

Positional Games and QBF: The *Corrective* Encoding

Valentin Mayer-Eichberger¹ and Abdallah Saffidine²

¹ Technische Universität Berlin, Germany

² University of New South Wales, Sydney, Australia

Abstract. Positional games are a mathematical class of two-player games comprising Tic-tac-toe and its generalizations. We propose a novel encoding of these games into Quantified Boolean Formulas (QBFs) such that a game instance admits a winning strategy for first player if and only if the corresponding formula is true. Our approach improves over previous QBF encodings of games in multiple ways. First, it is generic and lets us encode other positional games, such as Hex. Second, structural properties of positional games together with a careful treatment of illegal moves let us generate more compact instances that can be solved faster by state-of-the-art QBF solvers. We establish the latter fact through extensive experiments. Finally, the compactness of our new encoding makes it feasible to translate realistic game problems. We identify a few such problems of historical significance and put them forward to the QBF community as milestones of increasing difficulty.

Keywords: QBF Encodings · Positional Games · Quantified Boolean Formula.

1 Introduction

In a *positional game* [12,3], two players alternately claim unoccupied elements of the board of the game. The goal of a player is to claim a set of elements that form a winning set, and/or to prevent the other player from doing so. TIC-TAC-TOE, and its competitive variant played on a 15×15 board, GOMOKU, as well as HEX are the most well-known positional games. When the size of the board is not fixed, the decision problem, whether the first player has a winning strategy from a given position in the game is PSPACE-complete for many such games. The first result was established for GENERALIZED HEX, a variant played on an arbitrary graph [8]. Reisch [24] soon followed up with results for GOMOKU [24] and HEX played on a board [25].

Recent work on the classical and parameterized computational complexity of positional games provides us with elegant first-order logic formulations of such domains [4,3]. We draw inspiration from this approach and introduce a practical implementation for such games into QBF. We believe that Positional Games exhibit a class of games large enough to include diverse and interesting

games and benchmarks, yet allow for a specific encoding exploiting structural properties.

Our contributions are as follows: (1) Introduce the *Corrective Encoding*: a generic translation of positional games into QBF; (2) We identify a few positional games of historical significance and put them forward to the QBF community as milestones of increasing difficulty; (3) Demonstrate on previously published benchmark instances that our encoding leads to more compact instances that can be solved faster by state-of-the-art QBF solvers; (4) Establish that the Corrective Encoding enables QBF solving of realistic small scale puzzles of interest to human players.

After a formal introduction to QBF and to positional games (Section 2), we describe the contributed translation of positional games into QBFs (Section 3). We then describe the selected benchmark game problems, including the proposed milestones (Section 4), before experimentally evaluating the quality of our encoding and comparing it to previous work (Section 5). We conclude with a discussion contrasting our encoding with related work (Section 6).

2 Preliminaries on QBF and Positional Games

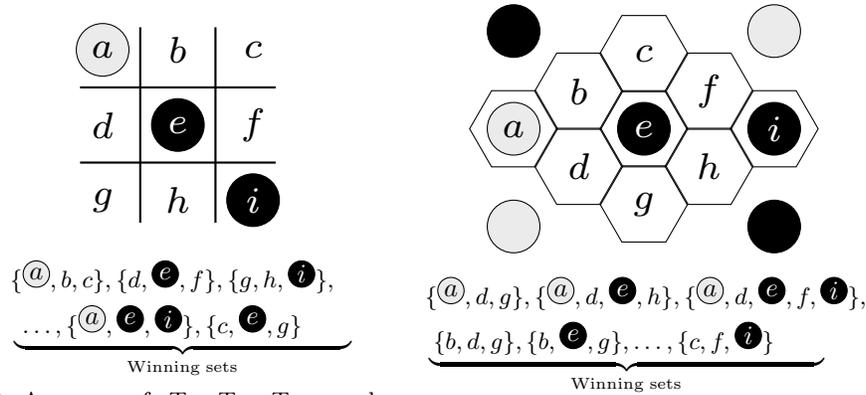
We assume a finite set of propositional variables X . A literal is a variable x or its negation $\neg x$. A clause is a disjunction of literals. A Conjunctive Normal Form (CNF) formula is a conjunction of clauses. An assignment of the variables is a mapping $\tau : X \rightarrow \{\perp, \top\}$. A literal x (resp. $\neg x$) is satisfied by the assignment τ if $\tau(x) = \top$ (resp. $\tau(x) = \perp$). A clause is satisfied by τ if at least one of the literals is satisfied. A CNF formula is satisfied if all the clauses are satisfied.

A QBF formula (in *Prenex*-CNF) is a sequence of alternating blocks of existential (\exists) and universal (\forall) quantifiers over the propositional variables followed by a CNF formula. A QBF formula may be interpreted as a game where an \exists and \forall player take turns building a variable assignment τ selecting the variables in the order of the quantifier prefix. The objective of \exists (resp. \forall) is that τ satisfies (resp. falsifies) the formula.

