Journal on Satisfiability, Boolean Modeling, and Computation 11 (2019) 211-220

QBFRelay, QRATPre+, and DepQBF: Incremental Preprocessing Meets Search-Based QBF Solving

Florian Lonsing[*] ⟨lastname⟩@cs.stanford.edu
Computer Science Department
Stanford University
Stanford, CA 94305, USA

Abstract

DepQBF is a search-based quantified Boolean formula (QBF) solver implementing the QCDCL paradigm. We submitted DepQBF as part of several tool packages to the QBFEVAL'18 competition, which was part of the FLoC Olympic Games 2018. These packages integrate DepQBF as a back end QBF solver and a preprocessing front end called QBFRelay. This front end consists of a shell script that runs several preprocessors in multiple rounds on a given QBF, thus resulting in incremental preprocessing. QBFRelay employs publicly available preprocessors developed by the QBF community and, additionally, our novel preprocessor QRATPre+ that is based on a generalization of the QRAT proof system. We present an overview of DepQBF, QRATPre+, and QBFRelay and report on the performance of our submissions, which were awarded a medal in the FLoC Olympic Games.

KEYWORDS: *QBF solving, Q-resolution, QCDCL, preprocessing, QRAT proof system.*

Submitted November 2018; revised April 2019; published September 2019

1. Introduction

The satisfiability problem of quantified Boolean formulas (QBFs) is the canonical PSPACE-complete problem and has many practically relevant applications in model checking and formal verification, for example. For practical applications, efficient QBF reasoning tools are highly requested. Unlike in propositional satisfiability (SAT), where conflict-driven clause learning (CDCL) [34] is the single dominating solving paradigm, several successful paradigms have emerged in the context of QBF. Moreover, it was empirically observed that preprocessing can have a highly positive impact on solving performance when coupled with certain paradigms, and a highly negative one when coupled with others [27, 28].

QBFEVAL'18[1.] is the 2018 edition in a series of competitive QBF evaluations with the goal to drive the development of QBF reasoning tools. We describe the tool packages we submitted to QBFEVAL'18, which include our search-based QBF solver DepQBF [21, 23], the preprocessing module QBFRelay, and the preprocessor QRATPre+ [24, 25]. QBFRelay is a front end used to apply effective preprocessing by running several preprocessors iteratively and incrementally on the given QBF. The preprocessed formula is forwarded to the back end solver DepQBF. The rationale behind combining DepQBF and QBFRelay is that preprocessing

∗ Part of this work was carried out while the author was employed at the Institute of Logic and Computation, TU Wien, Austria, and supported by the Austrian Science Fund (FWF) under grant S11409-N23.

1. http://www.qbflib.org/qbfeval18.php

was found to be beneficial for search-based solvers in many cases. We apply the preprocessor QRATPre+ in QBFRelay because it was the first available implementation of preprocessing based on the QRAT proof system [11] and on its generalization QRAT$^+$ [24]. As such, QRATPre+ contributed functionality to the preprocessing module that was unique compared to the other tools used in QBFRelay. Our submissions were awarded a medal in the FLoC Olympic Games,[2.] which demonstrates their competitiveness with state-of-the-art solvers.

2. Overview

In the following, we first present an overview of DepQBF, QRATPre+, and QBFRelay as the building blocks of the packages we submitted to QBFEVAL'18. Then we provide implementation details and report on experimental results.

