Validation of QBF Encodings with Winning Strategies

Irfansha Shaik
Aarhus University, Denmark

Maximilian Heisinger
Johannes Kepler Universität Linz, Austria

Martina Seidl
Johannes Kepler Universität Linz, Austria

Jaco van de Pol
Aarhus University, Denmark

Abstract

When using a QBF solver for solving application problems encoded to quantified Boolean formulas (QBFs), mainly two things can potentially go wrong: (1) the solver could be buggy and return a wrong result or (2) the encoding could be incorrect. To ensure the correctness of solvers, sophisticated fuzzing and testing techniques have been presented. To ultimately trust a solving result, solvers have to provide a proof certificate that can be independently checked. Much less attention, however, has been paid to the question how to ensure the correctness of encodings.

The validation of QBF encodings is particularly challenging because of the variable dependencies introduced by the quantifiers. In contrast to SAT, the solution of a true QBF is not simply a variable assignment, but a winning strategy. For each existential variable x, a winning strategy provides a function that defines how to set x based on the values of the universal variables that precede x in the quantifier prefix. Winning strategies for false formulas are defined dually.

In this paper, we provide a tool for validating encodings using winning strategies and interactive game play with a QBF solver. As the representation of winning strategies can get huge, we also introduce validation based on partial winning strategies. Finally, we employ winning strategies for testing if two different encodings of one problem have the same solutions.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Theory of computation → Logic and verification

Keywords and phrases QBF, Validation, Winning Strategy, Equivalence, Certificates

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.24

Supplementary Material *Software*: `https://github.com/irfansha/SQval`

Funding *Maximilian Heisinger*: Supported by the LIT AI Lab funded by the State of Upper Austria.
Martina Seidl: Supported by the LIT AI Lab funded by the State of Upper Austria.

1 Introduction

Quantified Boolean formulas (QBFs) extend propositional formulas with universal and existential quantifiers over the Boolean variables [3], rendering their decision problem PSPACE-complete. As many application problems from artificial intelligence and formal verification have efficient QBF representations (see [16] for a survey) and as much progress has been made in the development of QBF solving tools [13], QBFs provide an appealing framework for solving such problems. In practice, however, obtaining correct and concise QBF encodings can be complex and error-prone as currently hardly any support for testing and debugging QBF encodings is available. The complexity of getting correct encodings comes on the one hand from the fact, that QBFs provide only a low-level language operating on the bit level

and on the other hand from their interactive nature resulting from the quantifier alternations. The evaluation of a QBF can be seen as a two-player game between the existential and the universal player, where the existential player's goal is to satisfy the formula and the universal player's goal is to falsify the formula. Hence, models of true QBFs and counter-models of false QBFs are also called their *winning strategies*.

A winning strategy is a set of Boolean functions that defines how to set existential (universal) variables of a QBF to satisfy (falsify) a true (false) formula. A winning strategy provides the solution of the encoded application problem like the plan of a planning problem or the witness that no plan exists. Basically, winning strategies can be obtained in two forms: (1) statically, in terms of serialized functions that map existential variables to universal variables (or vice versa) [1], or (2) as interactive game play either with a QBF solver as opponent or by proof rewriting [6]. In contrast to SAT, where a solution is simply a variable assignment, the correctness of a winning strategy is challenging to validate, because of the dependencies between the variables. To prove that a winning strategy in serialized form is indeed a solution of a given QBF ϕ, a co-NP-hard problem has to be solved if ϕ is true, and an NP-hard problem has to be solved otherwise. This check can be automated by using a SAT solver. To show that a winning strategy is indeed a solution of an application problem, mainly remains a manual task. It is even non-trivial to find out if two variants of a problem encoding (e.g., a basic version and an optimization) share some common solutions.