Positional games are played by two players on a hypergraph $G = (V, E)$. The vertex set V indicates the set of available positions, while the each hyperedge $e \in E$ denotes a winning configuration. For some games, the hyperedges are implicitly defined, instead of being explicitly part of the input. The two players alternatively claim unclaimed vertices of V until either all elements are claimed or one player wins. A *position* in a positional game is an allocation of vertices to the players who have already claimed these vertices. The *empty position* is the position where no vertex is allocated to a player. The notion of winning depends on the game type. In a *Maker-Maker game*, the first player to claim all vertices of some hyperedge $e \in E$ wins. In a *Maker-Breaker game*, the first player (*Maker*) wins if she claims all vertices of some hyperedge $e \in E$. If the game ends and player 1 has not won, then the second player (*Breaker*) wins. The class of (p, q) -positional games is defined similarly to that of positional games, except that on the first move, Player 1 claims q vertices and then each move after the

first, a player claims p vertices instead of 1. A *winning strategy* for player 1 is a move for player 1 such that for all moves of player 2 there exists a move of player 1... such that player 1 wins.

To illustrate these concepts, Figure 1 displays a position from a well-known Maker-Maker game, TIC-TAC-TOE, and a position from a Maker-Breaker game, HEX. Although the rules of HEX are typically stated as Player 1 trying to create a path from top left to bottom right and Player 2 trying to connect the top right to bottom left, the objective of Player 2 is equivalent to preventing Player 1 from connecting their edges [20]. Therefore, HEX can indeed be seen as a Maker-Breaker positional game.



(a) A game of TIC-TAC-TOE and its winning configurations: the set of aligned triples. (b) A game of HEX and its winning configurations for Black (Player 1): the set of paths from the top left edge to the bottom right edge.

Fig. 1: Two positional games played on the same vertices: $a-i$ where vertex a has been claimed by Player 2 and vertices e and i have been claimed by Player 1.

3 The Corrective Encoding

In this section we present the Corrective encoding (*COR*). First we define positional games formally, describe the set of variables and the clauses in detail and analyse the size of the encoding.

3.1 Description

A positional game is a tuple $\prod = \langle T_B, T_W, F, N, E_B, E_W \rangle$ consisting of:

- Disjoint sets T_B and T_W of time points in which Black and White make moves. We denote $T = T_B \cup T_W$ as the set of all time points that range

- from $\{1, 2, \dots, F\}$. For example, in a positional game where black starts and $p = q = 1$, T_B contains all odd and T_W all even numbers of T .
- $F \in \mathbb{N}$ the depth (or length) of the game.
 - A set of vertices $V = \{1 \dots N\}$ and two sets of hyperedges E_B and E_W of winning configurations for Black and White, respectively.

The remainder of this section defines a translation of a positional game configuration Π into a prenex QBF in CNF. For this we introduce variables defined in the following table. For readability we use a function style notation instead of variable subscripts. Let A denote the set of the two players $\{B, W\}$.

Variable	Description
$\text{time}(t)$	The game is still running by time point $t \in T$
$\text{board}(a, v, t)$	Player $a \in A$ owns vertex $v \in V$ at time point $t \in T$
$\text{occupied}(v, t)$	Vertex $v \in V$ is occupied at $t \in T$
$\text{win}(e)$	Black has claimed winning configuration $e \in E_B$
$\text{move}(v, t)$	Vertex $v \in V$ is chosen at $t \in T_a$ by player $a \in A$
$\text{moveL}(i, t)$	The moves for White encoded <i>logarithmically</i> , $0 \leq i < \lceil \log_2(N) \rceil$, $t \in T_W$
$\text{ladder}(i, t)$	Auxiliary variables for the ladder encoding, $i \in V$ and $t \in T$

The last two sets of variables are of technical nature; `ladder` is used to encode that a player must claim 1 vertex when it is their time t to move, whereas `moveL` encodes the choices of White in a way that prevents the universal player from falsifying the formula by breaking the rules of the game.

Quantification. Here we specify the quantifier prefix of our encoding. In the order of the time point $t = \{1 \dots F\}$ we introduce a level of quantifier blocks as follows:

$$\begin{aligned}
& \exists \text{time}(t) \\
& \text{if } t \in T_W \text{ then for all } 0 \leq i < \lceil \log_2(N) \rceil \quad \forall \text{moveL}(i, t) \\
& \qquad \text{for } v \in V \quad \exists \text{move}(v, t) \\
& \qquad \text{for } v \in V, a \in A \quad \exists \text{board}(a, v, t) \\
& \qquad \text{for } v \in V \quad \exists \text{occupied}(v, t) \\
& \qquad \text{for } i \in V \quad \exists \text{ladder}(i, t)
\end{aligned} \tag{1}$$

On the innermost level we have

$$e \in E_B \quad \exists \text{win}(e) \tag{2}$$

In the remainder of this section we list the clauses that make the body of the generated QBF instance. This body is constituted of sets of clauses encoding different aspects of the game. The encoding is almost entirely symmetric for both players apart from clauses which specify the interaction of the universal variables.

