Encoding Connect-4 using Quantified Boolean Formulae

Ian P Gent and Andrew G D Rowley
University of St. Andrews, Fife, Scotland
{ipg, agdr}@dcs.st-and.ac.uk

Abstract. The game of Connect-4 has been solved, as a win for the first player. Since games are one domain of application of QBF (Quantified Boolean Formulae), it is natural to ask if QBF solvers can prove this result. So, Toby Walsh has proposed solving the game of Connect-4 as a challenge problem for QBF solvers and researchers. In this report we take the zeroth step towards this challenge, by showing how the general game of Connect-4 on a $w \times h$ grid can be encoded as a QBF. We present in detail the sets of QBF clauses used, but do not prove them correct. For the main Connect-4 problem, our encoding contains 18687 variables, 70946 clauses, and 21 alternations of quantifiers. A key part of our encoding are variables capturing the notion of a player “cheating”. A win for the first player is any normal win, or any game in which the second player cheats. Without these, a player could win by, for example, making four moves simultaneously. In terms of the QBF, these are existential variables which are made equivalent to an expression involving universal variables. Our generator and selected instances are being submitted to QBFLib1.

1 Introduction

Quantified Boolean Formulae (QBF) can be seen as games between a player setting values of existential variables, and one setting values of universal variables. The existential player wants to make the underlying SAT formula true, while the universal player wants to make it false. This is not just an analogy. Indeed any QBF instance can be made into a game, although not necessarily an enjoyable one.2 Thus, two player games encoded as QBF instances are a natural domain of application of QBF.

In his invited talk at SAT-03, Toby Walsh suggested that solving the game of Connect-4 would be a productive challenge for QBF researchers [2]. Since it has already been solved using direct techniques (it is a win for the first player), the challenge is not in advancing the understanding of games. Rather, it is in the fact that since we know the problem can be solved exhaustively, it provides a good test case for QBF solvers. If we cannot solve it using reasonably general QBF techniques, that suggests that those techniques are quite limited. On the other hand, if we can solve it, the problem is large enough that it is likely that a number of advances in QBF solving will have been made. Even if it turns out to be easy to solve, there are harder games and other applications of QBF which will benefit from the experience of solving Connect-4.

1 http://www.qbflib.org/
2 A Java implementation by Andrew Pickering makes the Game of QBF playable over the web [1].
Before anyone can solve Connect-4 as a QBF, somebody has to encode it. That is the initial step that we take in this paper.

Our encoding has two aspects that may be of value to others in encoding problems in QBF. First, we introduce “cheat variables”. These solve the problem that the existential and universal players of the game of QBF are not obliged to obey the rules of Connect-4: cheat variables become true if a player cheats, and give the win to the opposite player. Second, we have “gameover” variables, which stop the game continuing when a player wins, and are intended to avoid unnecessary search when a game is won.

2 The game of Connect-c on a w by h board

Connect-4, sometimes known as “Four in a row” is played on a vertical grid of 7 squares across by 6 high, giving a total of 42 squares. There are two players, red and white, who each have 21 pieces. The players take it in turns to place their pieces in one of the 7 columns, where it falls to the bottom-most unoccupied square in the column. Only a total of 6 pieces can be played in any column. The players must play a piece on their turn and must only play one piece. The first player to get 4 pieces in a horizontal, vertical, or diagonal line wins, while if neither player achieves this the game is drawn. The red player goes first.

Connect-4 is available from toyshops under a variety of names, and a number of sites on the web allow online play [3]. It has been solved and it is known that the first player can always win [4].

We consider a slight generalisation of the game, Connect-c, where the players must connect c pieces instead of 4. We also allow variation in the board size of w squares across by h high. We force w ≥ c and h ≥ c to make the game interesting: if either w or h are less than c the game loses a dimension of play since diagonal lines and horizontal or vertical lines respectively become impossible to create. Also, c ≥ 2 must be enforced otherwise the first player wins by playing the first piece. In fact c = 2 is trivial as the first player always wins on the second move; it is observed that Connect-c on a c dimensional board is always trivial. We restrict ourselves to a 2-dimensional board here.

3 Quantified Boolean Formulae

We provide some very brief details of QBF’s, and refer the reader elsewhere for a more detailed introduction [5].