We present a tool that supports the interactive testing of encodings based on serialized winning strategies in terms of Boolean functions as well as on dynamic validation, i.e., playing interactively against a QBF solver. We also propose a combination of both approaches for scalable validation based on partial winning strategies. To automate the testing process of an encoding, we implement a fuzz-testing approach for randomly exploring different parts of the search space. Fuzz testing has been successfully employed for testing solvers [5], but not for testing encodings. Finally, we present an approach to compare different variants of an encoding. While it is in general not feasible to prove that two encodings are equivalent, our tool supports testing if the two encodings share a common winning strategy. With a case study, we illustrate how our tool can be used to evaluate and understand QBF encodings.

2 Preliminaries

We consider closed QBF formulas in prenex normal form, i.e., of the form $Q_1X_1 \cdots Q_nX_n.\phi$, where quantifiers $Q_i \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$, and X_i are disjoint sets of variables. The matrix ϕ is a propositional formula over $\bigcup X_i$. A QBF $\forall X\Pi.\phi$ is true iff both $\forall X'\Pi.\phi[x/\top]$ and $\forall X'\Pi.\phi[x/\bot]$ are true where $X' = X \setminus \{x\}$ and $\phi[x/t]$ is obtained from ϕ by replacing x by truth constant t. A QBF $\exists X\Pi.\phi$ is true iff $\exists X'\Pi.\phi[x/\top]$ or $\exists X'\Pi.\phi[x/\bot]$ is true. Often, the semantics of a QBF is also expressed as a two-player game: In the ith move, the values of the variables in X_i are chosen by the existential player if $Q_i = \exists$ and otherwise by the universal player. If all variables are assigned and the formula evaluates to true (false), then the existential (universal) player wins. A QBF is true (false) iff there is a winning strategy for the existential (universal) player. Winning strategies can be represented in terms of *Skolem/Herbrand functions*. A winning strategy for a true QBF $\Pi.\phi$ over existential variables X is a set $S = \{f_x \mid x \in X\}$ of Skolem functions such that each f_x is a Boolean function over the universal variables preceding x in the prefix and $\phi[X/S]$ is valid. Dually, a winning strategy for a false QBF $\Pi.\phi$ over universal variables Y is a set $H = \{f_y \mid y \in Y\}$ of Herbrand functions such that f_y is a Boolean function over the existential variables preceding y in the prefix and $\phi[Y/H]$ is unsatisfiable.

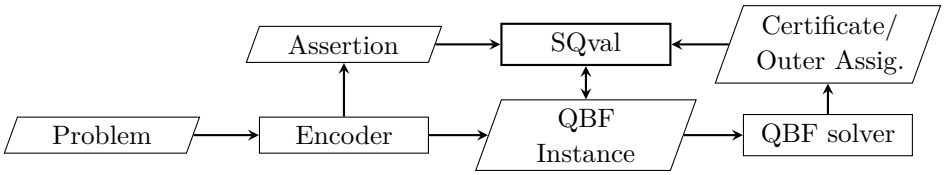


Figure 1 Validation workflow with SQval.

3 QBF Validation With Interactive Play

With our tool SQval (Scalable QBF Validator), we support the validation of QBF encodings independent of any specific QBF application. The general workflow is shown in Figure 1. The tool accepts both formulas in prenex conjunctive normal form (PCNF) and formulas in prenex non-CNF format. Hence, the QDIMACS format for PCNF formulas and the QCIR format [9] for prenex non-CNF formulas are supported.

3.1 Playing with Skolem/Herbrand Functions

Given a true QBF $\exists X_1 \forall Y_1 \exists X_2 \forall Y_2 \ldots \exists X_n \forall Y_n . \phi$ which has the set S of Skolem functions as one solution. These functions can be used to calculate the values of the existential variables in X_i based on provided values of the universal variables Y_j with $j < i$. As the variables of X_1 occur in the outermost quantifier block, they do not depend on any universal variables. Therefore, their Skolem functions are constant and can immediately be provided. Then the user has to enter the values for variables Y_1. Alternatively, they can also be randomly selected. Based on the values of the variables in Y_1, the values of the variables in X_2 are calculated. This procedure is repeated until all variables are assigned a value and the matrix ϕ evaluates to true. Evaluation of a false QBF works dually by using Herbrand functions.