Time Handling. Variable $\text{time}(t)$ holds if the game is not over at time point t .

$$\{\neg \text{time}(t) \vee \text{time}(t-1) \mid t \in T\} \quad (3)$$

Structure of the board. Clauses (5) encode that both players cannot own the same vertex. One fundamental property of positional games is that claimed vertices never change owner. This basic property is captured in clause (6). Note that these two clauses are independent from the time variable and act also when the game is over. Once a vertex is claimed and $\text{board}(a, v, t)$ is true the implication chain in (6) sets all board variables for that vertex in the future, in particular the last one $\text{board}(a, v, F)$. Once the game is over (i.e. $\text{time}(t)$ is set to false), then all unclaimed vertices stay unclaimed (7) and the situation of the board at the last active move of the game is propagated through to the final time point.

$$\{\neg \text{board}(a, v, 0) \mid a \in A, v \in V\} \quad (4)$$

$$\{\neg \text{board}(B, v, t) \vee \neg \text{board}(W, v, t) \mid v \in V, t \in T\} \quad (5)$$

$$\{\neg \text{board}(a, v, t-1) \vee \text{board}(a, v, t) \mid a \in A, v \in V, t \in T\} \quad (6)$$

$$\{\text{time}(t) \vee \text{board}(a, v, t-1) \vee \neg \text{board}(a, v, t) \mid a \in A, v \in V, t \in T\} \quad (7)$$

The following clauses (8) - (10) define the meaning of *occupied*. Initially all vertices are unoccupied (8) and by definition a vertex is occupied if it is owned by Black or White.

$$\{\neg \text{occupied}(v, 0) \mid v \in V\} \quad (8)$$

$$\{\text{occupied}(v, t) \vee \neg \text{board}(a, v, t) \mid a \in A, v \in V, t \in T\} \quad (9)$$

$$\{\neg \text{occupied}(v, t) \vee \text{board}(B, v, t) \vee \text{board}(W, v, t) \mid v \in V, t \in T\} \quad (10)$$

Player's actions. The action clauses specify how the moves of the players affect the board. If player a claims vertex v at time $t \in T_a$, i.e. $\text{move}(v, t)$ is true, then the game still has to be running (11). This clause can also be understood as if the game is over no moves are allowed anymore. Moreover, when $\text{move}(v, t)$ is true, then vertex v was not occupied at the previous time point (12) and as a result of the action the vertex is occupied (13).

$$\{\text{time}(t) \vee \neg \text{move}(v, t) \mid v \in V, t \in T\} \quad (11)$$

$$\{\neg \text{occupied}(v, t-1) \vee \neg \text{move}(v, t) \mid v \in V, t \in T\} \quad (12)$$

$$\{\text{board}(a, v, t) \vee \neg \text{move}(v, t) \mid v \in V, a \in A, t \in T_a\} \quad (13)$$

White's Choice. The core of this encoding is how the universal variables interact with the rest of the encoding without having to prevent illegal moves by White. To avoid that White chooses too many vertices we encode the move logarithmically through variables $\text{move}_L(i, t)$. Moreover, these variables only actually imply a move of White in case the game is still running and the vertex was

unoccupied before. In case one of the prerequisites is not given, then no move will be forced by this clause. Let $L_1(v)$ denote to the set of indices that are 1 in the binary representation of v , likewise $L_0(v)$ where there is a 0. The following equality holds: $v = \sum_{j \in L_1(v)} 2^j$. For example, for $13 = 1101|_2$ the respective sets are $L_1(13) = \{0, 2, 3\}$ and $L_0(13) = \{1\}$.

$$\left\{ \begin{array}{l} \bigvee_{i \in L_1(v)} \neg \text{moveL}(i, t) \vee \bigvee_{i \in L_0(v)} \text{moveL}(i, t) \\ \vee \neg \text{time}(t) \vee \text{occupied}(v, t-1) \vee \text{move}(v, t) \mid v \in V, t \in T_W \end{array} \right\} \quad (14)$$

Notice how these clauses interact with (11) and (12) such that any choice for the universal variables is not able to cause a contradiction. In case White chooses a combination of `moveL` that does not imply an existing vertex, then still a move is selected for White to satisfy the ladder encoding (see (21) to (25))

Frame axioms. The following two clauses specify what happens when no action is performed on a position and the board variable is unchanged. Clause (15) says that in time points where player s does not claim vertex v and the vertex has not been previously owned by s then it will also not be owned in the following step. In time points t of the opponent to s , all unclaimed vertices by s will be unclaimed in the next time point. Clause (16) forces this.

$$\{\text{move}(v, t) \vee \text{board}(a, v, t-1) \vee \neg \text{board}(a, v, t) \mid a \in A, v \in V, t \in T_a\} \quad (15)$$

$$\{\text{board}(a, v, t-1) \vee \neg \text{board}(a, v, t) \mid a \in A, v \in V, t \in T \setminus T_a\} \quad (16)$$

Winning configuration. For each winning configuration $e \in E_B$ we have introduced a variable `win(e)` and clauses (17) specifies that at least one of the winning configuration have to be reached and (17) defines which vertices belong to it.

White should never reach a winning position, for this we introduce a clause for each winning positions specified in clauses (19). We only need to encode this for the last time point F due to the implication chain (6). This looks straightforward from the definition, but we need to make sure with other clauses that White is unable to play illegal moves to reach a winning position.

$$\left\{ \bigvee_{e \in E_B} \text{win}(e) \right\} \quad (17)$$

$$\{\neg \text{win}(e) \vee \text{board}(B, v, F) \mid v \in e, e \in E_B\} \quad (18)$$

$$\left\{ \bigvee_{v \in e} \neg \text{board}(W, v, F) \mid e \in E_W \right\} \quad (19)$$

$$\left\{ \text{win}(e) \vee \bigvee_{v \in e} \neg \text{board}(B, v, F) \mid e \in E_B \right\} \quad (20)$$

Number of moves. To restrict the number of moves we apply the *ladder encoding* [9] to translate the cardinality constraint specifying the number of moves that a player can make in a round.

