

Boston College Computer Science Senior Thesis

USING QBF SOLVERS TO SOLVE GAMES AND PUZZLES

Zhihe Shen

Advisor: Howard Straubing

Abstract

There are multiple types of games, such as board games and card games. Some are multiplayer games and some are single-player games. Many games such as 2-player games are hard to solve because the problem of determining whether a given player has a winning strategy for these games is PSPACE-complete. It is proved that the problem of determining whether a quantified boolean formula is true is also PSPACE-complete. Because of the PSPACE-completeness of TQBF, every problem in PSPACE, in particular these games, can be encoded as an instance of TQBF. Thus, one way to understand the complexity of a game is to encode it as a quantified Boolean formula.

This thesis aims to investigate the computational complexity of different kinds of games. We choose to work on games played between two agents, for example, simple geography games. Because they are in PSPACE, we convert them into non-clausal quantified Boolean formulas based on the rules of each game. By solving those formulas, we can find a winning strategy for either player. One way to solve these formulas is to use a quantified Boolean formula solver (QBF solver). In this paper, we will use GhostQ to solve the non-clausal quantified Boolean formula.

Contents

1. Introduction	4
2. Concept	6
2.1 Boolean formula	6
2.2 Conjunctive Normal Form	6
2.3 Boolean Satisfiability Problem	6
2.4 Quantified Boolean Formulas	7
2.5 Clausal vs. Non-Clausal Quantified Boolean Formulas	8
2.6 PSPACE and PSPACE-complete	8
2.7 True Quantified Boolean Formulas	9
2.8 TQBF is PSPACE-complete	10
2.9 SAT vs. QBF	11
2.10 SAT solvers vs. QBF solvers	12
2.11 A QBF solver: GhostQ	13
3. Solve Simple Geography Games	16
3.1 Introduction	16
3.2 Methodology	16
3.3 Encoding	16
3.4 Implementation	18
3.5 Result	23
3.6 Game Interface	23

1. Introduction

Let us first think about a game like Rush Hour. The goal is to find a sequence of legal moves that would allow a target car to exit the game board. In each round, one of the players can make a legal move based on the rules of the game and state of the board. When the game board is very large, it is hard to solve because we cannot find a way better than a brute-force search of all possible sequences of legal

that the players have chosen to the variables in the formula.

Then we can use a quanti ed Boolean formula solver to complete the process of evaluating quanti ed Boolean formulas. In this paper, we will encode simple geography games as instances of quanti ed Boolean formulas. In section 3.3, we will introduce the symbols we use and the rules we follow to encode these games. So how can we solve these instances of quanti ed Boolean formulas? Given that fairly

$(\text{true} _ \text{true}) \quad \text{true}.$

2.4 Concept of Quantified Boolean Formulas (QBF)

Quantified Boolean formulas (QBF) extend propositional formulas by allowing explicit quantification (\exists ; \forall) over the propositional variables [5].

Syntax:

Boolean formulas together with quantifiers \forall (for all) and \exists (there exists) are called **quantified Boolean formulas**. If all the variables in a formula are within the scope of some quantifier, then the formula is fully quantified.

Semantics:

$\forall x \text{ '}$

means ϕ is true. Then we assume x_1 to be false, so we can let x_2 be true to make the statement ϕ true. Since for all possible values of x_1 , we can find a value of x_2 to make the statement ϕ true. We conclude that the statement is always true.

2.5 Clausal vs. Non-Clausal Quantified Boolean Formulas

A **Clausal quantified Boolean formula** is constructed by one or more quantifiers followed by a Boolean formula in conjunctive normal form.

For example, $\exists x_1 \exists x_2 \forall y_1 \forall y_2 \exists z ((x_1 \wedge x_2) \wedge (y_1 \wedge y_2) \wedge z)$ is a clausal quantified Boolean formula.

However, a **non-clausal quantified Boolean formula** does not have this restriction of order. It can be constructed by one or more quantifiers followed by a propositional expression which is followed by quantifiers followed by another propositional expression and so on.