There are solvers like Caqe [14] that construct Herbrand and Skolem functions during the solving process and there are frameworks that retrospectively extract functions from proofs produced by the solvers [11, 2, 4, 12]. Such functions allow to independently certify the correctness of a solving result by using a SAT solver. Little information, however, is provided on the correctness of the problem encoding. Here, we use the functions to "execute" test cases and interpret the results in an interactive manner. In our interactive play, Skolem functions automatically decide the moves of the existential player in the case of true instances. For false instances, Herbrand functions provide the moves of the universal player.

The Herbrand and Skolem functions have to be precomputed and can be provided in the AIGER format or as propositional formula in CNF format. In addition, encoding-specific assertions can be provided to SQval as CNF formulas. These assertions are checked under the full variable assignment at the end of the play. Such an assertion could be used, for example, to check if some goal condition is met or if some invariants are not violated in the game play. An example is given in the case study presented in Section 5.

The advantage of this method is the one-time cost of generating Skolem/Herbrand functions. Their evaluation is computationally cheap, because only truth values need to be propagated. However, even for simple problems, these functions can become very large and producing such functions often considerably slows down the solvers, because powerful pre- and inprocessing techniques have to be disabled. To overcome this drawback, we present an alternative approach in the following.

3.2 Playing with a QBF Solver

Given a true QBF of the form $\Phi_1 = \exists X_1 \forall Y_1 \Pi.\phi_1$ or a false QBF of the form $\Phi_2 = \forall Y_2 \exists X_2 \Pi.\phi_2$, most QBF solvers are able to provide assignments σ_{X_1} and σ_{Y_2} such that Φ_1 evaluates to true under σ_{X_1} and Φ_2 evaluates to false under σ_{Y_2}. If we apply σ_{X_1} on Φ_1, we obtain the true QBF $\Phi_1' = \forall Y_1.\phi'$. As this QBF has to be true for all assignments of variables Y_1, we pick now an interesting assignment and apply it on Φ_1' to obtain the true QBF Φ_1'' which starts with an existential quantifier block (or it has become the empty formula). Now we can ask a QBF solver for a satisfying assignment of these outermost variables and proceed in this way until all variables are finally assigned. Similarly, we can interactively evaluate Φ_2.

Based on this approach, we can replace large Skolem/Herbrand functions and avoid slower QBF solvers. The disadvantages of the approach is that a linear number of QBF problems must be solved: one for each round of validation.

3.3 Hybrid Validation

For many instances, the two interactive play approaches presented above are either limited by the size of the Skolem/Herbrand functions or by the costs of the QBF solver calls. As a solution, we suggest combining both approaches. First, we precompute Skolem/Herbrand functions only for the variables in the first k quantifier blocks of a QBF Φ. Based on these functions, we calculate the truth values of the variables which they define, and replace them respectively in Φ. Now we obtain a QBF Φ' with k quantifier alternations less and fewer variables. Then we proceed with evaluating Φ' by playing against a QBF solver.

The certification framework QBFcert [11] supports the generation of partial winning strategies, so we can use the respective options to obtain the functions of the variables from the first k quantifier blocks. We also provide an extractor for obtaining partial winning strategies from a full winning strategy by specifying the variables that should be considered. The partial winning strategies remain smaller than full winning strategies and we can take advantage of faster non-certifying solvers for the validation. Note that, one can generate assignments in the outer-most quantifier block with almost any non-certifying solver. In case the certifying solvers are too slow, one would use such an assignment to speed up the validation. In Section 5 we will demonstrate the memory/time tradeoff with partial strategies.