The ladder essentially encodes the equivalence $\text{move}(i+1, t) \Leftrightarrow \neg \text{ladder}(i, t) \wedge \text{ladder}(i+1, t)$. As soon as a move variable is set to true, all following ladder variables are forced to true (21, 22) and all previous are forced to false (21, 23). This ensures that no two move variables can be set to true. Clauses (25, 26) ensure that at least one move variable is true.

$$\{\neg \text{ladder}(i, t) \vee \text{ladder}(i+1, t) \mid i \in V, i < \mathbf{N}, t \in T\} \quad (21)$$

$$\{\neg \text{move}(i, t) \vee \text{ladder}(i, t) \mid i \in V, t \in T\} \quad (22)$$

$$\{\neg \text{move}(i+1, t) \vee \neg \text{ladder}(i, t) \mid i \in V, i < \mathbf{N}, t \in T\} \quad (23)$$

$$\{\text{move}(1, t) \vee \neg \text{ladder}(1, t) \mid t \in T\} \quad (24)$$

$$\{\text{move}(i+1, t) \vee \text{ladder}(i, t) \vee \neg \text{ladder}(i+1, t) \mid i \in V, i < \mathbf{N}, t \in T\} \quad (25)$$

$$\{\neg \text{time}(t) \vee \text{ladder}(\mathbf{N}, t) \mid t \in T\} \quad (26)$$

These clauses also enforce a move by White even if White had chosen an already claimed vertex or no vertex with the `moveL` variables and clause (14) does not fire. This is a crucial property of the encoding. An arbitrary vertex for White is chosen by, the game continues and there is no backtracking even though the universal player acted illegal.

Initial positions. The QBF generator can also translate positional games that contain initial positions of White and Black, i.e. vertices that players own before the actual game starts. It is straight forward to turn this into an equivalent description without initial positions: For each initial position v of one player remove this vertex from all its winning configurations and remove all winning configurations of the opposing player that contain v . After this operation we can remove v from V and have an equivalent game.

Symmetry breaking. We employ a simple form of manual symmetry breaking by restricting the set of vertices from which the first move can be chosen. For instance in Generalized Tic-tac-toe (GTTT) and a $n \times n$ board, if this set contains the upper left triangle of the board (the set of coordinates (i, j) such that $1 \leq i \leq j \leq n/2$), the symmetries of the squared board are broken. Typically, for other games with some initial positions for White and Black there is not much need for symmetry breaking since row or column symmetries are usually already broken by such a position.

Consecutive moves. The positional game description need not have Black and White alternate moves: a description may allow a player to select several vertices consecutively. For instance, when $q = 2$ players claim two vertices in each round. For the sake of simplicity, our presentation of *COR* above does not break this symmetry. Our implementation avoids such symmetries by merging consecutive moves into a single turn where a subset of vertices of the right cardinality must be chosen. We implemented this cardinality constraint as a *sequential counter* [26]. It coincides with the ladder encodings in the single move case.

3.2 Size of the *COR* Encoding

It is straightforward to estimate the number of variables and clauses of the encoding. For instance, clauses with a description containing $v \in V, t \in T$ are generated at most $N \cdot F$ times. Since the depth of a game is limited by the number of vertices, the number of clauses is roughly $20N^2 + N \cdot |E_B| + |E_W|$.

\exists variables	\forall variables	binary clauses	ternary clauses	long clauses
$4FN$	$F \log_2(N)$	$3N + 12NF + N E_B $	$7NF$	$NF + E_B + E_W $

4 Instances

We used the encoding above to generate three sets of QBF instances based on some well-known positional games.³ The first two sets consist of positions of HEX and of a generalization of TIC-TAC-TOE on boards that are relatively small by human playing standards. Positions from that benchmark are fairly easy to solve even for relatively inexperienced human players of these games, and they can be solved almost instantaneously by specialized solvers. Our encoding of these positions should provide a reasonable challenge for QBF solvers as of 2019.

The third set contains the starting position of 4 positional games that are of interest to experienced human players and to mathematicians. At least 3 of these positions can be solved by specialized game algorithms developed in the 1990s and 2000s albeit with a non-trivial programming effort. The instances in this third set are out of reach of current QBF solvers and we believe that solving these positions with a QBF solver—via our encoding or a better one—can constitute a good milestone for the field.

4.1 Harary’s Tic-Tac-Toe and GTTT(p, q)

HARARY’S TIC-TAC-TOE is a Maker-Maker generalization of TIC-TAC-TOE where instead of marking 3 aligned stones, the players are trying to mark a set of cells congruent to a given polyomino. This type of game has received accrued interest from the mathematical community which was able to show the existence of a winning strategy or lack thereof for most polyomino shapes. GTTT(p, q) is a further generalization of HARARY’S TIC-TAC-TOE along the principle of (p, q)-positional games [7].