For example, $\exists x_1 \exists x_2 ((x_1 \wedge x_2) \wedge \forall y_1 \forall y_2 ((y_1 \wedge y_2) \wedge z))$

can loop through all possible moves of the two players. Assume there exists a game tree. For an $n \times n$ game board, we need at most $O(n^2)$ space to store the board, so each level of the recursion stack uses at most $O(n^2)$ space. Then we need to keep track of the moves that have been examined. The height of the recursion stack is less than or equal to the depth of the game tree, which is n^2 . Thus, the algorithm to solve go-moku runs in space $O(n^4)$.

2.8 Sketch of the proof that TQBF is PSPACE-complete

The problem of determining whether a quantified Boolean formula is true is PSPACE-complete. Our approach to prove the statement follows that of Sipser [3]. The proof consists of two parts.

First, we use a recursive algorithm to show that TQBF is in PSPACE. Let T be a polynomial space algorithm that decides TQBF and ϕ - a fully quantified Boolean formula - be the input of the algorithm T :

1. If ϕ contains no quantifiers i.e. \exists, \forall , then we evaluate ϕ directly because the expression only contains constants. If ϕ is true, then accept. Otherwise, reject.
2. If ϕ equals $\exists x \phi'$, then we recursively call T on ϕ' because variable x can have different values. That is to say, we replace variable x with 1 and 0 to evaluate ϕ' .

a quantified Boolean formula that is true if and only if a Turing Machine M accepts the input string w . To get an idea of how to construct ϕ , we first construct a formula $\phi_{c_1;c_2;t}$ where $c_1;c_2$ are two configurations and t is a positive number. We let the formula to be true if and only if M can go from c_1 to c_2 in at most t steps.

If $t = 1$, we can construct $\phi_{c_1;c_2;t}$ such that one of the following two conditions is true: 1. c_1 equals c_2 2. M can go from c_1 to c_2 in one step

If $t > 1$, we construct $\phi_{c_1;c_2;t} = \exists m_1 [\phi_{c_3;c_4;t-2} \wedge f(c_1;m_1);(m_1;c_2)]$, where m_1 is a configuration of M . This formula indicates that the variable representing the configurations c_3, c_4 can take either the values of the variables of c_1 and m_1 or m_1 and c_2 . In either case, the formula $\phi_{c_3;c_4;t-2}$ is true, which means that M can go from c_3 to c_4 in at most $t-2$ steps. To convert the formula $\phi_{c_1;c_2;t}$ into a quantified Boolean formula, we replace $\phi_{c_3;c_4;t-2} \wedge f(c_1;m_1);(m_1;c_2)$ by $\phi_{c_3;c_4;t-2} \wedge [(c_3;c_4) = (c_1;m_1) \vee (c_3;c_4) = (m_1;c_2)]$

The formula $\phi_{c_{start};c_{accept};h}$, where $h = 2^{d \cdot f(n)}$, and d is a constant. When $t > 1$, we construct ϕ recursively. The size of each level of recursion is $O(f(n))$, and the number of levels of recursion is also $O(f(n))$. Thus, the formula we get after recursive calls is of size $O(f^2(n))$, which is polynomially large.

2.9 SAT vs. QBF

Boolean Satisfiability Problem (SAT) are hard to solve. It is believed that no algorithm can solve all Boolean Satisfiability Problems efficiently. According to Cook-Levin theorem, the Boolean Satisfiability Problem is NP-complete, which means that any problem in class NP is polynomial time reducible to the Boolean Satisfiability Problem. However, the decision problem of QBF is PSPACE-complete,

as shown in the previous section. Thus, according to the definition of PSPACE-complete, the decision problem of QBF is in PSPACE and is PSPACE hard. Since $NP \neq PSPACE$ and NP is believed to be not equal to PSPACE, we know that PSPACE problems are harder than NP problems. That is to say, the decision problems of QBF are even harder than satisfiability problems.

