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 quanti ed 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 quanti ed Boolean formula.

This thesis aims to investigate the computational complexity of di erent 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 quanti ed Boolean formulas based on the rules of each game. By solving those formulas, we can nd a winning strategy for either player. One way to solve these formulas is to use a quanti ed Boolean formula solver (QBF solver). In this paper, we will use GhostQ to solve the non-clausal quanti ed Boolean formula.

Contents

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

(true _ true) true.

2.4 Concept of Quanti ed Boolean Formulas (QBF)

Quanti ed Boolean formulas (QBF) extend propositional formulas by allowing explicit quanti cation (*9;8*) over the propositional variables [5].

Syntax:

Boolean formulas together with quanti ers *8* (for all) and *9* (there exists) are called quanti ed Boolean formulas. If all the variables in a formula are within the scope of some quanti er, then the formula is fully quanti ed.

Semantics:

8x '

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

2.5 Clausal vs. Non-Clausal Quanti ed Boolean Formulas

A **Clausal quanti ed Boolean formula** is constructed by one or more quantiers followed by a Boolean formula in conjunctive normal form.

For example, $9x_1 9x_2 8y_1 8y_2 9z3\ ((:x_1 \_ x_2) \wedge (y_2 \_ y_3) \wedge z3)$ is a clausal quanti ed Boolean formula.

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

For example, $9x_1 9x_2\ ((x_1 \_ x_2) \wedge 8y_1 8y$

can loop through all possible moves of the two players. Assume there exists a game tree. For an n 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 quanti ed 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 $h\ i$ - a fully quanti ed Boolean formula - be the input of the algorithm T:

1. If contains no quanti ers i.e. 8, 9, then we evaluate directly because the expression only contains constants. If is true, then accept. Otherwise, reject.
2. If equals $8x\ '$, then we recursively call T on $'$ because variable x can have di erent values. That is to say, we replace variable x with 1 and 0 to evaluate $'$.

a quanti ed 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 , we rst construct a formula $_{c_1;c_2;t}$ where $c_1;c_2$ are two con gurations 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 step.

If t = 1, we can construct $_{c_1;c_2;t}$ 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 $_{c_1;c_2;t} = 9m_1 8(c_3;c_4)\ 2\ f(c_1;m_1);(m_1;c_2)g[\ _{c_3;c_4:t=2}]$, where m_1 is a con guration of M. This formula indicates that the variable representing the con gurations 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 $_{c_3;c_4:t=2}$ is true, which means that M can go from c_3 to c_4 in at most t=2 steps. To convert the formula $_{c_1;c_2;t}$ into a quanti ed Boolean formula, we replace $8(c_3;c_4)\ 2\ f(c_1;m_1);(m_1;c_2)g$ by $8(c_3;c_4)\ [(c_3;c_4) = (c_1;m_1)[:::] \_ (c_3;c_4) = (m_1;c_2)\ !\ :::]$

The formula $_{c_{start};c_{accept};h}$, where $h = 2^{df(n)}$, and d is a constant. When $t > 1$, we construct 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 Satis ability Problem (SAT) are hard to solve. It is believed that no algorithm can solve all Boolean Satis ability Problems e ciently. According to Cook-Levin theorem, the Boolean Satis ability Problem is NP-complete, which means that any problem in class NP is polynomial time reducible to the Boolean Satis ability Problem. However, the decision problem of QBF is PSPACE-complete,

as shown in the previous section. Thus, according to the de nition of PSPACE-complete, the decision problem of QBF is in PSPACE and is PSPACE hard. Since NP 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 satis ability problems.

2.10 QBF solvers vs. SAT solvers

outweighs the disadvantage in practice [6]. Thus, in this paper, we use GhostQ, a QBF solver which accepts non-CNF input, to solve decision problems of QBF.

2.11 A QBF solver: GhostQ

Syntax: The input to the QhostQ solver is a QCIR formula. QCIR formulas are de ned by the BNF grammar below. (The listing of the grammar is reproduced from [7].)

(1) y_1 = true, y_2 = true

(2) y_1 = true, y_2 = false

(3) y_1 = false, y_2 = true

(4) y_1 = false, y_2 = false

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

(1) If y_1 = true, y_2 = true, then $y_1 \wedge y_2$ true. Thus, is true.