Search-Based QBF Solving. DepQBF[3.] is a search-based solver [8] with conflict-driven clause and solution-driven cube learning (QCDCL) [9, 18, 39]. It takes as input QBFs in *prenex CNF (PCNF)*, which consist of a quantifier prefix and a quantifier-free CNF over propositional variables. QCDCL generalizes the successful CDCL paradigm from SAT to QBF. Similar to a CDCL SAT solver, a QCDCL QBF solver derives new learned clauses during the search process. The clauses that participate in the derivation of a new learned clause are selected depending on the current search context. The derivation of a learned clause relies on the Q-resolution calculus [17] and its variants, e.g. long-distance Q-resolution [38]. Q-resolution differs from propositional resolution as used in CDCL in the additional *universal reduction rule*, which allows the deletion of certain universal literals in clauses. In a dual way to learning clauses, learned cubes, i.e., conjunctions of literals, are derived in search contexts that correspond to satisfiable subcases. The given PCNF to be solved is augmented by the learned clauses and cubes with the purpose to guide backtracking and the future search process of the solver.

Since its initial release [21], DepQBF integrates the standard dependency scheme [33]. In general, a dependency scheme allows to exploit independence of certain variables in order to strengthen the universal reduction rule in Q-resolution. Exploiting independence by dependency schemes and related concepts was explored in practical work on solving [29] and in theoretical work on QBF proof complexity [6].

The variant of DepQBF we submitted to QBFEVAL'18 is based on version 6.03 [23]. Compared to previous versions, version 6.03 comes with an advanced technique for cube learning [19] by tightly integrating blocked clause elimination [5, 10] into the QCDCL workflow. Thereby, the goal is to dynamically and temporarily remove clauses that are redundant in the current search context. This may allow to learn shorter cubes earlier.

Additionally, version 6.03 integrates the SAT solver PicoSAT [4] (version 960[4.]) and the QBF solver Nenofex [20] (version 1.1[5.]). Nenofex is an expansion-based QBF solver. Expansion [1, 3, 12] and Q-resolution have different strengths, i.e., regarding QBF proof complexity these approaches are orthogonal to each other. PicoSAT and Nenofex are applied dynamically in QCDCL as resource-bounded, incomplete QBF oracles. These oracles are

2. https://www.floc2018.org/floc-olympic-games/
3. http://lonsing.github.io/depqbf/
4. http://fmv.jku.at/picosat/
5. https://github.com/lonsing/nenofex

used to solve the QBF within the current search context. Depending on the results returned by the oracles, new clauses and cubes can potentially be derived and used as ordinary learned clauses and cubes [26]. Due to the integration of expansion in QCDCL via Nenofex, the solver potentially can produce proofs that are exponentially shorter than without expansion.

The variant of DepQBF we submitted to QBFEVAL'18 differs from version 6.03. The submitted one comes with a built-in module for limited preprocessing based on detecting clausal patterns that result from Tseitin translation [35]. By identifying such patterns, it is possible to reverse-engineer (parts of) the circuit structure of a given PCNF. We use reconstructed circuit information only to shift existential Tseitin definition variables in the prefix of the PCNF, if possible [2]. Apart from shifting variables, circuit information is not exploited during solving.

Generalizing the QRAT Proof System for Preprocessing. QRATPre+ [25] is a QBF preprocessor that implements redundancy removal based on a generalization of the QRAT proof system [11], called QRAT$^+$ [24]. The QRAT proof system simulates virtually all techniques applied in QBF reasoning tools, including expansion and long-distance resolution [14], and allows for a complete workflow of preprocessing, solving, and certifying QBFs [7]. In the following, we present the high-level idea of preprocessing based on QRAT and QRAT$^+$ and refer to a related tool paper [25] for implementation details of QRATPre+.

Preprocessing based on QRAT and QRAT$^+$ aims at eliminating redundant clauses and universal literals from the PCNF. Informally, to this end the clauses C' in the resolution neighborhood of a given clause C are inspected, cf. [13, 15, 16]. The resolution neighborhood is the set of all clauses that can potentially be resolved with C on a literal $l \in C$. If all the potential resolvents of C and C' under certain conditions and restrictions are redundant, then either clause C or literal $l \in C$ is redundant, depending on whether the variable of l is existential or universal, respectively. Thereby, redundancy of the potential resolvents is checked by tentatively adding the negated resolvent to the formula and trying to derive a conflict using QBF unit propagation. This approach requires to apply QBF unit propagation to abstractions of the formula where certain universal quantifiers are converted to existential ones. The QRAT$^+$ proof system generalizes QRAT by exploiting the quantifier prefix when checking resolvents in the resolution neighborhood. In our preprocessor QRATPre+,[6.] we implemented resource-bounded preprocessing based on QRAT$^+$ to heuristically eliminate redundant clauses and universal literals.