A Quantified Boolean Formula is of the form QB where Q is a sequence of quantifiers Q1v1…Qnvn, where each Qi quantifies (∃ or ∀) a variable vi and each variable occurs exactly once in the sequence Q. B is a Boolean formula in conjunctive normal form (CNF), a conjunction of clauses where each clause is a disjunction of literals. Each literal is a variable and a sign. The literal is said to be negative if negated and positive otherwise. Universals and existentials are those variables quantified by a universal or existential quantifier respectively. A QBF of the form ∃v1Q2v2…QnvnB is true if either Q2v2…QnvnB[v1 := T] or Q2v2…QnvnB[v1 := F] is true, where T represents
true and \( F \) represents false. Similarly, a QBF of the form \( \forall v_1 Q_2 v_2 ... Q_n v_n B \) is true if both \( Q_2 v_2 ... Q_n v_n B[v_1 := T] \) and \( Q_2 v_2 ... Q_n v_n B[v_1 := F] \) are true.

4 Connect-\( c \) as a QBF

In this section we present our encoding of Connect-\( c \) on a \( w \) wide by \( h \) high board. We will encode Connect-4 using a QBF in conjunctive normal form (CNF). QBF is still PSPACE-complete in CNF, and most current solvers work only on CNF instances. Before moving to the excruciating detail, we describe some general principles. These may be useful to those developing encodings for other problems.

Not surprisingly, the key variables we use encode the possible moves for each player. As there up to \( w \) possible moves for each player at each turn, we introduce \( w^2 h \) boolean variables.\(^3\) The variable \( \text{redmove}_{p,z} \) (respectively \( \text{whitemove}_{p,z} \)) is true when red (respectively white) plays a piece in column \( p \) at move \( z \), \( 1 \leq z \leq h \times w \).

Encoding a game into a game-like logical system leads to some possible confusions, certainly enough to confuse us at various points in developing the encoding. The most crucial of these is that neither red nor white, viewed as the existential or universal player, need respect the set of clauses if there is a trivial way to “win”. The red player wins by getting the formula to be true, and the white player by making it false. But many clauses encode rules of the game, not choices of the players. Without detailed care, this leads to the situation where one or other player can win by gratuitous cheating. For example, we encode that the white (universal) player must make a move at its first turn. Naively, this can be done by writing a clause \( \text{whitemove}_{1,2} \lor \ldots \lor \text{whitemove}_{w,2} \). This gives white the winning strategy of cheating by not playing a move, falsifying the clause and thus the whole QBF. A little thought suggests the idea of adding an existential variable \( \text{whitecheat}_{0,2} \), quantified later than the universal variables, to be true if this happens. We declare that red wins if \( \text{whitecheat}_{0,2} \) is true, and revise the clause to \( \text{whitecheat}_{0,2} \lor \text{whitemove}_{1,2} \lor \ldots \lor \text{whitemove}_{w,2} \). Unfortunately, now we have guaranteed the red player an equally fatuous winning strategy. However conscientiously white plays, red declares that white has cheated, setting \( \text{whitecheat}_{0,2} \) true even though (say) \( \text{whitemove}_{3,2} \) is also true. We seem to have a situation where either one player or the other has a trivial winning strategy. Of course neither player is cheating within the rules of QBF, but neither is prepared to accept the rules of Connect-4.

The solution to this problem is to introduce clauses forcing equivalence between the “cheat variables” and their intended meaning. As well as the clause \( \text{whitecheat}_{0,2} \lor \text{whitemove}_{1,2} \lor \ldots \lor \text{whitemove}_{w,2} \), we therefore add \( w \) binary clauses, \( \neg \text{whitecheat}_{0,2} \lor \neg \text{whitemove}_{p,2} \). In combination, these mean that \( \text{whitecheat}_{0,2} \) has the intended meaning. If white tries to set the first clause false by making no move, red declares that white has cheated. But red can only make this claim when white has indeed made no move, or one of the binary clauses will become false. Note that the set of clauses, taken together, encodes \( \text{whitecheat}_{0,2} \equiv \{ \text{whitemove}_{1,2} \lor \ldots \lor \text{whitemove}_{w,2} \} \), but as we are work-

\(^3\) One could use a logarithmic number of variables for each move. Doing so would complicate the encoding, and conventional wisdom in SAT is that logarithmic encodings are often ineffective.
ing in CNF we cannot add this directly and instead encode it by one large clause and many binary clauses.

This pattern is ubiquitous, and absolutely dominates our encoding. In SAT, we are used to encoding some property by simply writing a clause for it. Here, we must be more subtle. Starting from the clause we want, we introduce a “cheat variable” to be true when the clause is false. We guard the original clause with the cheat variable, and then introduce new clauses to ensure that when the cheat variable is true the clause is indeed false.

In many cases, such as the example above, the term “cheat” is fair, while in others we simply need a variable to express some property for reference elsewhere, e.g. that red or white has a line of 4. In general, we therefore use the more neutral term “indicator variables”. But for all indicator variables we see the same pattern: a clause with an indicator variable guarding it, and then many with the negated indicator forcing it to have the intended meaning. Note that indicator variables are all existential, even though many encode properties of universal variables. If they were universal variables, the universal player could win by setting them inconsistently with their intended meaning, thereby falsifying a clause and winning. They are quantified inside all variables they depend on, and as far out as possible given that constraint. That is, an indicator variable is quantified existentially following all the variables involved in the formula it indicates.

