1 3-TQBF

Recall the definition of a quantified boolean formula, which takes the form

\[ \psi = Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n) \]

where \( x_1, \ldots, x_n \) are boolean variables, \( Q_1, \ldots, Q_n \) are quantifiers (either \( \exists \) or \( \forall \)), and \( \phi \) is a boolean formula with variables \( x_1, \ldots, x_n \). Now recall the definition of the language TQBF where

\[ \text{TQBF} = \{ \langle \psi \rangle \mid \psi \text{ is a quantified boolean formula that is true} \} \]

We proved that the language TQBF is PSPACE-complete. There is another similar language:

\[ \text{3-TQBF} = \{ \langle \psi \rangle \mid \psi = Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n) \text{ and } \phi \text{ is a 3-CNF} \} \]

This language is also PSPACE-complete. The proof is left as an exercise.

2 FormulaGame

Consider a 2-player game played on a quantified boolean formula \( \psi = Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n) \). In this game, player 1 controls the values of all variables \( x_i \) that are preceded by an \( \exists \) quantifier, and player 2 controls the values of all variables \( x_i \) that are preceded by a \( \forall \) quantifier. The quantifiers and variables are considered in order (\( Q_1 x_1 \) is considered first, and \( Q_n x_n \) is considered last), and for each quantifier/variable pair \( Q_i x_i \), if \( Q_i \) is \( \exists \) then player 1 chooses to set \( x_i \) to either \( \text{true} \) or \( \text{false} \). Otherwise, player 2 chooses the value of \( x_i \). Player 1 wins if the formula \( \phi \) is true after all assignments have been made, and player 2 wins if \( \phi \) is false after all assignments are made. Here is an example of the game:

\[ \psi = \exists x_1 \exists x_2 \forall x_3 \exists x_4 \forall x_5 (\neg x_1 \lor x_2 \lor x_3) \land (x_4 \lor \neg x_5) \]

Since \( x_1 \) and \( x_2 \) are both preceded by a \( \exists \), player 1 will choose values for \( x_1 \) and \( x_2 \). Next, player 2 will choose a value for \( x_3 \), and then player 1 will choose a value for \( x_4 \) and so on. If, for example, both players choose to set all their variables to \( \text{true} \) whenever it is their turn, then we see that the above boolean formula will evaluate to \( \text{true} \). Hence in this case player 1 would win.

We say that player 1 has a \textbf{winning strategy} if no matter how player 2 assigns truth values to his variables, there is always a way for player 1 to assign his own variables so that \( \phi \) comes out \text{true} in the end. We define the language

\[ \text{FormulaGame} = \{ \langle \psi \rangle \mid \text{player 1 has a winning strategy on the TQBF } \psi \} \]

If you stare at this definition for a while, you can see that

\[ \text{FormulaGame} = \text{TQBF} \]

What this means is that \text{FormulaGame} is PSPACE-complete.
3 Generalized Geography

We now consider a somewhat more exciting game, which is a 2-player game played on directed graphs. An instance of this game is a directed graph $G = (V,E)$ and a designated starting vertex $b$. A pebble is placed on the starting vertex, and players 1 and 2 take alternating turns moving the pebble to an unvisited neighboring node that the current node is pointing to. More formally, at each turn, if the current node of the pebble is $c$, the current player must move the pebble to a node $u$ such that $(c,u) \in E$ and $u$ has not been visited before by the pebble. Player 1 gets the first turn, and the player who is first unable to make a move loses. We call this game Generalized Geography, or GG for short.