4 Common Winning Strategies of Two Encodings

Often, the same problem can be encoded in many ways resulting in formulas with different winning strategies. Consider for example two encodings of a two-player game for which we have a basic reference encoding that is very hard to solve, but which is most likely correct. Further, there is an optimized encoding which can be solved faster and which only has winning strategies that must also be solutions to the basic encoding. Therefore, we call the basic encoding *more relaxed* than the optimized encoding. To increase the trust in the optimized encoding, we want to validate if a found winning strategy is indeed a solution of the basic encoding. To this end, we want to take a winning strategy of one formula and enrich the other formula with this encoding. If this enriched formula has the same truth value, then we can conclude that the winning strategy of the first encoding is also a winning strategy of the second encoding. As two encodings might be defined over different variables, we need to introduce a set of common variables $\mathcal{C}$ that occur in both formulas (in practice some renaming of the variables not occurring in $\mathcal{C}$ might be necessary to avoid name clashes). This validation approach also works both for true and false formulas.

We first define a subsumption check between two true QBF formulas ϕ_1, ϕ_2. For this purpose, we use certificates, which are essentially winning strategies.

▶ **Definition 1.** *Let ϕ_1 and ϕ_2 be two true QBFs with common variables $\mathcal{C}$. We define ϕ_1 solution-subsumes ϕ_2 (written as $\phi_1 \sqsubseteq \phi_2$) iff for all winning strategies S for ϕ_1, also $\phi_2[S/\mathcal{C}]$ is true.*

We can only test $\phi_1 \sqsubseteq \phi_2$ for particular instances of S. To show the subsumption of ϕ_1 to ϕ_2, we need to show that a given strategy S of ϕ_1 is also a strategy for ϕ_2. Recall that a strategy is simply a function from universal variables to existential variables. As long as the same function works for ϕ_2, the subsumption relation is not refuted.

To show that, we first rewrite S to S' to avoid common name conflicts with ϕ_2. While rewriting S to S', we leave the common variables untouched (which are part of the winning strategy we consider). For example, in the game instances in the case study 5, we do not rewrite black player and white player moves. Finally, we create a new formula $S' \wedge \phi_2$. We claim that the new formula is True iff S' is a strategy for ϕ_2. Otherwise, checking this single S is sufficient to refute solution-subsumption, which is useful for bug detection. The S' formula essentially forces the existential variables given values to the universal variables. Since the QBF solver has to satisfy both S' and ϕ_2 which share common variables, the assignments to the common variables are always the same. In our game instances, the black moves for the opponent's white moves are forced by the rewritten strategy S'.

While for true formulas, the (possibly modified) winning strategy is just conjunctively added to the matrix, for false formulas it is additionally necessary to change the quantifier type of the variables defined by the winning strategy to existential. If the formula remains false, then the winning strategy of the first formula is also a winning strategy of the second formula. Since checking for common solutions has similar memory problems as the winning strategies for the interactive play presented in Section 3, we also support checking solution-subsumption with partial winning strategies. Our tool is completely agnostic to the completeness of a winning strategy. Examples are shown in the next section.

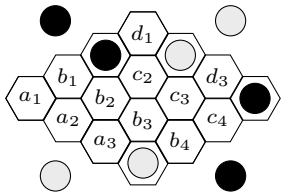
5 Case Study

We provide an open source implementation of validation and winning strategy equivalence. All benchmarks and data are available online.[1] To obtain a winning strategy, our tool SQval (Scalable QBF Validator) first generates a proof trace in QRP-format using the solver DepQBF [10]. Then it extracts the winning strategy using the QRPcert framework [11]. For interactive plays, it uses solver DepQBF for QDIMACS instances and solver Quabs [8] for QCIR instances. All computations for the experiments are run on a cluster.[2]

As a case study, we conducted an experiment with two QBF encodings for positional games. In particular, we compare two encodings for the game Hex: Lifted Neighbour-Based (LN) and Stateless Neighbour-based (SN) in [15]. In the Hex game, players (black, white) take turns to occupy empty positions on a NxN board with hexagonal cells. The player who connects appropriate opposite borders with a path of pegs of their own color wins the game. To demonstrate validation and equivalence checking, we first consider a small Hex

[1] `https://github.com/irfansha/SQval`

[2] `http://www.cscaa.dk/grendel-s`, each problem uses one core on a Huawei FusionServer Pro V1288H V5 server, with 384 GB main memory and 48 cores of 3.0 GHz (Intel Xeon Gold 6248R).