One set of indicator variables are true when the game is over at a given move. These play two roles. First, we use them to encode correctly the achievement of a winning line of 4. A winning line only counts if the game is not over yet: if one player has won, then any winning lines achieved later are irrelevant. The gameover variables have another important role. We guard all clauses by stating that they only apply if the game is not over at the relevant move. This counteracts a disadvantage of encoding in QBF: that all later variables have to be set even if a game is already won. By making clauses irrelevant if a game is over, we make sure that any assignment to later variables is valid and yields the same result. We hope this will make search easier.

Throughout our encoding we do not try to formalise deductions such as that there is only one column playable at the last move, or that the game cannot be over after the first $c - 1$ moves by each player. While this would reduce the size of the encoding, it would complicate its presentation and understanding. Furthermore, we suspect that most of these trivial deductions will be made by propagation in search, although it may well be that there is a set of implied clauses that will greatly improve search.

### 4.1 Indicator Variables

We now describe in detail the indicator variables we introduce. As red variables are existential, redcheat variables could be omitted in favour of direct clauses. However, we include them and similar variables, making our encoding pleasingly symmetric in red and white. While leading to slightly more variables, the performance penalty should be minor as the superfluous red indicator variables will be set by propagation.

In the case of cheating and of constructing a line, we number arbitrarily the ways of doing this, introducing a variable for each type of cheating and line.

- **redwin, whitewin and draw** - true iff red, white or nobody (respectively) wins.
– $\text{gameover}_z$ - true if the game is over at move $z$, i.e. the result is determined solely by moves up to $z-1$ or earlier.
– $\text{redwin}_z$ and $\text{whitewin}_z$ - true iff red or white respectively wins at move $z$.
– $\text{redcheat}_z$ and $\text{whitecheat}_z$ - true iff red or white respectively cheats at move $z$. As there are many ways to cheat, we will make $\text{redcheat}$ and $\text{whitecheat}$ equivalent to the disjunction of the following variables.
– $\text{redcheat}_{f,z}$ and $\text{whitecheat}_{f,z}$ - true iff red or white respectively cheat in fashion $f$ at move $z$. The fashions are numbered from 0 to $w(w-1)/2$. Fashion 0 is for playing no counter on a turn, while the other $w(w-1)/2$ enumerate the ways of playing counters in two different columns on the same turn.
– $\text{rline}_{n,z}$ and $\text{whiteline}_{n,z}$ - true iff red or white respectively has a line of $c$ of type $n$ at move $z$.
– $\text{red}_{x,y,z}$ and $\text{white}_{x,y,z}$ - true iff there is a red or white piece (respectively) in row $x$, column $y$ after move $z$ has been made.
– $\text{occupied}_{x,y,z}$ - true iff there is a piece in row $x$, column $y$ before move $z$ has been made ($1 \leq z \leq hw+1$). To simplify our encoding, we include a notional row 0, occupied with uncoloured pieces before move 1.
– $\text{redflip}_{x,y,z}$ and $\text{whiteflip}_{x,y,z}$ - true iff a red or white piece respectively appeared in row $x$ column $y$ at move $z$.

4.2 Quantifiers

We need to quantify the move variables $\text{redmove}$, and $\text{whitemove}$, in the correct way. We would like to know if there is a way for red to win given any move that white makes. We therefore quantify the $\text{redmove}$, variables existentially and the $\text{whitemove}$, variables universally. In addition to this, each player makes their move in turn starting with the red player; we do not want to allow white to play a piece at move $n$ that is independent of move $n-1$. This means that we must alternate the quantification between red’s moves at move $n-1$ and white’s moves at move $n$. The quantifiers of the key variables are therefore as follows:

$$\exists (\text{redmove}_{1,1}, \ldots, \text{redmove}_{w,1})$$
$$\forall (\text{whitemove}_{1,2}, \ldots, \text{whitemove}_{w,2})$$
$$\ldots$$

$$\exists (\text{redmove}_{1,h \times w-1}, \ldots, \text{redmove}_{w,h \times w-1})$$
$$\forall (\text{whitemove}_{1,h \times w}, \ldots, \text{whitemove}_{w,h \times w})$$

In most cases, an indicator variable should be quantified so it can only be set after values have been assigned to the relevant move variables. This is done by inserting an existential quantifier for the indicator variable after the quantifier for the last relevant move variable. The exception are the $\text{redwin}$, $\text{whitewin}$ and $\text{draw}$ variables. These are the to specify the outcome that we require before the game has begun. While their truth value depends on variables quantified later, they must be quantified at the outermost level. They express properties of all assignments to variables, rather than indicating status of a particular partial assignment.