Previous work has already proposed an encoding of GTTT(p, q) played on small boards to QBF [7]. We refer to this existing in encoding as *DYS*. In our first set of benchmark, we encode the exact same GTTT(p, q) configurations as previous work. However, since our encoding is different, we obtain a different set of QBF instances. This provides us with an opportunity to directly compare our approach with existing work. We report results on the 96 instances

³ All our generated instances are available at github.com/vale1410/positional-games-qbf-encoding.

of GTTT(1, 1) played on a 4×4 board and compare formula size and solving performance with the DYS encoding.⁴

4.2 Hex

We use 20 hand-crafted HEX puzzles of board size 4×4 up to 7×7 that all have a winning strategy for Player 1. The first 19 of these HEX instances are of historical significance. Indeed, they were created by Piet Hein, one of the inventors of HEX and first appeared in the Danish newspaper *Politiken* [14,20] during World War II [13].⁵ The remaining puzzle is a 5×5 position proposed by Cameron Browne, it arose during standard play and offers a significant challenge for the Monte Carlo Tree Search (MCTS) algorithm and the associated RAVE enhancement [5]. This is noteworthy because MCTS is the foundational algorithm behind the top artificial players for numerous games including HEX and GO [6].

4.3 Challenges

We now put forward a few positional games that have attracted the attention of board game players as well as AI or mathematics researchers. Table 1 summarizes the proposed challenges together with the size of their *COR* encoding.

QUBIC, also known as 3-DIMENSIONAL TIC-TAC-TOE, is played on a $4 \times 4 \times 4$ cube and the goal is to mark 4 aligned cells, horizontally, vertically, or diagonally. Our first domain was solved for the first time in 1980 by combining depth-first search with expert domain knowledge [21]. A second time in the 1990s using Proof Number Search (PNS), a tree search algorithm for two-player games [27].

The second domain, *freestyle* GOMOKU, is played on a 15×15 board and the goal is to mark 5 aligned cells, horizontally, vertically, or diagonally. Already in the 1930s, GOMOKU was perceived to be giving an overwhelming advantage to Black [27], the starting player, and by the 1980s professional GOMOKU players from Japan had claimed that the initial position admitted a Black winning strategy [1]. This was confirmed in 1993 using the PNS algorithm, a domain heuristic used to dramatically reduce the branching factor, and a database decomposing the work in independent subtasks [1].

CONNECT6 is akin to GOMOKU but the board is 19×19 , the goal is 6 aligned cells, and players place 2 stones per move [16]. The *Mickey Mouse* setup once was among the most popular openings of CONNECT6 until it was solved in 2010 [30]. The resolution of CONNECT6 was based on PNS distributed over a cluster.

Our last challenge is an open-problem in HARARY'S TIC-TAC-TOE which corresponds to achieving shape SNAKY  on a 9×9 board. This problem was recently put forward as an intriguing challenge for QBF solvers [7] and we offer here an alternative, more compact, encoding.

⁴ We also generated the instances for larger values of p and q but the formulas are much easier to solve and provide less insight.

⁵ We are grateful to Ryan Hayward and Bjarne Toft for providing us with this collection of puzzles.

Table 1: Selected problems put forward to the QBF community and size of their Corrective encoding. No preprocessing has been applied to these instances.

Challenge problem		First system- atic solution	Size in QBF				
Domain	Variant		#qb	# \forall	# \exists	#cl	#lits
QUBIC	$4 \times 4 \times 4$	1980	65	192	29275	80343	245723
SNAKY	9×9	open	81	280	47137	130702	403595
GOMOKU	15×15 freestyle	1993	225	896	357097	991430	3078404
CONNECT6	19×19 Mickey Mouse	2010	179	1602	510651	1527064	5031059

5 Analysis

5.1 Setup of Experiments

When solving problems encoded in QBF, the ideas underlying the encoding of a problem are only a factor in whether the instances can be solved withing reasonable resources. Two other important factors are the specific solver invoked and the kind of preprocessing performed on the instance before solving, if any. In our experiments, we chose four state-of-the-art QBF solvers, including the top three solvers of the latest QBF Competition⁶ and three preprocessors, as indicated in Table 2a.⁷ All software was called with default command line parameters.

Table 2: Software used and resulting performance on the first benchmark.

(a) Solvers and preprocessors used in the experiments.

	Software	Shorthand
Solver	DepQbf 6.03 [18]	depqbf
	Caqe 4.0.1 [23]	caqe
	Qesto 1.0 [17]	qesto
	Qute 1.1 [22]	qute
Preprocessor	QratPre+ 2.0 [19]	Q
	HQSPRE 1.4 [29]	H
	Bloqqr v37 [15]	B
	None	N

(b) Solver performance depending on the encoding, always using the best preprocessor.

	Solver	Preproc.	S	T	\perp	U	time(s)
<i>DYS</i>	caqe	B	92	31	61	4	11468
	depqbf	Q	82	28	54	14	27211
	qesto	BQ	74	27	47	22	34887
	qute	BQ	69	27	42	27	35837
<i>COR</i>	caqe	Q	96	34	62	0	4099
	depqbf	N	96	34	62	0	602
	qesto	BQ	96	34	62	0	3404
	qute	B	82	30	52	14	23266

The experiments have been running on a i7-7820X CPU @ 3.60GHz with 8 cores, 24GB RAM. All solvers have been running with a dedicated single core.

⁶ <http://www.qbflib.org/eval19.html>

⁷ We also attempted to use the `rareqs` QBF solver, but it timed out on almost all instances. Preprocessor H was omitted due to large timeouts.

5.2 Experimental comparison of our new encoding to the *DYS*

Before attempting to solve positional games, let us first examine how large and amenable to preprocessing the generated encodings are. We use an approach inspired by recent work on QBF preprocessors [19] and report in Table 3 the number of quantifier blocks, universal and existential variables, clauses, and literals, as well the time needed for the preprocessing of a representative instance of $\text{GTTT}(1, 1)$. On both the existing *DYS* encoding and our proposed *COR*, we test each preprocessor individually as well as the outcome of running one preprocessor then another. No preprocessor timed out.