(a) Hein puzzle 12, before Hex preprocessing.

```
e 1 2 3 0
a 4 5 6 0
e 7 8 9 0
a 10 11 12 0
e 13 14 15 0
a 16 17 18 0
e 19 20 21 0
```

(b) Move variables in all Hein-12 QBF instances.

Table 1 Number of valid runs/ Number of invalid runs for each instance and each assertion.

	static			dynamic			hybrid		
Inst: / Assert:	GA	LPA	LBA	GA	LPA	LBA	GA	LPA	LBA
LN-Hein-12	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0
SN-Hein-12	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0
SN-R-Hein-12	100/0	100/0	91/9	100/0	100/0	84/16	100/0	100/0	84/16

instance, Hein-12 (see Figure 2a), due to Piet Hein [7]. We first preprocess the instance as in [15], resulting in 8 open positions. Using the preprocessed instance, we generate three QBF instances for a winning strategy of depth 7:

- LN-Hein-12 : Instance generated with LN, both players can only occupy empty positions.
- SN-Hein-12 : Instance generated with SN, black player can only occupy empty positions.
- SN-R-Hein-12 : Instance generated with SN with relaxed constraints for optimization, in which the black player is allowed to occupy previous black positions.

All three QBF instances solve the Hex instance, and their first 7 layers (cf. 2b) correspond to the moves taken, i.e., variables encoding 1 out of max 8 open positions (3 bits) per layer.

5.1 Validating Hein-12 Instances

The Hein-12 puzzle indeed has a winning strategy at depth 7, so all 3 instances are true formulas. For validation, we propose three assertions that are relevant to these encodings.

- Goal-Assertion (GA): “Goal is reached at the end of the play”. For the LN instance, satisfying the goal constraint is specified by the assertion clause “`314 0`”.
- Legal-Play-Assertion (LPA): “Black does not play on white positions”. We generate inequality constraints between black moves and preceding white moves as a CNF.
- Legal-Black-Assertion (LBA): “Black does not play on black positions”. We generate inequality constraints between different black moves as a CNF.

GA and LPA should hold for all instances, whereas LBA should only hold for the LN-Hein-12 and SN-Hein-12 instances. We use all 3 types of QBF validation for checking these 3 assertions on all 3 encodings. We run 100 iterations with a random generator with seeds ranging from 0-99. For hybrid validation, we use partial certificates up to depth 3 and validate the rest with a QBF solver. In Table 1, we present for each case the number of passing/failing runs. Indeed, both static, dynamic, and hybrid validation show some runs revealing the failure of assertion LBA for SN-R-hein-12, while all other tests pass. For Hein-12, all three validation techniques take a few seconds and a few MB for each iteration.

Table 2 Listing the result of the subsumption tests between instances for Q1 $\sqsubseteq$ Q2.

Q1: / Q2:	LN-Hein-12	SN-Hein-12	SN-R-Hein-12
LN-Hein-12	T	T	T
SN-Hein-12	T	T	T
SN-R-Hein-12	F	F	T

5.2 Equivalence Check for Hein-12

Validating GA and LPA increases the confidence in the correctness of the previous encodings. Additionally, we expect that LN-Hein-12 and SN-Hein-12 have the same winning strategies, since in both encodings, the black player can only play on open positions, and the same moves lead to equivalent states. This cannot be checked with our testing or fuzzing approach. Instead, we apply our subsumption check on all combinations of the 3 encodings. Table 2 shows the results from our subsumption check. Indeed, LN-Hein-12 and SN-Hein-12 appear to be equivalent (on the winning strategy returned by the solver). However, we found a strategy for SN-R that is not valid for SN and LN. Indeed, SN-R can play on already occupied black positions, which leads to invalid strategies for the LN and SN encodings. On the other hand, every move played in LN or SN is also valid in SN-R, resulting in subsumption in the other direction. These results are consistent with the intention behind the encodings.