This leaves the quantification scheme as shown in Fig. 1.
Fig. 1. Quantification scheme for encoding Connect-4 on a \( w \times h \) board. Note that the only universally quantified variables are \( \text{whitemove} \). The list of existential indicator variables after move \( z \) are (for all appropriate \( x, y, n \)): \( \text{gameover}_{z+1}, \text{redwin}_z, \text{whitewin}_z, \text{redcheat}_z, \text{whitecheat}_z, \text{redcheat}_{n,z}, \text{whitecheat}_{n,z}, \text{rline}_{n,z}, \text{whiteline}_{n,z}, \text{red}_{x,y,z}, \text{white}_{x,y,z}, \text{occupied}_{x,y,z+1}, \text{redflip}_{x,y,z}, \text{whiteflip}_{x,y,z} \). The scheme shown is where \( h \times w \) is even. If both of \( h \) and \( w \) are odd the last move quantification is existential on \( \text{redmove} \).

### 4.3 Clauses

Our encoding contains 20 sets of clauses. Of course we do not expect a casual reader to study them in detail, but we wish to provide them in detail so that they are documented beyond code in a Perl script, and would encourage other researchers to do the same.

We explain the intention behind each set before detailing the clauses. In many cases there are red- and white-oriented versions of clauses, and we use a horizontal line to separate the red from white clauses. In most cases many versions of clauses are necessary, and we indicate this by one or more conjunctions. As the number of moves is \( hw \), this can be odd or even, and this leads to an asymmetry between red and white, as red makes \( \lceil \frac{1}{2}hw \rceil \) moves while white makes \( \lfloor \frac{1}{2}hw \rfloor \).

\[ \lceil \frac{1}{2}hw \rceil \]

\[ \bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} (\text{gameover}_{2z-1} \lor \text{redcheat}_{0,2z-1} \lor \bigvee_{p=1}^{w} \text{redmove}_{p,2z-1}) \]

\[ \lfloor \frac{1}{2}hw \rfloor \]

\[ \bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \bigwedge_{p=1}^{\lfloor \frac{1}{2}hw \rfloor} (\text{gameover}_{2z-1} \lor \neg\text{redcheat}_{0,2z-1} \lor \neg\text{redmove}_{p,2z-1}) \]

\[ \bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} (\text{gameover}_{2z} \lor \text{whitecheat}_{0,2z} \lor \bigvee_{p=1}^{w} \text{whitemove}_{p,2z}) \]
2. This set of clauses expresses that, unless the game is over, each player must make only one move at their turn or be declared to have cheated. There are \( \frac{1}{2} w(w - 1) \) ways of playing two moves simultaneously. We use the function \( f(w, p, p') \) to yield a unique number between 1 and \( \frac{1}{2} w(w - 1) \) for each possible way of cheating for red or white respectively: this is implemented in code by a simple counter.

\[
\left( \bigwedge_{z=1}^{w} \bigwedge_{p=1}^{p'=p+1} \frac{1}{2} w \bigwedge_{z=1}^{w} \bigwedge_{p=1}^{p'=p+1} \big( \text{gameover}_{2z} \lor \text{whitemove}_{0,2z} \lor \neg \text{whitemove}_{p,2z} \big) \right)
\]

\[
\left( \bigwedge_{z=1}^{w-1} \bigwedge_{p=1}^{p'=p+1} \left( \text{gameover}_{2z-1} \lor \text{redcheat}_f(w, p, p'),2z-1 \lor \neg \text{redmove}_{p,2z-1} \lor \neg \text{redmove}_{p',2z-1} \right) \right)
\]

\[
\left( \bigwedge_{z=1}^{w-1} \bigwedge_{p=1}^{p'=p+1} \left( \text{gameover}_{2z-1} \lor \text{redcheat}_f(w, p, p'),2z-1 \lor \text{redmove}_{p,2z-1} \right) \right)
\]

\[
\left( \bigwedge_{z=1}^{w-1} \bigwedge_{p=1}^{p'=p+1} \left( \text{gameover}_{2z-1} \lor \text{redcheat}_f(w, p, p'),2z-1 \lor \text{redmove}_{p',2z-1} \right) \right)
\]

3. We now specify that a player has cheated at move \( z \) if and only if the player has cheated in some fashion at move \( z \).