Incremental Preprocessing. QBFRelay[7.] is a shell script used to coordinate several preprocessors. It runs the preprocessors QxBF version 1.2 [22],[8.] Bloqqer version 37 [5, 10],[9.] HQSpre version 1.3 [37],[10.] and QRATPre+ version 1.0 with a time limit in sequences and in multiple rounds on a given QBF in prenex CNF. We use runsolver [32][11.] to control resources. The formula produced by one preprocessor is used as input for the next preprocessor in an execution sequence. Preprocessing stops as soon as the formula is solved or does not change

6. https://lonsing.github.io/qratpreplus/
7. https://github.com/lonsing/qbfrelay
8. http://fmv.jku.at/qxbf/
9. http://fmv.jku.at/bloqqer/
10. https://projects.informatik.uni-freiburg.de/projects/dqbf/files
11. http://www.cril.univ-artois.fr/~roussel/runsolver/

anymore. QBFRelay is inspired by promising experimental results related to incremental preprocessing [27]. Although the techniques applied by the preprocessors in QBFRelay overlap to some extent, diversity may result from different schedulings and implementations.

Bloqqer and HQSpre implement techniques like expansion of universal variables, blocked clause elimination with optional addition of hidden or covered literals, blocked literal elimination, and self-subsuming resolution. Techniques applied only in HQSpre are blocked literal addition, gate substitutions, SAT-based redundancy removal, and the use of dependency schemes to optimize universal variable expansion. QxBF is unique among the considered tools in QBFRelay in that it applies failed literal detection based on quantifier abstractions where certain universal quantifiers are converted to existential ones. This kind of abstractions are a restricted form of those applied in QRATPre+. The application of redundancy removal based on QRAT and QRAT$^+$ in QRATPre+ is unique with respect to the other preprocessors. Some techniques such as blocked clause/literal elimination that are implemented in Bloqqer and HQSpre are subsumed by the elimination techniques implemented in QRATPre+.

QBFRelay can be configured via command line options to adapt the sets of called preprocessors, their execution ordering, and timeouts. It can also easily be extended by integrating further preprocessors. To this end, the preprocessor to be integrated has to satisfy the simple requirement that it terminates with an exit code of zero to indicate normal termination and with exit codes 10 and 20 to indicate that the formula was solved and found satisfiable or unsatisfiable, respectively (similar to the QBFEVAL competitions). The actual integration amounts to implementing a new function in the script for calling the preprocessor based on the pattern of the functions where the already integrated preprocessors are called.

3. Submitted Tool Packages and Experiments

We submitted the following tool packages to QBFEVAL'18. The package DepQBF-prefix-opt participated in the 2QBF and PCNF track and consists of the standalone solver DepQBF without preprocessing except limited reverse-engineering of the formula structure to optimize the quantifier prefix. We did not apply additional preprocessing in DepQBF-prefix-opt to allow for the straightforward generation of partial QDIMACS output certificates.[12.]

The package DepQBF-QxQBH consists of a combination of QBFRelay and DepQBF-prefix-opt, where the letter code "QxQBH" indicates the use of the preprocessors QxBF ("Qx"), QRATPre+ ("Q"), Bloqqer ("B"), and HQSpre ("H") in QBFRelay. We implemented a time-slicing approach in DepQBF-QxQBH to control the time spent on preprocessing and solving. In QBFEVAL'18 the time and memory limits were set to 900s and 32 GB. We determined the scheduling of the tools based on the results of many experiments using only one or at most two preprocessors in different orderings. This way, we obtained a basic picture of the impact of different orderings on the performance. However, it is important to note that such empirical approach of course largely depends on the underlying benchmark sets. There is no guarantee that the scheduling implemented in DepQBF-QxQBH performs best on any benchmark set. The scheduling consists of the following stages.