5.3 Validating Larger Hex Instances With Partial Certificates

The Hein-12 instance has only 8 open positions after preprocessing, and we checked only for depth 7. On this small example, all 3 validation approaches worked equally well. Since the certificates remain small, the subsumption checks can be done within a few seconds. However, certificates grow exponentially in size with the number of variables in each layer and alternation depth. To show the difference between the validation strategies, we experiment with a harder Hex instance, Hein-09, which has 10 open positions after preprocessing, and we generate instances with a winning strategy of depth 9.

In Table 3, we observe that the full certificates in AAG format (Ascii And-Inverter Graph) are quite large, in the range of 532.7 MB – 7.8 GB. Note that the partial certificates are much smaller, but increasing with the level. Table 4 shows the resources required for generating the QRP traces for different settings, and for extracting certificates from the traces.

To show the difference between the 3 validation approaches on the harder instance Hein-09, we consider assertion GA for validation. From Table 5, we observe that static validation with full certificates for LN is infeasible. While we can validate the other encodings using full certificates, it takes up to 20 GB. Dynamic validation performs well on SN and SN-R, while it takes 1203 seconds for validating a single iteration of an LN instance. Note that

Table 3 Size in Bytes of (partial) certificates in AAG format, for each encoding instance with increasing levels of partial certificates. QRP trace is the size of the trace generated by solver DepQBF.

Enc: / Cert:	Full	L1	L3	L5	L7	L9	QRP trace
LN-Hein-09	7.8G	108	1.1K	21.8K	434K	8.2M	6.8G
SN-Hein-09	641.6M	96	1.1K	23.4K	437.3K	8.2M	344M
SN-R-Hein-09	532.7M	96	1.2K	24.2K	461.1K	8.7M	394.5M

Table 4 Time and Memory used for generating the QRP trace and extracting certificates from it.

	Peak Memory (MB)				Time Taken (Sec)			
Enc: / Cert:	QRP trace	Full	L3	L9	QRP trace	Full	L3	L9
LN-Hein-09	13470	10390	7580	9750	1165	261	59	233
SN-Hein-09	615.46	1.54	1.54	1.54	54	20	21	15
SN-R-Hein-09	535.9	1.54	1.54	1.54	54	13	4	15

Table 5 Peak Memory (PM) in MB and Time Taken (TT) in seconds for assertion GA (seed 0).

	static		dynamic		hybrid-L3	
Inst:	PM	TT	PM	TT	PM	TT
LN-Hein-09	–	TO	64.1	1203	**1.54**	**1.6**
SN-Hein-09	20.08K	2100	1.54	9.4	**1.53**	**0.4**
SN-R-Hein-09	18.35K	1226	**1.53**	5.4	**1.53**	**0.4**

to run 100 iterations with different seeds, we would need approximately 100*1203 seconds. Hybrid validation performs clearly the best in both time and memory, never exceeding a couple of seconds or 2 MB. Of course, generating a partial certificate via QRP trace is still the bottleneck for hybrid validation. However, we only pay a one-time cost for generating partial certificates, which can be used any number of times for validation or subsumption checks.

We will now experiment with subsumption checking, using full or partial certificates. For Hein-09 the QBF instances appended with a certificate can exceed 15 GB in CNF. Note that, we only need partial certificates with all existential black move variables, i.e., L9 in Table 3. These never exceed 10 MB (in AAG format), so a complete subsumption check with L9 partial certificates only is feasible. We compute subsumption checks on all combinations of 3 encodings, i.e., for each combination we append one QBF with the certificate of the other, and use a QBF solver. From Table 6, it is clear that subsumption with full certificates blows up, often exceeding 50 GB of memory. In fact, we could not generate the appended instance with LN certificates since the instances themselves can exceed 20 GB.