\[
\left( \bigwedge_{z=1}^{w} \left( \text{gameover}_{2z-1} \lor \text{redcheat}_{2z-1} \lor \bigvee_{f=0}^{w(w-1)} \text{redcheat}_f,2z-1 \right) \right)
\]

\[
\left( \bigwedge_{z=1}^{w-1} \bigwedge_{f=0}^{w(w-1)} \left( \text{gameover}_{2z-1} \lor \text{redcheat}_{2z-1} \lor \neg \text{redcheat}_f,2z-1 \right) \right)
\]

\[
\left( \bigwedge_{z=1}^{w} \left( \text{gameover}_{2z} \lor \neg \text{whitemove}_{2z} \lor \bigvee_{f=0}^{w(w-1)} \text{whitecheat}_f,2z \right) \right)
\]

\[
\left( \bigwedge_{z=1}^{w-1} \bigwedge_{f=0}^{w(w-1)} \left( \text{gameover}_{2z} \lor \text{whitecheat}_{2z} \lor \neg \text{whitecheat}_f,2z \right) \right)
\]
4. The following clauses specify that only one of a red and white piece can occupy any space at any time.

\[
\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{red}_{x,y,z} \lor \lnot \text{white}_{x,y,z})
\]

5. These clauses express that a red or white respectively in a position at move \( z \) means that there is a piece in the position at move \( z + 1 \). Additionally, a red or white in a position means a red or white respectively in the same position on the next move.

\[
\begin{align*}
&\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{red}_{x,y,z} \lor \text{occupied}_{x,y,z+1}) \\
&\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{white}_{x,y,z} \lor \text{occupied}_{x,y,z+1}) \\
&\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{occupied}_{x,y,z+1} \lor \text{red}_{x,y,z} \lor \text{white}_{x,y,z}) \\
&\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{red}_{x,y,z} \lor \text{red}_{x,y,z+1}) \\
&\bigwedge_{z=1} \bigwedge_{x=1} \bigwedge_{y=1} (\text{gameover}_z \lor \lnot \text{white}_{x,y,z} \lor \text{white}_{x,y,z+1})
\end{align*}
\]

6. If the game is not over at move \( z \) and there is a piece in row \( x \), column \( y \) and no piece above it, then if a red or white is played in column \( y \) then there is now a red or white respectively in the empty square. This is represented by the following clauses.

\[
\begin{align*}
&\left[\frac{h}{h+1} w\right] \bigwedge_{z=1} \bigwedge_{x=0} \bigwedge_{y=1} (\text{gameover}_{z-1} \lor \lnot \text{occupied}_{x,y,2z-1} \lor \text{occupied}_{x+1,y,2z-1} \lor \lnot \text{redmove}_{y,2z-1} \lor \text{redflip}_{x+1,y,2z-1}) \\
&\left[\frac{h}{h+1} w\right] \bigwedge_{z=1} \bigwedge_{x=0} \bigwedge_{y=1} (\text{gameover}_{z-1} \lor \lnot \text{redflip}_{x+1,y,2z-1} \lor \text{occupied}_{x,y,2z-1}) \\
&\left[\frac{h}{h+1} w\right] \bigwedge_{z=1} \bigwedge_{x=0} \bigwedge_{y=1} (\text{gameover}_{z-1} \lor \lnot \text{redflip}_{x+1,y,2z-1} \lor \lnot \text{occupied}_{x+1,y,2z-1}) \\
&\left[\frac{h}{h+1} w\right] \bigwedge_{z=1} \bigwedge_{x=0} \bigwedge_{y=1} (\text{gameover}_{z-1} \lor \lnot \text{redflip}_{x+1,y,2z-1} \lor \text{redmove}_{y,2z-1}) \\
&\left[\frac{h}{h+1} w\right] \bigwedge_{z=1} \bigwedge_{x=0} \bigwedge_{y=1} (\text{gameover}_{z-1} \lor \lnot \text{redflip}_{x+1,y,2z-1} \lor \text{red}_{x+1,y,2z-1})
\end{align*}
\]
7. If there is a red or white in a position at the next move, then there was either one there at this move, or a red or white respectively was added at this move. This is specified as follows.

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{occupied}_{x,y,2z} \lor \text{occupied}_{x+1,y,2z} \lor \\
\neg\text{whitemove}_{y,2z} \lor \text{whiteflip}_{x+1,y,2z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{whiteflip}_{x+1,y,2z} \lor \text{occupied}_{x,y,2z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{whiteflip}_{x+1,y,2z} \lor \neg\text{occupied}_{x+1,y,2z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{whiteflip}_{x+1,y,2z} \lor \text{whitemove}_{y,2z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{whiteflip}_{x+1,y,2z} \lor \text{white}_{x+1,y,2z})
\]

8. We must populate the notional row zero so that the pieces have something to hold them up. The following clauses do this.

