)$ , this would be

$$(\bar{y}_1 + y_2)(\bar{y}_2 + y_3) \dots (\bar{y}_n + y_1)$$

Then for each clause that  $x$  is in, we use one of the  $y_i$  variables. Now each of the  $y_i$  variables appears twice in the ‘chain’ of implications, and once in the actual clauses, meaning it appears a total of 3 times.

We illustrate with an example, using the simple example with only the variable  $x$ :

$$(x + a_1 + b_1)(x + a_2 + b_2)(\bar{x} + a_2 + b_2)(\bar{x} + a_4 + b_4)$$

This becomes with these new rules:

$$\begin{aligned}
 &(\bar{y}_1 + y_2)(\bar{y}_2 + y_3)(\bar{y}_3 + y_4)(\bar{y}_4 + y_1) \\
 &(y_1 + a_1 + b_1)(y_2 + a_2 + b_2)(\bar{y}_3 + a_2 + b_2)(\bar{y}_4 + a_4 + b_4)
 \end{aligned}$$

Now we need to make sure this is still a polynomial time transformation. Say there are  $n$  variables and  $m$  clauses. In the worse case, a variable appears in every clause. This means we add  $m$  variables to the equation, and  $m$  clauses for the ‘chain’ of implications. If this happens for each variable we get  $mn$  new clauses, and  $mn$  new variables, given  $O(mn)$  sized problem, which is just a quadratic blow up in this size of the formula, and therefore still a polynomial time transformation. Therefore if we can solve  $CNF_3$  we can solve  $3SAT$  which we know to be NP-complete. Since checking a boolean formula is in NP since we can use the assignment as a certificate, this is NP-complete

### Problem 3

We can reduce SAT to the Mine Consistency Problem.

Let us define a variable  $x_{ijk}$  that, for each  $0 \leq k \leq 8$ , represents if the cell at row  $i$  and column  $j$  has value  $k$ . Note in this notation we are using  $k = 0$  to represent the presence of a mine.

The reduction works as follows. For all  $i, j, k$ , we create a conjunction  $(x_{ijk} \cdot x_{\bar{i}\bar{j}k'})$  for each  $k' \neq k$ . This enforces that each cell cannot be assigned two different values. We also produce the disjunction  $(x_{ij0} + x_{ij1} + \dots + x_{ij8})$  for each  $i, j$  pair. This means each cell must be assigned some value. These expressions in total enforce each cell to have exactly one value.

Next we will enforce the rules of Minesweeper. For each  $x_{ijk}$  we enumerate all positional combinations of  $k$  mines around the number and build expressions to match them. Note for  $x_{ij0}$  we will not place any restrictions since the expressions for the numbered cells will enforce the game rules.

### Problem 4

If  $P = NP$ , then there exists a polynomial time decider  $D$  for SAT runs in polynomial time. From this we can create a new machine  $M$  that finds the assignment of the variables. The idea is that we assign true or false to each of the variables in order, and use  $D$  to determine if it is a valid assignment. If there are multiple assignments we just pick the one that works in order of the variables.

```

function  $M(\phi)$ 
  Create  $\psi$ , a modifiable copy of  $\phi$ 
  Create bit vector  $b$  for the final assignment
  for all  $i \in \{1, \dots, n\}$  do
    Set  $\psi' \leftarrow \psi(x_i = 1)$ 
    if  $D(\psi')$  accepts then
       $\psi \leftarrow \psi'$ 
       $b_i \leftarrow 1$ 
    else
       $\psi \leftarrow \psi(x_i = 0)$ 
       $b_i \leftarrow 0$ 
    end if
  end for
  Output  $b$ 
end function

```

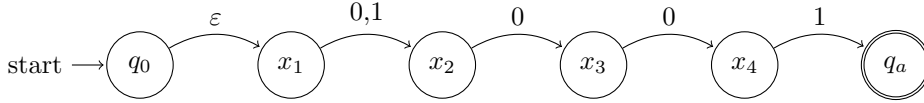
In words, we greedily go through the variables assigning them. We initially assign each variable  $x_i$  to 1, and use  $D$  to check if that assignment is feasible. If it is, then there must be a valid assignment where  $x_i$  is 1, and if not  $x_i$  must be 0. So we assign  $x_i$  appropriately. Then we update the formula to reflect this choice, and continue appropriately.