On the other hand, with L9 certificates, we could check non-subsumption for SN-R with SN and LN, taking 446 and 84 seconds, respectively. DepQBF runs out of time when trying to prove the subsumption cases, but never uses more than 2 GB for solving with L9 certificates. Intuitively, proving non-subsumption is indeed easier than proving subsumption. One could try to use preprocessors, to speed up the subsumption checks for L9 certificates, but full certificate subsumption would still be out of reach.

Table 6 Peak Memory in GB during subsumption checks between Hein-09 instances.

	LN-Hein-09		SN-Hein-09		SN-R-Hein-09	
Inst:	Full	L9	Full	L9	Full	L9
LN-Hein-09	-	1.53	-	1.57	-	1.57
SN-Hein-09	63.7	1.53	63.7	1.6	63.7	1.6
SN-R-Hein-09	50.74	1.6	50.78	1.6	50.8	1.69

6 Conclusion and Future Work

In this paper, we proposed validation techniques with winning strategies and interactive play with a QBF solver. For scalable validation, we proposed using partial winning strategies for outer layers and interactive play for deeper layers. We extended the idea of validation to solution-equivalence of encodings that have some common winning strategy. To evaluate various techniques proposed, we conducted a case study on 2-player game encodings for the game Hex. We showed that with the use of winning strategies, one can increase the confidence of encoding correctness. In the most scalable approach, generating QRP traces remains the bottleneck, as solvers generate complete traces. While checking solver correctness requires complete traces, partial traces/certificates are sufficient for encoding validation. One future research direction would be to allow QBF solvers to generate partial traces efficiently.

References

1 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.

2 Marco Benedetti. Extracting certificates from quantified boolean formulas. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 47–53. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0985.pdf.

3 Olaf Beyersdorff, Mikolás Janota, Florian Lonsing, and Martina Seidl. Quantified Boolean Formulas. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1177–1221. IOS Press, 2021.

4 Roderick Bloem, Vedad Hadzic, Ankit Shukla, and Martina Seidl. Ferpmodels: A certification framework for expansion-based qbf solving. In *Proc. of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) 2022*, September 2022.

5 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proc. of the 13th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010.

6 Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 546–553. IJCAI/AAAI, 2011.

7 Ryan B. Hayward and Bjarne Toft. *Hex, the full story.* AK Peters/CRC Press/Taylor Francis, 2019.

8 Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In *Proc. of the 9th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF)*, volume 277 of *EPTCS*, pages 88–102, 2018. doi:10.4204/EPTCS.277.7.

9 Charles Jordan, Will Klieber, and Martina Seidl. Non-CNF QBF solving with QCIR. In *AAAI-16 Workshop on Beyond NP*, February 2016.

10 Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *CADE*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017. doi:10.1007/978-3-319-63046-5_23.

11 Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF. In *Proc. of the 15th Int. Conf. on the Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer, 2012. doi:10.1007/978-3-642-31612-8_33.

12 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Polynomial-time validation of QCDCL certificates. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. of the 21st Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2018)*, volume 10929 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018. doi:10.1007/978-3-319-94144-8_16.

13 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (qbfeval'16 and qbfeval'17). *Artif. Intell.*, 274:224–248, 2019. doi:10.1016/j.artint.2019.04.002.

14 Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *FMCAD*, pages 136–143. IEEE, 2015. URL: https://www.react.uni-saarland.de/publications/RT15.pdf.

15 Irfansha Shaik, Valentin Mayer-Eichberger, Jaco van de Pol, and Abdallah Saffidine. Implicit state and goals in QBF encodings for positional games (extended version). *CoRR*, abs/2301.07345, 2023. doi:10.48550/arXiv.2301.07345.

16 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A Survey on Applications of Quantified Boolean Formulas. In *Proc. of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84. IEEE, 2019.