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{z} \lor \neg\text{red}_{x,y,z+1} \lor \text{red}_{x,y,z} \lor \text{redflip}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{z} \lor \neg\text{white}_{x,y,z+1} \lor \text{white}_{x,y,z} \lor \text{whiteflip}_{x,y,z})
\]

9. This set of clauses states that if there is a piece in some position in row \( x \) that has a space above it and neither red nor white play in that space then there is no piece in the space on the next move.

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z-1} \lor \neg\text{occupied}_{x,y,2z-1} \lor \text{occupied}_{x+1,y,2z-1} \lor \\
\text{redmove}_{y,2z-1} \lor \neg\text{occupied}_{x+1,y,2z})
\]

\[
\bigwedge_{z=1}^{h-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_{2z} \lor \neg\text{occupied}_{x,y,2z} \lor \text{occupied}_{x+1,y,2z} \lor \\
\text{whitemove}_{y,2z} \lor \neg\text{occupied}_{x+1,y,2z+1})
\]
10. If there is no piece in a position at a move then there is no red and no white in the position above. This is represented as follows.

$$\bigwedge_{z=1}^{h} \bigwedge_{x=1}^{w} \bigwedge_{y=1}^{w} (\text{gameover}_{z} \lor \text{occupied}_{x,y,z} \lor \lnot \text{red}_{x+1,y,z})$$

$$\bigwedge_{z=1}^{h} \bigwedge_{x=1}^{w} \bigwedge_{y=1}^{w} (\text{gameover}_{z} \lor \text{occupied}_{x,y,z} \lor \lnot \text{white}_{x+1,y,z})$$

11. We must disallow the placing of pieces in the row above the top of the board. There is also no piece in the bottom row on the first move. To express this we add the following clauses.

$$\bigwedge_{z=1}^{w} \bigwedge_{y=1}^{w} (\lnot \text{occupied}_{h+1,y,z})$$

$$\bigwedge_{y=1}^{w} (\lnot \text{occupied}_{1,y,1})$$

12. The game over state must be propagated so that if the game is over at a move it is over at the next move as well. If the game is not over at a move, and someone wins at that move, the game is over at the next move. The game is not over on the first move. These constraints are stated as follows.

$$\bigwedge_{z=1}^{w-1} (\lnot \text{gameover}_{z} \lor \text{gameover}_{z+1})$$

$$\bigwedge_{z=1}^{w-1} (\text{gameover}_{z} \lor \lnot \text{redwin}_{z} \lor \text{gameover}_{z+1})$$

$$\bigwedge_{z=1}^{w-1} (\text{gameover}_{z} \lor \lnot \text{whitewin}_{z} \lor \text{gameover}_{z+1})$$

$$\bigwedge_{z=1}^{w-1} (\lnot \text{gameover}_{z+1} \lor \text{gameover}_{z} \lor \text{redwin}_{z} \lor \text{whitewin}_{z})$$

$$\lnot \text{gameover}_{1}$$

$$\lnot \text{redwin}_{1}$$

$$\lnot \text{whitewin}_{1}$$

13. The following clauses represent the facts that if the game is not over at the end then it is a draw, and every game must be either a win for red, a win for white or a draw.

$$(\text{gameover}_{hw+1} \lor \text{draw})$$

$$\lnot \text{draw} \lor \lnot \text{gameover}_{hw+1}$$

$$(\text{redwin} \lor \text{whitewin} \lor \text{draw})$$

$$(\lnot \text{redwin} \lor \lnot \text{whitewin})$$
14. The fact that Red or white wins if red or white respectively wins at any intermediate move is encoded as follows.

\[
\bigwedge_{z=1}^{hw} (\text{gameover}_z \lor \neg \text{redwin}_z \lor \text{redwin})
\]

\[
(\text{redwin} \lor \bigvee_{z=1}^{hw} \text{redwin}_z)
\]

\[
\bigwedge_{z=1}^{hw} (\text{gameover}_z \lor \neg \text{whitewin}_z \lor \text{whitewin})
\]

\[
(\text{whitewin} \lor \bigvee_{z=1}^{hw} \text{whitewin}_z)
\]

15. The next set of clauses encode that red or white has a line if there are \(c\) reds or whites respectively in a line. \(l_v = w(h - c + 1)\) is the number of vertical lines, \(l_h = h(w - c + 1)\) is the number of horizontal lines and \(l_d = (w - c + 1)(h - c + 1)\) is number of diagonal lines in each direction.