1. Filter instances using QBFRelay. To this end, we run two variants of QBFRelay on an instance. In one variant, first HQSpre and then Bloqqer is applied, and in the second

12. http://www.qbflib.org/qdimacs.html

variant that execution ordering is reversed. In each of these variants, we run at most two rounds. The total time limit for this stage was 60s. If the instance is solved already in this stage, then the result is returned immediately. Otherwise, the preprocessed formula is discarded.

2. Spend at most 60s to solve the original PCNF using DepQBF.

3. Optimize the quantifier prefix by limited reverse-engineering of the formula structure. In the following, the resulting formula is called the "prefix-optimized instance".

4. Spend at most 100s to filter the prefix-optimized instance by QBFRelay. Thereby, the same configurations of QBFRelay are used as in stage 1. As above, if the instance is solved, then the result is returned. Otherwise, the preprocessed formula is discarded.

5. Spend at most 60s to solve the prefix-optimized instance by DepQBF.

6. Preprocess the prefix-optimized instances for at most 250s using QBFRelay. To this end, the preprocessors QxBF, QRATPre+, Bloqqer, and HQSpre are applied in that ordering for at most two rounds and with a time limit of 60s for each call of an individual preprocessor. If QBFRelay exceeds the resource limits then we run DepQBF on the prefix-optimized instance for the remaining amount of time. Otherwise, we run DepQBF on the preprocessed formula produced by QBFRelay, unless the prefix-optimized instance was solved already by QBFRelay.

3.1 Experiments

We present experiments to highlight the performance or our submissions to QBFEVAL'18. For a comprehensive comparison of our tools to state-of-the-art solvers, we refer to the official competition results and instead illustrate the impact of incremental preprocessing and of our novel preprocessor QRATPre+ by supplementary experiments.

Official QBFEVAL'18 Results. The package DepQBF-QxQBH was ranked first in the 2QBF track, winning by a small margin, and second in the PCNF and hard instances tracks, respectively. In the latter track, only three solvers participated, and we used a different time-slicing strategy in DepQBF-QxQBH than in the PCNF track. DepQBF-QxQBH was awarded a medal in the FLoC Olympic Games.

Official QBFEVAL'19 Results. We submitted the same version of DepQBF-QxQBH that we submitted to QBFEVAL'18 also to the 2019 edition of QBFEVAL,[13.] where it was ranked second in the PCNF track.

Supplementary Results. The following experiments were carried out on a cluster of Intel Xeon CPUs (E5-2650v4, 2.20 GHz) running Ubuntu 16.04.1. We used a limit of 1800s CPU time and 7 GB of memory, which is different from the setup of QBFEVAL'18. Table 1 shows the numbers of instances in the PCNF track of QBFEVAL'18 (463 instances) that were solved in the stages of the time-slicing approach implemented in DepQBF-QxQBH. Table 2a shows results obtained by various configurations of our workflow, where single preprocessors

13. http://www.qbflib.org/qbfeval19.php

Table 1. PCNFs solved in stages by the tools in DepQBF-QxQBH and in total ("Σ").

Stages	1	2	3	4	5	6	Σ
Solved by HQSpre	141	–	–	16	–	8	165
Solved by Bloqqer	33	–	–	5	–	9	47
Solved by DepQBF	–	44	–	–	2	30	76
Total Solved	174	44	–	21	2	47	288

Table 2. Performance results on PCNF and 2QBF instances. See also Figures 1a and 1b.

(a) PCNF (463 instances).

