# CIS 511 Homework 6

Stephen Phillips, Dagaen Golomb

April 8, 2015

## Problem 1

Let $TQBF_{CNF} = \{\phi \mid \phi \text{ is a TQBF with the part after the quantifiers being in CNF}\}$. We show that $TQBF_{CNF}$ is PSPACE-Complete by reducing $TQBF$ to it in polynomial time.

So we just need to show that any boolean formula can be converted to an equivalent CNF formula in polynomial time. We showed something similar to this in class in our reduction from SAT to 3SAT. Here we show this again.

Consider the formula like logical a gate network (unable to show diagram), with the operators acting as two input gates. We can attach intermediate variables to the wires of the gates to change the formula into CNF form. We can consider the network to only have AND, OR and NOT gates without loss of generality. We now find the formulas that convert the gates into an equivalent CNF form, with existance qualifiers:

- AND gate:

$$\begin{aligned} xy &\iff \exists z : (z \implies (xy))(\bar{z} \implies \overline{(xy)}) && \text{Equivalent definitions} \\ &\iff \exists z : (\bar{z} + (xy))(z + \bar{x} + \bar{y}) && \text{DeMorgan's law and Definition of Implication} \\ &\iff \exists z : (\bar{z} + x)(\bar{z} + y)(z + \bar{x} + \bar{y}) && \text{Distributive law} \end{aligned}$$

- OR gate:

$$\begin{aligned} x + y &\iff \exists z : (z \implies (x + y))(\bar{z} \implies \overline{(x + y)}) && \text{Equivalent definitions} \\ &\iff \exists z : (\bar{z} + x + y)(z + \bar{x}\bar{y}) && \text{DeMorgan's law and Definition of Implication} \\ &\iff \exists z : (\bar{z} + x + y)(z + \bar{x})(z + \bar{y}) && \text{Distributive law} \end{aligned}$$

- NOT gate:

$$\begin{aligned} \bar{x} &\iff \exists z : (z \implies x)(\bar{z} \implies \bar{x}) && \text{Equivalent definitions} \\ &\iff \exists z : (\bar{z} + x)(z + \bar{x}) && \text{DeMorgan's law and Definition of Implication} \end{aligned}$$

Thus for each gate we add in at most 3 more variables. If there are $n$ variables, and $m$ gates, this means the new formula would be of size $O(m + n)$ still. You can also think of this in terms of the original formula instead of the gates, There is a correspondence between gates and the $+$ or $\cdot$ operators, so it would still be the same factor of three blowup.

## Problem 2

We are looking at the game of cat and mouse on a graph. The game is given an undirected graph $G$ and nodes $c$ and $m$, the starting nodes for the cat and mouse respectively, and a special 'hole' node $h$, the cat wants to get to the same position as the mouse, and the mouse wants to get to the whole before that happens. The language is:

$$
\begin{aligned}
HAPPY-CAT = \{\langle G,c,m,h\rangle \mid\ & \\
& G,c,m,h \text{ form a game of cat and mouse and Cat has a winning strategy} \\
& \text{ if it moves first}\}
\end{aligned}
$$

First we get out the obvious case: If the mouse is closer to the hole than the cat is, then the mouse always wins. This holds even when the graph is disconnected if you define the distance between nodes in disconnected components to be infinity. So now we consider the cases where the mouse is further away or equal distance from the hole:

- The mouse and cat are on different connected components, and the mouse and hole are on different connected components. Clearly the game is a draw as no one can further their goals.
- The mouse

## Problem 3

We are considering

$$MIN-FORMULA = \{\phi \mid \phi \text{ has no equivalent formula smaller than it}\}$$

We want to show that if $P = NP$, then $MIN - FORMULA \in P$. We will heavily rely on the assumption that $P = NP$. First it is easy to see that to check that two formulas are not equivalent is in NP: $\overline{EQUIV} = \{\langle\phi,\psi\rangle \mid \phi \text{ is not equivalent to } \psi\}$. The verifier is the set of inputs that makes the two functions differ. Since we assume $P = NP$, we know that its complement $EQUIV$ is in $P$ as well. That means there must be a deterministic Turing machine $D_{EQUIV}$ that decides $EQUIV$ in polynomial time.

It is also easy to see that with $D_{EQUIV}$ it is easy to decide

$$\overline{MIN-FORMULA} = \{\phi \mid \phi \text{ has an equivalent formula smaller than it}\}$$

the complement of what we are looking for. The verifier is the formula that is equivalent and smaller. Note that this only works as a verifier since we assume $D_{EQUIV}$ runs in polynomial time. And since has a verifier it is in $NP$ and by assumption in $P$. So there is a deterministic Turing Machine $M$ that decides $\overline{MIN - FORMULA}$, and hence we can complement its output and get a deterministic Turing Machine that decides $MIN - FORMULA$

In summary since we assume that $P = NP$, we can find the deterministic Turing Machines of the complements of our languages of interest and then complement the output.

## Problem 4

We use the definition of $MIN - FORMULA$ from above.

### 0.0.1 Part a

We want to show that $MIN - FORMULA \in PSPACE$ Here is the algorithm:

**function** $M(\phi)$
  **for all** $\psi$ boolean formulas smaller than $\phi$ **do**
    **for all** Inputs $x_1,\dots,x_n \in \{0,1\}$ **do**
      **if** $\phi(x_1,\dots,x_n) \neq \psi(x_1,\dots,x_n)$ **then**
        Break the loop
      **end if**
    **end for**
    **if** All inputs were the same **then**
      Reject
    **end if**

**end for**
Accept
**end function**

This can be shown to be correct fairly straightforwardly, since it iterates through all formulas smaller than $\phi$, and ensures that none of them are equivalent by checking iterating through all possible inputs.

This takes exponential time, but it only needs to store the boolean formulas and the input values, and a constant amount of space to keep track of the solution whether the loop accepted space which is $O(m+n)$, hence polynomial. Since we only use polynomial space, we are in $PSPACE$.

### 0.0.2 Part b

We will show that the following argument is wrong: $MIN-FORMULA \in coNP$ since if $\phi \notin MIN-FORMULA$ then $\phi$ has a smaller equivalent formula which a NTM can guess. In the previous problem we showed that $MIN-FORMULA \in P$ if $P = NP$. We used the fact that $P = NP$ to construct a polynomial time Turing Machine to check that two formulas are equivalent. However, we do not know such a Turing Machine, so we don't know how to verify in polynomial time if two formulas are equivalent. So even if we were shown a smaller equivalent formula, we wouldn't know how to prove it except by iterating through all possible inputs. So this proof is incomplete unless it described a way to do that.

## Problem 5

Show that the language of properly nested parenthesis and brackets is in $L$ (e.g. ([()()]([])) )

First we make sure the brackets and parenthesis are both balanced with respect to each other. This is easy: keep a count of the number of outstanding brackets (or parentheses). Increment on each opening bracket and decrement on each closing one. If the count ever goes negative, reject. If we reach the end of the string and the count is non-zero, reject. This count number takes up O(log($n$)) space in the worst case (when all characters are opening brackets, for example, the count will be the length of the string which can be represented in log($n$) bits).

Given that the previous two checks succeeded, we must now check that they occur in the correct order, i.e. that the situation given in the example does not occur. Note that we can work on this recursively: we can check that the first bracket set is correct, and assume that the whole string is correct if the string between them is also correct. Therefore, we then check the next inner set of brackets for correctness, and so on.

## Problem 6

Show that $UCYCLE = \{\langle G \rangle \mid G \text{is an undirected graph with a simple cycle}\}$ is in $L$.