Now we show this runs in polynomial time. Let  $n$  be the number of variables, and  $m$  be the number of clauses in the original formula. On each iteration of the outer loop, we update the formula, which takes  $O(m)$  time, and we run the decider  $D$ , which runs polynomial time for some polynomial  $p(n + m)$ . There are  $n$  iterations of the loop, so in total this takes  $O(nmp(n + m))$  time, which is still a polynomial.

## Problem 5

## Problem 6

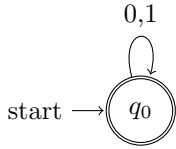
We want to, given a CNF formula  $\phi$ , construct an NFA that accepts all nonaccepting configurations of  $\phi$ . The idea is we create a sub-NFA for each clause that accepts if and only if the clause is false. To illustrate, we use a simple formula over 4 variables  $\phi(x_1, x_2, x_3, x_4) = (x_2 + x_3 + \bar{x}_4)$



This takes in as input the bit string representing the assignment of all the variables. Each of the successive nodes represents the variables, and they transition to the next node on values that would allow the clause to reject. Notice that since  $x_1$  does not appear in the clause, it can take whatever value it wants. So this accepts the non-satisfying strings 0, 1001.

We can generalize this to more clauses, by making the initial state have  $\varepsilon$ -transitions to a chain for each clause, constructed in the exact same way as the above example. That would give, with  $m$  variables and  $c$  clauses,  $cm$  nodes. The transition function would take then  $O(cm)$  space to create. This means it can be created in  $O(cm)$ , or polynomial, time. This new NFA would accept on any rejecting input since it only takes one failing clause to make the formula reject.

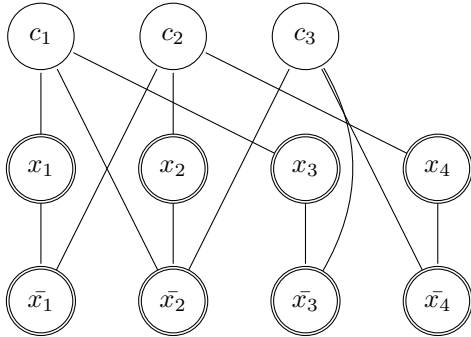
If this formula has no accepting input, this NFA would reduce to one that accepts  $\{0, 1\}^*$ , or:



Therefore if we could reduce this NFA in polynomial, we would be able to solve SAT in polynomial time. In other words we just reduced SAT to minimizing an NFA. Therefore if  $P \neq NP$  then there is no polynomial time solution for this.

## Problem 7

We want to show that the problem of designating directions to edges of an undirected graph  $G$  so that a given subset  $C$  will have all nodes in it have either indegree 0 or outdegree 0, and all other nodes have indegree at least one is NP-complete. We reduce SAT to this. And example of the mapping is show for the formula  $\phi(x_1, x_2, x_3, x_4) = (x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_4)(\bar{x}_2 + \bar{x}_3 + \bar{x}_4)$



The highlighted nodes are the ones in  $C$ .

We want to create such a graph for any SAT formula  $\phi$ . So in words the procedure is for every variable  $x_i$ , create nodes  $x_i$  and  $\bar{x}_i$ , and for every clause  $c_j$  create a node  $c_j$ . Connect the node  $x_i$  (or  $\bar{x}_i$ ) to node  $c_j$  if the variable  $x_i$  (or  $\bar{x}_i$ ) is in the clause  $c_j$ . Also create an edge between nodes  $x_i$  and  $\bar{x}_i$ . Place all the nodes  $x_i$  and  $\bar{x}_i$  into  $C$ .

This graph has a solution the the direction problem if and only if the formula  $\phi$  has a solution. If the formula  $\phi$  has a solution, select the true literals  $x_i/\bar{x}_i$  as the 'all outdegree' or 0 indegree nodes in  $C$ , and the complement pair as the 'all indegree' or 0 outdegree ones. Since only the pairs of literal nodes  $x_i/\bar{x}_i$  are

connected in  $C$ , this does not lead to an inconsistency. If this is the satisfying assignment then we should have at least one edge into each clause node  $c_j$ . Similarly, if we have such an assignment of edge directions, we can reconstruct the satisfying assignment, since only one in each of the pairs of literal nodes  $x_i/\bar{x}_i$  has ‘all outdegree’ we can set those literals to true, and we know by the direction assignment that every clause will have one corresponding true literal inside it. Therefore, this graph has a solution to the problem if and only if the formula has a satisfying assignment.