(2) If y_1 = true, y_2 = false, since x_1 is false, then $y_1 \wedge \neg x_1$ true. Thus, is true.

(3) If y_1 = false, y_2 = true, since x_1 is false, then $y_2 \wedge \neg x_1$ true. Thus, is true.

(4) If y_1 = false, y_2 = false, then $\neg y_1 \wedge \neg y_2$ true. Thus, is true.

Thus, x_1 = false, x_2 = true is a solution to the formula .

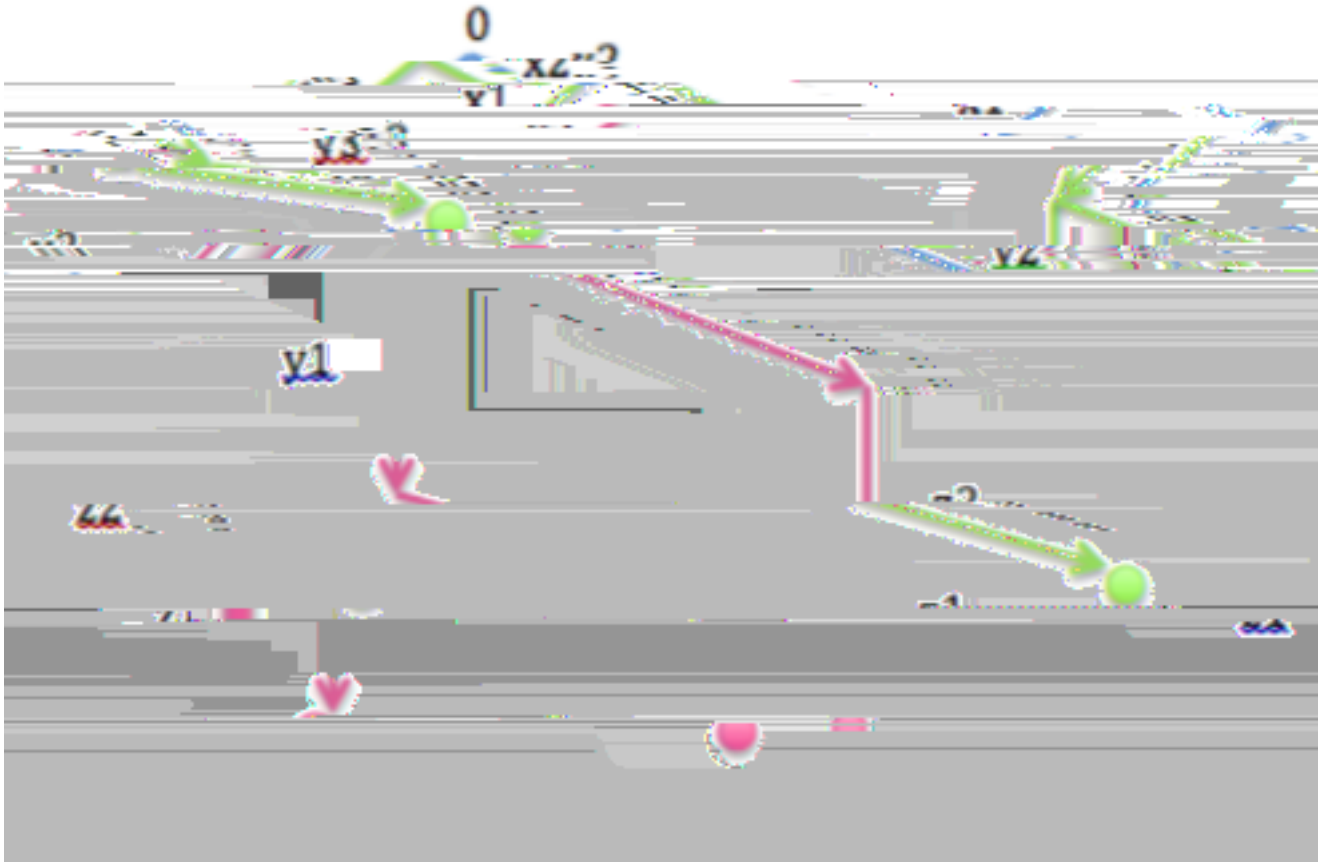
10.21 [y1 (Th1) 1 ([98F488.8 7.9 -/F15 11.955