We say that player 1 has a winning strategy for a game of GG if player 1 can always win the game if he makes the “smart” move. In other words, player 1 has a winning strategy if there is always a place for player 1 to move such that no matter what choices player 2 makes, player 1 can force player 2 to eventually lose. We define that language:

$\text{Generalized Geography} = \text{GG} = \{ \langle G, b \rangle \mid \text{player 1 has a winning strategy on directed graph } G, \text{ starting at } b \}$

**Theorem 1.** $\text{GG}$ is PSPACE-complete.

**Proof.** We first show that GG is in PSPACE.

Consider the following recursive algorithm for solving GG:

$\text{GGSOLVER}(G,b)$:

- Let $N$ be the set of nodes $b$ points to. That is the $N = \{ v \mid (b, v) \in E \}$.
- If $N$ is empty, reject
- Else, for each $v \in N$
  - Let $G'$ be the graph creating by removing the node $b$ and removing all edges with $b$ as an endpoint
  - Run $\text{GGSOLVER}(G',v)$
- If all of the subcalls to GGSOLVER above accept, then reject. If one of the sub calls rejects, then accept.

This algorithm correctly decides GG because it essentially tries all possible ways the game could play out, and checks whether there is a way for player 1 to win no matter what player 2 does. Moreover, the algorithm only takes polynomial space, because the algorithm only computes one recursive call to itself at a time. Since the base case (when $N$ is empty) takes constant space to decide, and each recursive call takes about $O(|V|)$ space to compute the neighbors of $b$, the total amount of space is $O(V^2)$. This is because the depth of recursion is at most $|V|$ since every recursive call considers a graph with one less vertex than the previous call.

Now we show that GG is PSPACE-hard. We do so via a reduction from $\text{FORMULAGAME}$. The reduction takes in an instance of $\text{FORMULAGAME}$ and outputs and instance of GG.
Now for any instance of FORMULAGAME:

\[ \psi = Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n) \]

We can assume without loss of generality that \( \phi \) is a 3-CNF. This is essentially because every boolean function can be represented as a 3-CNF. Moreover, we can also assume without loss of generality that \( Q_1 \) is a \( \exists \), and that \( Q_n \) is a \( \forall \), and that the quantifiers alternate. That is, we can assume that for any \( i \), if \( Q_i = \exists \) then \( Q_{i+1} = \forall \) and if \( Q_i = \forall \) then \( Q_{i+1} = \exists \). This is because given a quantified boolean formula, we can always pad the instance with dummy variables that don’t appear in formula. For example, if we have the quantified boolean formula

\[ \exists x_1 \exists x_2 (x_1 \lor x_2) \]

we can convert this into

\[ \exists x_1 \forall y_1 \exists x_2 \forall y_2 (x_1 \lor x_2 \lor x_2) \]

In this case the quantifiers are alternating, the first quantifier is a \( \exists \), and the last quantifier is a \( \forall \). The dummy variables \( y_i \) don’t affect truth value of the formula since they don’t appear in the formula at all. Now because we assume our instance of FORMULAGAME takes on this form, we can create the following directed graph.
In words, what we do is create this “rhombus” structure for each variable \( x_i \), and link them all together. Then we create a node for each of the \( m \) clauses in the formula. For each literal in a clause, we draw an edge from the node corresponding to the clause, to the node corresponding to each literal. For example, in the diagram above, the clause \( \phi_1 = (x_1 \lor \neg x_2 \lor \neg x_n) \). Note that when GG is played on this graph, the players take turns deciding whether to select place a pebble on a literal or its negation.

Now suppose that \( \langle \psi \rangle \in \text{FORMULAGAME} \). This implies there exists a winning strategy for player 1 in the formula game played on \( \psi \). We show that this implies there is a winning strategy for GG on the graph obtained from the construction. The winning strategy for player 1 is as follows: if the winning strategy for the formula game says to set \( x_i \) to true, then pick the node corresponding to \( \neg x_i \), else, pick the node corresponding to \( x_i \). Note that because the \( n \)th quantifier is a \( \forall \), we know that player 2 selects the clause node. Since the original quantified boolean formula is true, each clause is true. This means that no matter what clause node player 2 picks, there will always be an unvisited node for player 1 to then move to, and player 2 will lose.

Similar logic shows that if there is a winning strategy for player 1 on the graph. This similarly there exists a winning strategy for player 1 in the formula game played on \( \psi \).

Finally, note that this reduction takes polynomial time, as it simply requires creating a rhombus structure for each variable in the formula and a node for each clause in the formula.\[\square\]