Table 3: Preprocessing on 5×5 instance `gttt_1_1_00101121_5x5_b`

		N	Q	H	B	QB	BQ	HQ	QH
<i>DYS</i>	#qb	25	25	25	25	25	25	25	25
	# \forall	300	300	300	299	299	299	300	300
	# \exists	21056	12058	7553	2750	2605	2750	7553	7545
	#cl	53589	35875	32978	21434	19625	19257	30191	30444
	#lits	191485	127366	145480	120237	106590	105697	103312	135061
	time(s)	0	46	1210	9	55	22	1233	2030
<i>COR</i>	#qb	25	25	25	25	25	25	25	25
	# \forall	60	60	58	58	58	58	58	58
	# \exists	4649	3127	3433	1396	1360	1396	3432	2981
	#cl	12490	8183	28918	8245	7943	7394	14899	19757
	#lits	28544	19672	117953	35457	34412	30732	52381	88712
	time(s)	0	0	275	2	2	2	277	20

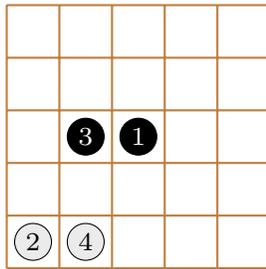
Two observations stand out when looking at Table 3. First, *DYS* is much larger than *COR* across most of the size dimensions. Second, preprocessors seem to be much more capable at reducing the size of the *DYS* instance than the size of the *COR* instance. Our interpretation is that it is a direct consequence of the effort we have put in crafting the proposed new encoding: there is relatively little improvement room left for the preprocessors to improve the formulas. Since the size of a formula directly impacts how hard it is to solve, we expect QBF solvers to struggle much more with *DYS*-encoded game instances than with *COR*-encoded ones.

In our next experiment, we compare how well QBF solvers manage to solve *GTTT* game instances when encoded with *DYS* and with *COR*. Since different preprocessors tend to play to the strength of different solvers, we report the preprocessor that lead to the best performance for each solver separately. We compare the solvers and the encodings using 96 $\text{GTTT}(1, 1)$ 4×4 game instances and assuming a timeout of 1000s, Table 2b displays for each configuration the number of formulas solved (S), proven satisfiable (\top), proven unsatisfiable (\perp), and unsolved (U), as well as the cumulative time spent by the solver.

The data in Table 2b confirms our intuition. GTTT games can be more effectively solved through our encoding: 3 out of 4 solvers solve all *COR* instances whereas none solve all *DYS* instances, and *COR* instances are solved between up to two orders of magnitude faster. Furthermore, our results demonstrate that the choice of encoding has a bigger impact than the choice of solver and preprocessor.

5.3 Solving increasingly realistic games

Iterative deepening is an algorithmic principle in game search recommending to search for a d -move strategy before attempting to find a deeper one. This principle lets one benefit from the memory-efficiency of depth-first search and from the completeness of breadth-first search. It is easily adapted to solving games via QBF: encode one formula per depth and attempt to solve them one by one in order. We demonstrate the benefits of this adaptation in Figure 2: the position admits a depth 5 winning strategy. Proving the existence of a strategy of depth ≤ 5 needs 0.1 second, but the formula stating the existence of a strategy of depth ≤ 13 needs 2 hours to be proven. Although the outcome of searches at short depths is subsumed by that of deeper searches, the exponential growth of the required solving time makes iterative deepening a worthy trade-off.



(a) Position after White's mistaken second move.

d	$\not\models \phi_d$	$\models \phi_d$
1	0.01	
3	0.01	
5		0.10
7		2.33
9		22.0
11		334.
13		7753.

(b) Time (s) needed by depqbf-N-COR to establish whether Black can win within depth $\leq d$.

Fig. 2: GTTT 5×5 L game where Black can force a 5-move win.

Our final benchmark is the set of 20 historical HEX puzzles described in Section 4.2. Except for one puzzle on which *qesto* needed more than 10GB of memory, all positions on board sizes 5×5 or less can be solved by state of the art QBF solvers (Table 4). The 5 remaining puzzles remain out of reach at this stage. This is a remarkable feat: for the first time, the QBF technology can address game situations considered of interest to human players.

Table 4: Solving classic HEX puzzles by encoding them through *COR*.

Puzzle	size	depth	caqe-Q		depqbf-N		questo-BQ	
			d	$\not\models \phi_{d-2}$	$\models \phi_d$	$\not\models \phi_{d-2}$	$\models \phi_d$	$\not\models \phi_{d-2}$
Hein 04	3x3	05	0.01	0.01	0.02	0.02	0.01	0.00
Hein 09	4x4	07	0.01	0.11	0.03	0.15	0.01	0.06
Hein 12	4x4	07	0.02	0.10	0.05	0.22	0.00	0.02
Hein 07	4x4	09	0.30	4.31	0.33	5.69	0.09	1.66
Hein 06	4x4	13	10.2	15.5	2.95	17.7	3.92	9.79
Hein 13	5x5	09	0.24	15.6	0.72	17.1	0.06	4.61
Hein 14	5x5	09	0.38	19.0	1.24	42.4	0.18	4.40
Hein 11	5x5	11	5.17	240.	21.0	457.	1.84	23.6
Hein 19	5x5	11	2.29	44.4	3.60	80.8	0.91	13.1
Hein 08	5x5	11	4.13	104.	6.84	247.0	1.98	34.4
Hein 10	5x5	13	367.	4906.	443.	10259.	74.3	1543.
Hein 16	5x5	13	651.	8964.	1794.	8506.	278.	4406.
Hein 02	5x5	13	719.	22526.	1258.	10876.	317.	2957.
Hein 15	5x5	15	3247.	26938.	2928.	19469.	767.	MO
Browne	5x5	09	0.87	57.45	0.91	21.2	0.25	2.89