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{w-c+1} \bigwedge_{x'=1}^{x' c+1} \bigwedge_{x'=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(x'-1)+y,z} \lor \bigvee_{x=x'}^{x'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{y' c+1} \bigwedge_{x'=1}^{y'} \bigwedge_{x'=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y'=1}^{y' c+1} \bigwedge_{x=1}^{x' c+1} \bigwedge_{x=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{y' c+1} \bigwedge_{x=1}^{x'} \bigwedge_{x=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{y' c+1} \bigwedge_{x=1}^{x'} \bigwedge_{x=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{y' c+1} \bigwedge_{x=1}^{x'} \bigwedge_{x=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]

\[
\bigwedge_{z=1}^{hw} \left( \bigwedge_{y=1}^{y' c+1} \bigwedge_{x=1}^{x'} \bigwedge_{x=1}^{x' + c} \right) (\text{gameover}_z \lor \text{rline}_{w(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{red}_{x,y,z})
\]
\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{h-c+1} (gameover_z \lor \text{rline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \\
&\quad \lor \bigvee_{d=0}^{c-1} \neg \text{red}_{x+d,y+c-1-d,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{h-c+1} (gameover_z \lor \neg \text{rline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \\
&\quad \lor \bigvee_{d=0}^{c-1} \text{red}_{x+d,y+c-1-d,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} \bigwedge_{x'=1}^{x'+c} (gameover_z \lor \text{whiteline}_{w(x'-1)+y,z} \lor \bigvee_{x=x'}^{z'+c} \neg \text{white}_{x,y,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} (gameover_z \lor \neg \text{whiteline}_{w(x'-1)+y,z} \lor \text{white}_{x,y,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} \bigwedge_{y'=1}^{y'+c} (gameover_z \lor \text{whiteline}_{l_y+h(y'-1)+x,z} \lor \bigvee_{y=y'}^{y'+c} \neg \text{white}_{x,y,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} (gameover_z \lor \neg \text{whiteline}_{l_y+h(y'-1)+x,z} \lor \text{white}_{x,y,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} \bigwedge_{x=1}^{c-1} (gameover_z \lor \text{whiteline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \lor \bigvee_{d=0}^{c-1} \neg \text{white}_{x+d,y+d,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} (gameover_z \lor \neg \text{whiteline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \lor \text{white}_{x+d,y+d,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} \bigwedge_{x=1}^{c-1} (gameover_z \lor \text{whiteline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \lor \bigvee_{d=0}^{c-1} \neg \text{white}_{x+d,y+c-1-d,z})
\end{align*}
\]

\[
\begin{align*}
&\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} \bigwedge_{x=1}^{w-h-c+1} (gameover_z \lor \neg \text{whiteline}_{l_x+l_h+l_d+(w-c+1)(x-1)+y,z} \lor \text{white}_{x+d,y+c-1-d,z})
\end{align*}
\]
16. If the game is not over and red has not cheated, then red has won at this move iff either white cheated or red obtained a line at this move. Similarly for white.

\[
\bigwedge_{z=1}^{hw} \left( \text{gameover}_z \lor \neg \text{redwin}_z \lor \text{redcheat}_z \lor \bigvee_{f=0}^{l_v+l_h+2l_d} \text{whitecheat}_z \lor \text{rline}_{f,z} \right)
\]

\[
\bigwedge_{z=1}^{hw} \bigwedge_{f=0}^{l_v+l_h+2l_d} \left( \text{gameover}_z \lor \text{redwin}_z \lor \text{redcheat}_z \lor \neg \text{rline}_{f,z} \right)
\]

\[
\bigwedge_{z=1}^{hw} \left( \text{gameover}_z \lor \text{redwin}_z \lor \text{redcheat}_z \lor \neg \text{whitecheat}_z \right)
\]

17. With the following clauses, we represent the fact that neither red or white can win by cheating and that red or white cannot cheat when it is not their turn.

\[
\bigwedge_{z=1}^{hw} \left( \text{redcheat}_z \land \neg \text{redwin}_z \right)
\]

\[
\bigwedge_{z=1}^{hw} \left( \text{whitecheat}_z \lor \neg \text{whitewin}_z \right)
\]

\[
\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \left( \neg \text{redcheat}_{2z} \right)
\]

\[
\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \left( \neg \text{whitecheat}_{2z-1} \right)
\]

18. These clauses dictate that if the game is already over, neither player can win.

\[
\bigwedge_{z=1}^{hw} \left( \neg \text{gameover}_z \lor \neg \text{redwin}_z \right)
\]
\[ \bigwedge_{z=1}^{hw} \left( \neg \text{gameover}_z \lor \neg \text{whitewin}_z \right) \]

19. This clause decides on which player we would like to win the game.

\[ \text{(redwin)} \]

20. Finally, we include clauses to break symmetry. If the width is even, we just say that the first move is on the left hand side of the board.

\[ \left( \text{gameover}_1 \lor \text{redcheat}_{0,1} \lor \bigvee_{y=1}^{\frac{w}{2}} \text{redmove}_{y,1} \right) \]