2.10 QBF solvers vs. SAT solvers

- (1) $y_1 = \text{true}, y_2 = \text{true}$
- (2) $y_1 = \text{true}, y_2 = \text{false}$
- (3) $y_1 = \text{false}, y_2 = \text{true}$
- (4) $y_1 = \text{false}, y_2 = \text{false}$

Since $x_1 = \text{false}, x_2 = \text{true}$, we have $x_1 \wedge x_2 = \text{false}$. Thus, we only need to check if $\neg(y_1 \wedge y_2) \vee (\neg y_1 \wedge y_2) \vee (y_1 \wedge \neg x_1) \vee (y_2 \wedge \neg x_1)$ is true in the 4 cases above when $x_1 = \text{false}, x_2 = \text{true}$. Since $\neg(y_1 \wedge y_2) \vee (\neg y_1 \wedge y_2) \vee (y_1 \wedge \neg x_1) \vee (y_2 \wedge \neg x_1)$ is a disjunction of 4 conjunctions, it would be true if the value of one of its conjunction is true.

- (1) If $y_1 = \text{true}, y_2 = \text{true}$, then $y_1 \wedge y_2 = \text{true}$. Thus, $\neg(y_1 \wedge y_2)$ is true.
- (2) If $y_1 = \text{true}, y_2 = \text{false}$, since x_1 is false, then $y_1 \wedge \neg x_1 = \text{true}$. Thus, $(y_1 \wedge \neg x_1)$ is true.
- (3) If $y_1 = \text{false}, y_2 = \text{true}$, since x_1 is false, then $y_2 \wedge \neg x_1 = \text{true}$. Thus, $(y_2 \wedge \neg x_1)$ is true.
- (4) If $y_1 = \text{false}, y_2 = \text{false}$, then $\neg(y_1 \wedge y_2) = \text{true}$. Thus, $\neg(y_1 \wedge y_2)$ is true.

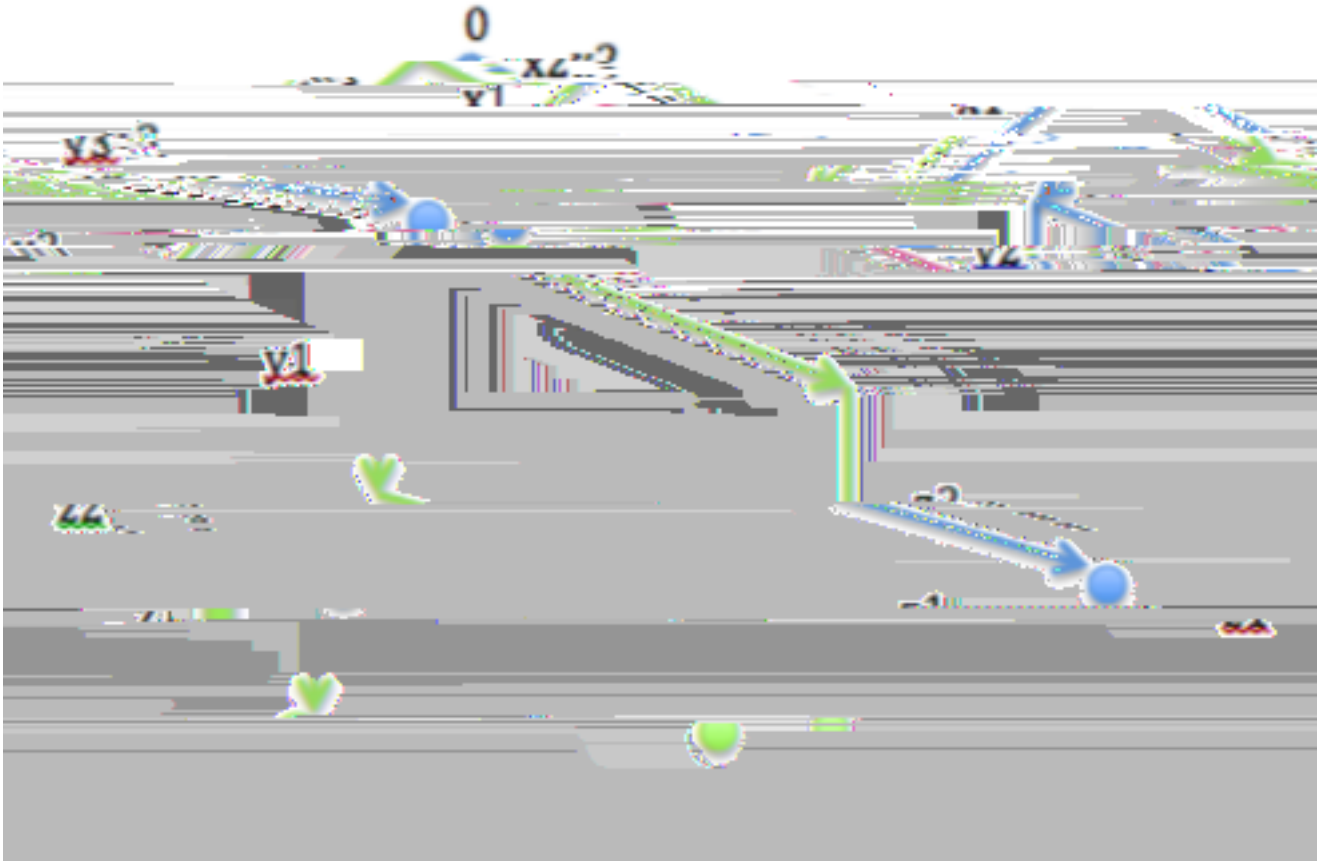
Thus, $x_1 = \text{false}, x_2 = \text{true}$ is a solution to the formula $\neg(y_1 \wedge y_2) \vee (\neg y_1 \wedge y_2) \vee (y_1 \wedge \neg x_1) \vee (y_2 \wedge \neg x_1)$.