6 Comparison to Related Encodings

Despite the similarity of QBF solving with systematic search for winning strategies in games with perfect information, to the best of our knowledge there are not many encodings published that have attempted translation from games to QBF. After Walsh [28] challenged the QBF community to solve CONNECT4 on a 7×6 board using QBF techniques in 2003, there was some activity in this direction but with rather little success in experiments.

The first concrete and implemented encoding of a game is the work by Gent and Rowley that presents a translation from CONNECT4 to QBF [11]. Building upon Gent’s encoding [2] presents a QBF encoding of an *Evader/Pursuer* game that resembles simpler chess-like endgames on boards of size 4×4 and 8×8 . Both papers analyse problems that are not positional games, but the authors do report similar challenges in the construction of QBF formulas.

The closest to our encoding is the *DYS* encoding of GTTT [7] that has been proposed recently which is an adaptation of the encoding for [11]. The structure and clauses for these two encodings are similar so our more detailed comparison to *DYS* also applies to the encoding in [11].

Apart from these games and encodings to the best of our knowledge we are not aware of any other QBF formulations of games that improve upon them. In the remainder of this section we will go into various properties regarding *COR* and the existing encodings.

Generalisation. Although we presume that the ideas behind *DYS* could be extended to arbitrary positional games, the description of the encoding was tailored to the GTTT domain. We chose positional game as an input formalism to

reach a reasonable level of generalisation, i.e. many two-player can be formulated as positional games, but enough structural properties from the description to create neat encodings.

Clausal Description. The description of the clauses in *DYS* is not purely clausal and contains many equivalences. The general translation of equivalences into CNF introduces auxiliary (Tseytin) variables of which some can be avoided through better techniques. Our description consists only of clauses and much work has gone to reduce the number of variables.

Size. The size of our encoding is quadratic in the number of vertices and linear in the number of winning configurations. Even for smaller boards these scale effects materialize and we demonstrate and discuss our observations in the following Section.

Binary Clauses. Binary clauses enjoy many theoretical and practical advantages. A purely 2CNF problem can be solved in polynomial time, SAT solvers invest in the special treatment of binary clauses to speed up propagation and learning. This should also apply to *QBF* solving. Discovering a binary clause structure of a certain aspect of a problem description might be the key to crafting encodings that also solve fast. Our encoding demonstrates that many aspects of positional games can be captured through sets of binary clauses forming chains.

Timing. The variable $gameover_z$ in *DYS* has the same meaning as *time* in our encoding, it marks the end of the game and is crucial to prevent white from reaching a winning position after black has already won. In *DYS* this variable is added to almost all clauses such that when the game is finished the universal variables cannot falsify these clauses anymore. This technique produces correct encodings, but affects propagation negatively and weakens clause learning. We avoid such weakening of the other clauses by de-coupling *time* and the board structure of a game. When the game is finished (i.e. *time* is set to false) independently of the choices of the players no moves are allowed anymore and all empty board positions are propagated through to the end. We expect that *COR* makes it easier for the solver to exploit transposition of sequence of moves leading to the same board configuration than previous encodings.

Monotonicity. Using the property of monotonicity of positional games—the set of claimed vertices only grows throughout the game—our clauses manage to capture this property more directly than the previous encodings. For instance, *DYS* introduces variables that are true if a player wins in time point t by reaching a winning configuration. We avoid the need to know explicitly by which time point a player won via propagating the claimed vertices through to the last time point and only need to test for the winning configuration of both players there. Through de-coupling the time aspect of games from checking winning positions our encoding has fewer variables and shorter clauses, that again benefit propagation.

Adapted Log Encoding. This concept was first introduced for the encoding of quantified constraint satisfaction problems (CSPs) into QBF [10]. There, a logarithmic number of universal variables encode the binary representation of a

CSP variable. This technique was also applied with success in a game encoding to QBF [2].

Indicator Variables. To the best of our knowledge all translations of games (including non-positional ones) to QBF introduce variables that indicate types of illegal moves by white. Such variables again weaken the encoding due to longer clauses. The encoding *DYS* has the following types of illegal moves; white claims too many vertices, too few vertices, already occupied vertices. All of these are explicitly encoded in *DYS* whereas *COR* corrects white’s move. The key insight in the work of [2] raises the question how to address White’s illegal behavior without weakening the encoding or introducing auxiliary variables.