For odd width, symmetry remains as long as both players play in the centre. So we state that the first move which is not in the centre column is played on the left hand side of the board. This must happen by the \( h + 1 \) move at the latest, so we add the following \( h + 1 \) clauses.

\[
\begin{align*}
\left\lfloor \frac{1}{2}(h+1) \right\rfloor \bigwedge_{z=1}^{\frac{w}{2}} & \left( \text{gameover}_{2z-1} \lor \text{redcheat}_{0,2z-1} \lor \bigvee_{z'=1}^{z-1} \neg \text{redmove}_{\frac{w+1}{2},2z'-1} \lor \
\right. \\
& \left. \bigvee_{z'=1}^{z-1} \neg \text{whitemove}_{\frac{w+1}{2},2z'} \lor \bigvee_{y=1}^{\frac{w}{2}} \text{redmove}_{y,z} \right) \\
\left\lceil \frac{1}{2}(h+1) \right\rceil \bigwedge_{z=1}^{\frac{w}{2}} & \left( \text{gameover}_{2z} \lor \text{whitecheat}_{0,2z} \lor \bigvee_{z'=1}^{z-1} \neg \text{redmove}_{\frac{w+1}{2},2z'-1} \lor \
\right. \\
& \left. \bigvee_{z'=1}^{z-1} \neg \text{whitemove}_{\frac{w+1}{2},2z'} \lor \bigvee_{y=1}^{\frac{w}{2}} \text{whitemove}_{y,z} \right)
\end{align*}
\]

## 5 Experiments

Here we describe some brief details of some basic experiments on the Connect-\( c \) encoding. These are not intended as a serious attack on solving the problem, but as indicative of the performance of a QBF solver using standard techniques and without any heuristics or propagation rules tuned for Connect-4. For all experiments, we used a local implementation of conflict and solution directed backjumping for QBF[6] using the Watched Clauses data structure[7]. The experiments were performed on an Intel Celeron 1.70 Ghz computer with 128MB RAM. The time-out was 3600 seconds.

Clearly, the dominating factor is the size of the board, and run time increases very fast in this. Increasing \( c \) does generally increase run time, in some cases by a factor of 40 or so. However, the increase from \( c = 3 \) to 4 was not so dramatic, and in one case run time actually reduced. The rapidly increasing run time for \( c = 2 \) is disappointing.
Table 1. Sample Run Times on Connect-\(c\) problems. \(c\) is the number of pieces that must be in a line for a win, \(w\) is the width of the board, \(h\) is the height of the board and the time is the run time in seconds. '-' denotes a time-out on this problem. The standard Connect-4 problem is 4/7/6, and was unsolved.

<table>
<thead>
<tr>
<th>(c)</th>
<th>(w)</th>
<th>(h)</th>
<th>Solution Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>T</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>T</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>T</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>T</td>
<td>4.23</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>T</td>
<td>22.80</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>T</td>
<td>99.11</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>T</td>
<td>137.59</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>F</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>T</td>
<td>8.10</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>T</td>
<td>451.41</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>T</td>
<td>3.08</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>F</td>
<td>703.92</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Red is guaranteed to be able to win on its second move, irrespective of where it plays its first move on any of the board sizes tested. This suggests that either our encoding necessarily leads to additional search, or that our solver is not recognising that a simple winning strategy exists. On looking in closer detail, the reason for the result is that red does not necessarily play a piece to make a line on his second move when the choice is presented; often the first piece is played in the left-most column and the second in the right-most. While disappointing, this does at least show that there are significant improvements available, either in modified encodings or improved solvers, and these may carry over to the more interesting cases.

6 Conclusions

We should make two caveats about our work. First, we have not proven correctness of our encoding. Second, this is just one encoding of the game. The importance of representation is fundamental throughout artificial intelligence, and in both constraint satisfaction and in SAT increasing attention is being paid to the difference a good encoding makes. We have no reason for suggesting that our encoding will be particularly effective compared to other possible encodings of Connect-4.

Finally, we would like to encourage other researchers encoding complicated problems into QBF to report their work in detail as well as submitting instances to QBFLib. As well as giving others a chance to spot mistakes, it will enable the community to learn from each other’s encoding tricks.

Acknowledgements

The first author is supported by a Royal Society of Edinburgh SEELLD Support Fellowship and the second by EPSRC grant GR/55382. The authors are members of the APES research group and thank other members. We especially thank Toby Walsh for taking up the first author’s suggestion of posing the Connect-4 problem as a challenge for QBF.
References

   (1988)
   http://www.mrg.dist.unige.it/~enrico/ftp/01ijcai.ps.gz.
   http://www.dcs.st-and.ac.uk/~apes/papers/sat03.ps.