3. Solve simple geography games

3.1 Introduction

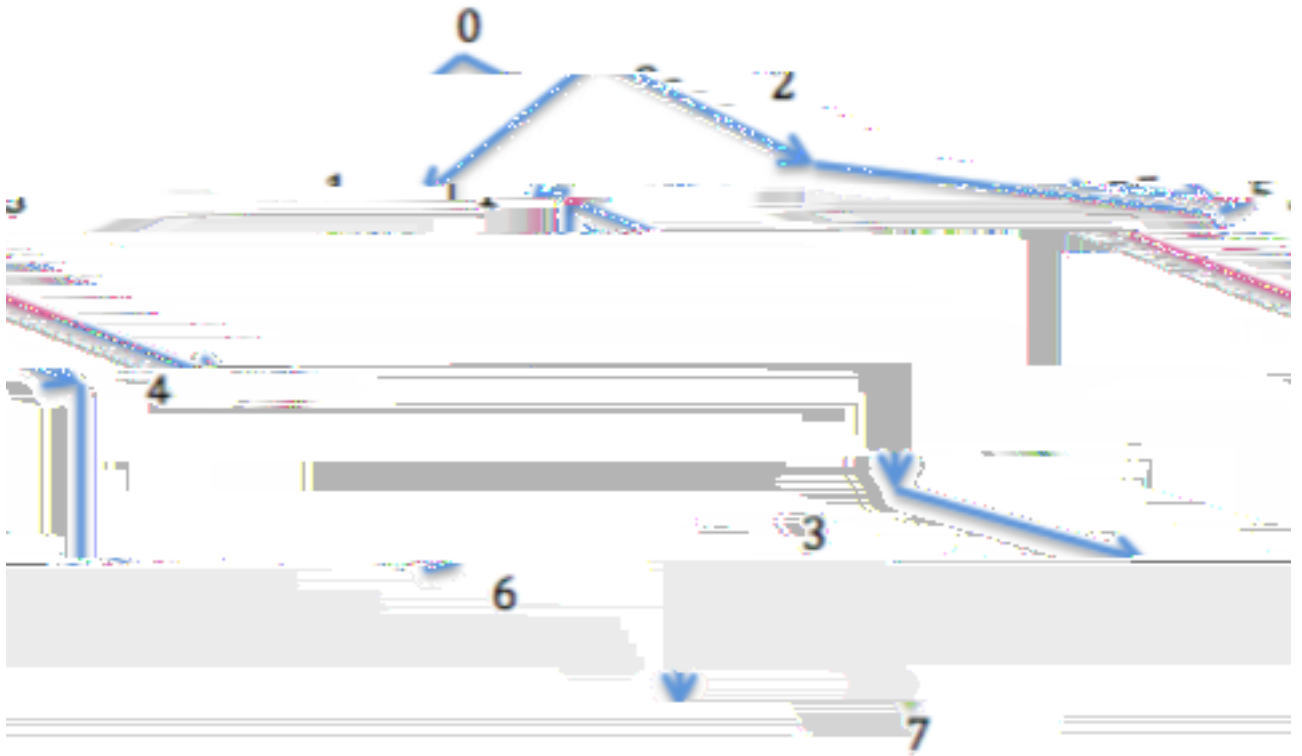
In a geography game, two players take turns to name cities from all over the

For example, the following is a directed graph with vertex 0 as the starting node.