7 Conclusion and Future Work

We consider the craft of finding efficient translations of a problem description to the clausal representation an important step towards better performances of *QBF* solvers. Our investigation regarding the class of positional games demonstrates that a carefully crafted translation using structural properties to decrease and shorten clauses and decrease the number of variables improves the applicability beyond trivial problems. We list some key insights:

- Binary implication chains capturing monotone structural properties of problem are crucial.
- Variables representing illegal moves by the universal player can be avoided.
- Encoding the choices of the choices by the universally player logarithmically helps in this problem description.
- Preprocessing is crucial for previous encodings to perform, whereas on our encoding has minor impact.

Our investigation focused on clause representation and we have yet to extend to non-clausal description languages to QBF such as QCIR.

The insights from our investigation can be applied to translation of other almost-positional games and to planning problems with similar structures.

Although HEX is a positional game, its hypergraph representation is exponential in the board size because it needs to account for all paths between a pair of sides. For larger boards one will need an implicit representation of the paths between the two sides, possibly drawing inspiration from existing first-order logic modeling [4].

References

1. Louis V. Allis, H. Jaap van den Herik, and Matty P.H. Huntjens. Go-Moku solved by new search techniques. *Computational Intelligence*, 12(1):7–23, 1996.
2. Carlos Ansótegui, Carla P. Gomes, and Bart Selman. The achilles’ heel of QBF. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 275–281. AAAI Press / The MIT Press, 2005.

3. Édouard Bonnet, Serge Gaspers, Antonin Lambilliotte, Stefan Rümmele, and Abdallah Saffidine. The parameterized complexity of positional games. In *International Colloquium on Automata, Languages and Programming (ICALP)*, July 2017.
4. Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. On the complexity of connection games. *Theoretical Computer Science (TCS)*, 644:2–28, 2016.
5. Cameron Browne. A problem case for UCT. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(1):69–74, 2013.
6. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.
7. Diptarama, Ryo Yoshinaka, and Ayumi Shinohara. QBF encoding of generalized tic-tac-toe. In *4th International Workshop on Quantified Boolean Formulas (QBF) co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 14–26, Bordeaux, France, July 2016.
8. Shimon Even and Robert Endre Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the ACM*, 23(4):710–719, 1976.
9. Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into sat. In *CP 2004 Workshop on Modelling and Reformulating CSPs*, pages 95–110. Third International Workshop, 2004.
10. Ian P. Gent, Peter Nightingale, and Andrew Rowley. Encoding quantified cps as quantified boolean formulae. In *In Proceedings of ECAI-2004*, pages 176–180, 2004.
11. Ian P. Gent and Andrew G. D. Rowley. Encoding connect-4 using quantified boolean formulae. In *Modelling and Reformulating Constraint Satisfaction Problems*, pages 78–93, 2003.
12. Alfred W. Hales and Robert I. Jewett. Regularity and positional games. *Transactions of the American Mathematical Society*, 106:222–229, 1963.
13. Ryan B. Hayward and Bjarne Toft. *Hex, the full story*. AK Peters/CRC Press/Taylor Francis, 2019.
14. Piet Hein. Polygon. *Politiken*, December 26, 1942–August 11, 1943.
15. Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *Journal of Artificial Intelligence Research (JAIR)*, 53:127–168, 2015. fmv.jku.at/bloqqr.
16. Ming Yu Hsieh and Shi-Chun Tsai. On the fairness and complexity of generalized k -in-a-row games. *Theoretical Computer Science*, 385(1):88–100, 2007.
17. Mikolás Janota and João Marques-Silva. Solving QBF by clause selection. In *Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 325–331, Buenos Aires, Argentina, July 2015. sat.inesc-id.pt/~mikolas/sw/questo/.
18. Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *26th International Conference on Automated Deduction (CADE)*, pages 371–384, Gothenburg, Sweden, August 2017. lonsing.github.io/depqbf/.
19. Florian Lonsing and Uwe Egly. Qratpre+: Effective QBF preprocessing via strong redundancy properties. In *Theory and Applications of Satisfiability Testing - SAT 2019 Lisbon, Portugal*, pages 203–210, July 2019. lonsing.github.io/qratpreplus/.
20. Thomas Maarup. Everything you always wanted to know about Hex but were afraid to ask. Master’s thesis, University of Southern Denmark, 2005.

21. Oren Patashnik. Qubic: $4 \times 4 \times 4$ Tic-Tac-Toe. *Mathematics Magazine*, 53(4):202–216, 1980.
22. Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. In Serge Gaspers and Toby Walsh, editors, *20th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 10491 of *Lecture Notes in Computer Science*, pages 298–313. Springer Verlag, August 2017. ac.tuwien.ac.at/research/qute/.
23. Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 136–143, Austin, Texas, USA, September 2015. github.com/ltentrup/caqe.
24. Stefan Reisch. Gobang ist PSPACE-vollständig. *Acta Informatica*, 13(1):59–66, 1980.
25. Stefan Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15:167–191, 1981.
26. Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, pages 827–831, 2005.
27. H. Jaap Van Den Herik, Jos W.H.M. Uiterwijk, and Jack Van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.
28. Toby Walsh. Challenges for sat and qbf. In *Keynote*, SAT-03.
29. Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. Hqspre— an effective preprocessor for QBF and DQBF. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 373–390, Uppsala, Sweden, April 2017. projects.informatik.uni-freiburg.de/projects/dqbf.
30. I-Chen Wu, Hung-Hsuan Lin, Der-Johng Sun, Kuo-Yuan Kao, Ping-Hung Lin, Yi-Chih Chan, and Bo-Ting Chen. Job-level proof number search. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(1):44–56, 2013.