Solver	*S*	⊥	⊤	*Time*
DepQBF-QBH	289	167	122	342K
DepQBF-QxQBH	288	169	119	346K
DepQBF-QxBH	278	162	116	360K
DepQBF-QxQH	256	156	100	395K
DepQBF-QxQB	235	123	112	443K
DepQBF-prefix-opt	154	96	58	580K
DepQBF-6.03	148	87	61	593K

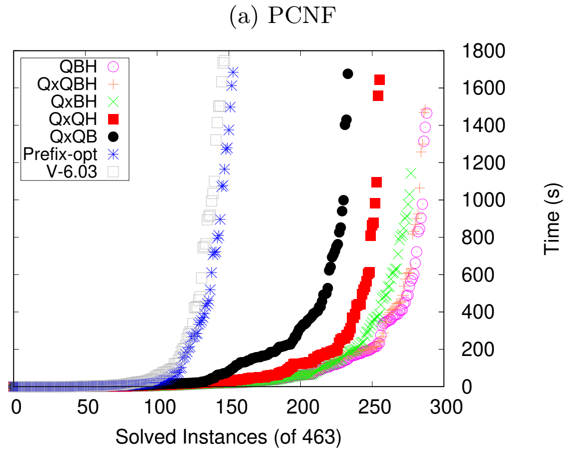
(b) 2QBF (402 instances).

Solver	*S*	⊥	⊤	*Time*
CADET-2.3.1	269	123	146	256K
DepQBF-QxQBH	268	108	160	264K
DepQBF-QBH	265	109	156	267K
DepQBF-QxQB	255	100	155	293K
DepQBF-QxBH	244	107	137	307K
DepQBF-QxQH	194	90	104	396K
DepQBF-6.03	81	64	17	593K

are omitted in all stages where applicable. When omitting preprocessor, we still allow 250s for the remaining tools in QBFRelay in stage 6. Moreover, the scheduling and other time limits remain the same as presented above.

In Table 1, we analyze the instances solved by variant DepQBF-QxQBH and not by the best-performing one (DepQBF-QBH in Table 2a) since the former applies all considered preprocessors. In the workflow, instances were solved by Bloqqer and HQSpre, but not by QxBF and QRATPre+. This is not surprising since QxBF and QRATPre+ implement a much more limited set of techniques than Bloqqer and HQSpre and, if at all, are able to solve either only unsatisfiable or only satisfiable instances, respectively. Every stage except stage 3, where no solving attempt is made, contributes solved instances. The majority of instances (60%) is solved already in the first stage, where we used a small timeout of only 60s. These results illustrate the power of preprocessing as resource-bounded solving, cf. [27].

We evaluate the impact of incremental preprocessing by omitting every single preprocessor in the workflow of QBFRelay. Table 2 shows related statistics for the PCNF (2a, 463 instances) and the 2QBF track (2b, 402 instances), and Figure 1 shows the respective cactus plots. As a baseline, we included the officially released version 6.03 of DepQBF in the experiments. The difference in solved instances between DepQBF with and without the variants of incremental preprocessing by QBFRelay is substantial on both PCNF and 2QBF instances. Optimizing the quantifier prefix of PCNF instances based on reverse-engineering the circuit structure as done in DepQBF-prefix-opt is beneficial compared to plain DepQBF-6.03. On PCNF instances, the variant DepQBF-QBH without QxBF outperforms variant DepQBF-QxQBH including all preprocessors by one instance. However, that latter variant performs best among all our



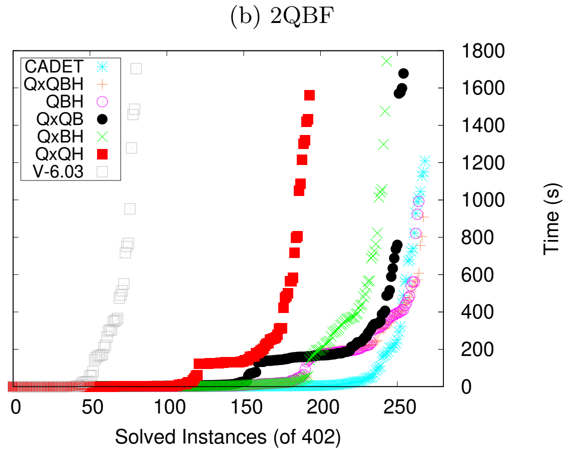


Figure 1. Cactus plots related to Tables 2a and 2b.