The graph has starting node 0 and three ending nodes. The maximum number of rounds the game can last is 3. We use $x; y; z$ to denote possible moves at the first, second and third round of the geography game. Let 1,2,3 be the index of possible ways of move at each round. Let's assume player 1 plays the first step of the game. The purpose of the game is to decide whether the first player has a winning strategy. If he has a strategy to win, then we need to find the winning strategy. For geography games, a winning strategy for player 1 means that player 1 can successfully reply to all of player 2's replies. From the graph,

we could see that at the first round, player 1 has 2 possible moves: x_1 or x_2 .



The input .txt file for our Python program looks like:

```
3
0 1 2
1 0 3 4
2 0 5
3 1 6
4 1 7
5 2
6 3
7 4
```

In the first row, 3 is the maximum number of rounds the game (represented by the above directed graph) can last. The second row means that from node 0, players can go to node 1 or node 2. The rest rows in the .txt file can be interpreted in similar way.

$$(X_{3_1} - X_{3_2} - X_{3_3} - X_{3_4} - X_{3_5} - X_{3_6} - X_{3_7})$$

Condition 2 can be encoded as the following:

$$\begin{aligned} & (: X_{1_1} - X_{1_2}) \wedge (: X_{1_1} - X_{1_3}) \wedge (: X_{1_1} - X_{1_4}) \wedge (: X_{1_1} - X_{1_5}) \wedge (: X_{1_1} - \\ & : X_{1_6}) \wedge (: X_{1_1} - X_{1_7}) \wedge (: X_{1_2} - X_{1_3}) \wedge (: X_{1_2} - X_{1_4}) \wedge (: X_{1_2} - X_{1_5}) \wedge \\ & (: X_{1_2} - X_{1_6}) \wedge (: X_{1_2} - X_{1_7}) \wedge (: X_{1_3} - X_{1_4}) \wedge (: X_{1_3} - X_{1_5}) \wedge (: X_{1_3} - X_{1_6}) \wedge \\ & (: X_{1_3} - X_{1_7}) \wedge (: X_{1_4} - X_{1_5}) \wedge (: X_{1_4} - X_{1_6}) \wedge (: X_{1_4} - X_{1_7}) \wedge (: X_{1_5} - X_{1_6}) \wedge \\ & (: X_{1_5} - X_{1_7}) \wedge (: X_{1_6} - X_{1_7}) \end{aligned}$$

b. The node visited at a certain level was not visited at all previous levels

This constraint can be encoded as the following:

$$\begin{aligned} & (: X_{1_1} - X_{2_1}) \wedge (: X_{1_2} - X_{2_2}) \wedge (: X_{1_3} - X_{2_3}) \wedge (: X_{1_4} - X_{2_4}) \wedge (: X_{1_5} - \\ & : X_{2_5}) \wedge (: X_{1_6} - X_{2_6}) \wedge (: X_{1_7} - X_{2_7}) \wedge (: X_{1_1} - X_{3_1}) \wedge (: X_{1_2} - X_{3_2}) \wedge \\ & (: X_{1_3} - X_{3_3}) \wedge (: X_{1_4} - X_{3_4}) \wedge (: X_{1_5} - X_{3_5}) \wedge (: X_{1_6} - X_{3_6}) \wedge (: X_{1_7} - X_{3_7}) \\ & (: X_{2_1} - X_{3_1}) \wedge (: X_{2_2} - X_{3_2}) \wedge (: X_{2_3} - X_{3_3}) \wedge (: X_{2_4} - X_{3_4}) \wedge (: X_{2_5} - \\ & : X_{3_5}) \wedge (: X_{2_6} - X_{3_6}) \wedge (: X_{2_7} - X_{3_7}) \end{aligned}$$

c. The node visited at a certain level is adjacent to the node visited at the previous level