3. Solve simple geography games

3.1 Introduction

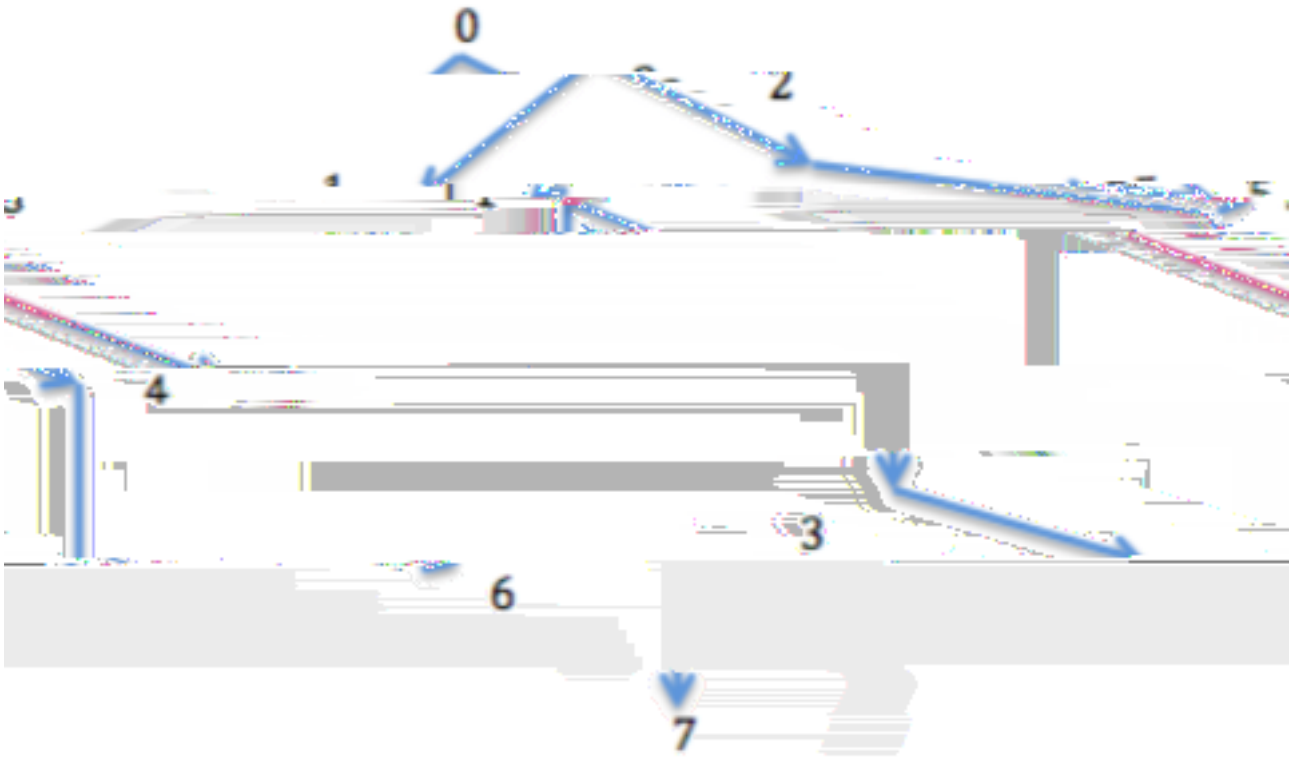
In a geography game, two players take turns to name cities from all over the

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 rst, 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 rst step of the game. The purpose of the game is to decide whether the rst player has a winning strategy. If he has a strategy to win, then we need to nd 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 le 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 rst 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 le 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:

$(x_{1\_1} \_ x_{1\_2}) \wedge (: x_{1\_1} \_ : x_{1\_3}) \wedge (: x_{1\_1} \_ : x_{1\_4}) \wedge (: x_{1\_1} \_ : x_{1\_5}) \wedge (: x_{1\_1} \_ : x_{1\_6}) \wedge (: x_{1\_1} \_ : x_{1\_7}) \wedge (: x_{1\_2} \_ : x_{1\_3}) \wedge (: x_{1\_2} \_ : x_{1\_4}) \wedge (: x_{1\_2} \_ : x_{1\_5}) \wedge (: x_{1\_2} \_ : x_{1\_6}) \wedge (: x_{1\_2} \_ : x_{1\_7}) \wedge (: x_{1\_3} \_ : x_{1\_4}) \wedge (: x_{1\_3} \_ : x_{1\_5}) \wedge (: x_{1\_3} \_ : x_{1\_6}) \wedge (: x_{1\_3} \_ : x_{1\_7}) \wedge (: x_{1\_4} \_ : x_{1\_5}) \wedge (: x_{1\_4} \_ : x_{1\_6}) \wedge (: x_{1\_4} \_ : x_{1\_7}) \wedge (: x_{1\_5} \_ : x_{1\_6}) \wedge (: x_{1\_5} \_ : x_{1\_7}) \wedge (: x_{1\_6} \_ : x_{1\_7})$

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

This constraint can be encoded as the following:

$(: x_{1\_1} \_ : x_{2\_1}) \wedge (: x_{1\_2} \_ : x_{2\_2}) \wedge (: x_{1\_3} \_ : x_{2\_3}) \wedge$ [illegible] $\_ : x_{2\_4}) \wedge (: x_{1\_5} \_ : x_{2\_5}) \wedge (: x_{1\_6} \_ : x_{2\_6}) \wedge (: x_{1\_7} \_ : x_{2\_7}) \wedge (: x_{1\_1} \_$ [illegible] $(: x_{1\_2} \_ : x_{3\_2}) \wedge (: x_{1\_3} \_ : x_{3\_3}) \wedge (: x_{1\_4} \_ : x_{3\_4}) \wedge (: x_{1\_5} \_ : x_{3\_5}) \wedge (: x_{1\_6} \_ :$ [illegible] $: x_{1\_7} \_ : x_{3\_7}) \wedge (: x_{2\_1} \_ : x_{3\_1}) \wedge (: x_{2\_2} \_ : x_{3\_2}) \wedge (: x_{2\_3} \_ : x_{3\_3}) \wedge (: x_{2\_4} \_ : x_{3\_4}$ [illegible] $\_5 \_ : x$ [illegible] $(: x_{2\_6} \_ : x_{3\_6}) \wedge (: x_{2\_7} \_ : x_{3\_7})$

c. The node visited at a certain level is adjacent to the node [illegible] he previous level

This constraint can be encoded as the following:

$(x_{2\_1} \; ! \; (x_{1\_3} \_ x_{1\_4})) \wedge (x_{2\_2} \; ! \; x_{1\_5}) \wedge (x_{2\_3} \; ! \; (x_{1\_1} \_ x$ [illegible] $(x_{2\_4} \; ! \; (x_{1\_1} \_ x_{1\_7})) \wedge (x_{2\_5} \; ! \; x_{1\_2}) \wedge (x_{2\_6} \; ! \; x_{1\_3}) \wedge (x_{2\_7} \; ! \; x_{1\_4}) \wedge$ [illegible] $! \; (x_{2\_3} \_ x_{2\_4})) \wedge (x_{3\_2} \; ! \; x_{2\_5}) \wedge (x_{3\_3} \; ! \; (x_{2\_1} \_ x_{2\_6})) \wedge (x_{3\_4} \; ! \; (x$ [illegible] $\_ x_{2\_7})) \wedge (x_{3\_}$

:

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(x_{1_1}, true()), list(x_{1_2}, false()), list(x_{1_3}, false()),
list(x_{1_4}, false()), list(x_{1_5}, false()), list(x_{1_6}, false()),
list(x_{1_7}, false()), list(x_{3_1}, false()), list(x_{3_2}, false()),
list(x_{3_3}), list(x_{3_4}, false()), list(x_{3_5}, false()),
list(x_{3_6}, ite(x_{2_4}, false(), true())),
list(x_{3_7}, ite(x_{2_4}, true(), false())))

b. Interpretation

According to the output above, the winning strategy for player 1 is the following:
x_{1_1}, true() means that player1 should go to node 1 at the first round.
list(x_{3_6}, ite(x_{2_4}, false(), 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(x_{3_7}, ite(x_{2_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 le 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 quanti ed Boolean formulas. In Handbook of Satis ability, volume 185 of Frontiers in Arti cial 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 Quanti ed Boolean Formulas, QBF Gallery 2014

[8] Formal Veri cation Using Quanti ed Boolean Formulas (QBF), William Klieber, 2014