This constraint can be encoded as the following:

$$\begin{aligned} & (X_{2_1} \neq (X_{1_3} - X_{1_4})) \wedge (X_{2_2} \neq X_{1_5}) \wedge (X_{2_3} \neq (X_{1_1} - X_{1_2})) \wedge (X_{2_4} \neq (X_{1_1} - \\ & X_{1_7})) \wedge (X_{2_5} \neq X_{1_2}) \wedge (X_{2_6} \neq X_{1_3}) \wedge (X_{2_7} \neq X_{1_4}) \wedge (X_{3_1} \neq (X_{2_3} - X_{2_4})) \wedge \\ & (X_{3_2} \neq X_{2_5}) \wedge (X_{3_3} \neq (X_{2_1} - X_{2_6})) \wedge (X_{3_4} \neq (X_{2_1} - X_{2_7})) \wedge (X_{3_5} \neq \\ & (X_{2_1} - X_{2_7})) \wedge (X_{3_6} \neq (X_{2_1} - X_{2_7})) \wedge (X_{3_7} \neq (X_{2_1} - X_{2_7})) \end{aligned}$$

3.5 Result

a. Output

We run GhostQ with the QCIR file we have generated for the directed graph on page 19 and get a cqbf file which contains the following:

```
Seed: 1. true. Bt: 1, D: 5. R: 0, P: 378, w: 448, C: 0, T: 0.000 s. true()
```

Interpretation: The first "true" after "Seed" means that there exists a winning strategy for player 1 for the geography game.

To find out the winning strategy, we run GhostQ with the cqbf file to generate the file which contains the strategy:

```
list(list(x1_1, true()), list(x1_2, false()), list(x1_3, false()),  
list(x1_4, false()), list(x1_5, false()), list(x1_6, false()),  
list(x1_7, false()), list(x3_1, false()), list(x3_2, false()),  
list(x3_3), list(x3_4, false()), list(x3_5, false()),  
list(x3_6, ite(x2_4, false(), true()))),  
list(x3_7, ite(x2_4, true(), false()))
```

b. Interpretation

According to the output above, the winning strategy for player 1 is the following:

$x_{1_1}, \text{true}()$ means that player 1 should go to node 1 at the first round.

$\text{list}(x_{3_6}, \text{ite}(x_{2_4}, \text{false}(), \text{true}()))$ means that if player 2 does not go to node 4 at the second round, then player 1 should go to node 6 at the second round. This is the same as if player 1 goes to node 3 at the second round, then player 1 should go to node 6 at the third round because at the second round, player 2 can only go to node 3 if he does not go to node 4 given that player 1 goes to node 1 at the first

round.

`list(x3_7, ite(x2_4, true(), false()))` means that if player 2 goes to node 4 at the second round, then player 1 should go to node 6 at the second round.

geography game can last increases to around 10, GhostQ starts to run slowly with our QCIR input. When the QCIR file becomes too large, GhostQ may fail due to the lack of stack space.

3.6 Game Interface

4. References

- [1] Gary William Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895-911, January 2002.
- [2] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125-129. IEEE, New York, 1972.
- [3] Michael Sipser, *Introduction to the Theory of Computation*, Second Edition, Thomson Course Technology, Boston, 2006.
- [4] Frank van Harmelen, Vladimir Lifschitz, Bruce Porter, *Handbook of Knowledge Representation*, Elsevier Science, San Diego, 2007.
- [5] Kleine Buning, H., and Bubeck, U. 2009. Theory of quantified Boolean formulas. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 735-760.
- [6] Non-CNF QBF Solving with QCIR. Charles Jordan, Will Klieber, and Martina Seidl. In *Beyond NP 2016*.
- [7] QCIR-G14: A Non-Prenex Non-CNF Format for Quantified Boolean Formulas, QBF Gallery 2014
- [8] *Formal Verification Using Quantified Boolean Formulas (QBF)*, William Klieber, 2014