variants on 2QBF instances. The use of Bloqqer and HQSpre in DepQBF-QxBH is crucial for performance since excluding any of these tools decreases the number of solved instances substantially. In addition to Bloqqer and HQSpre, QRATPre+ is necessary to achieve top performance. When excluding it from preprocessing as in variant DepQBF-QxBH, then considerably fewer PCNF and 2QBF instances are solved.

Incremental preprocessing by QBFRelay outperforms the application of the single state-of-the-art preprocessors Bloqqer and HQSpre. In a related experiment not shown in the tables we ran only Bloqqer or HQSpre, respectively, on the PCNF instances with a timeout of 600s. With an additional timeout of 1800s for solving, DepQBF solved 198 instances preprocessed by Bloqqer and 270 instances preprocessed by HQSpre. These results emphasize the benefits of incremental preprocessing compared to single preprocessors, given that we allowed 1800s for preprocessing *and* solving for the results reported in the tables above. We refer to related work [25] for an additional evaluation of preprocessing using combinations of Bloqqer, HQSpre, and QRATPre+.

On 2QBF instances, we omitted DepQBF-prefix-opt in the evaluation because prefix optimization does not have any effects there. Instead, we considered the state-of-the-art 2QBF solver CADET-2.3.1 [30, 31], which was ranked second after DepQBF-QxQBH in QBFEVAL'18. The results of CADET-2.3.1 are particularly remarkable given that it does not use preprocessing. The ranking in Table 2b differs from the official competition ranking as CADET-2.3.1 is ranked first and DepQBF-QxQBH second. The competition was run in a different computing environment. However, our results show that incremental preprocessing has a dramatic effect on the performance. The number of instances solved by variant DepQBF-QxQBH is more than three times larger than the one solved by DepQBF-6.03. Moreover, like on PCNF instances, the use of QRATPre+ is necessary in addition to Bloqqer and HQSpre to achieve top performance. Without QRATPre+, DepQBF-QxBH solves 24 instances less than DepQBF-QxQBH. The difference between DepQBF-QxQBH and DepQBF-QxBH is due to instance families where QRATPre+ is able to eliminate a considerable number of universal literals from the formulas.

4. Conclusion

We presented our submissions to QBFEVAL'18, which are combinations of search-based solving using our QCDCL solver DepQBF and incremental preprocessing using the preprocessing script QBFRelay. The iterative and incremental application of several preprocessors with different characteristics and employing different techniques in QBFRelay turned out to be crucial to achieve top performance. In our workflow, the majority of instances was solved already by incremental preprocessing using a small timeout of only one minute CPU time.

In the context of QCDCL solving, the official QBFEVAL'18 competition results and our experiments demonstrate the impact of incremental preprocessing and of preprocessing based on the QRAT and QRAT$^+$ proof systems implemented in QRATPre+. As future work, it is interesting to evaluate the impact of these approaches on solving paradigms other than QCDCL, e.g., expansion or incremental determinization [30] as applied in CADET-2.3.1.

In its nature, the time-slicing approach we implemented in DepQBF-QxQBH is similar to a very simple, sequential portfolio. It is effective in combining several, different techniques within a given timeout. However, our approach does not allow for a straightforward extraction of proofs and certificates, unlike, e.g., incremental determinization in the context 2QBF. Hence the lack of certification naturally limits the practical applicability of time-slicing, cf. [36]. This limitation can be overcome by an integrated solving and certification workflow that seamlessly integrates the individual techniques applied in the time-slicing approach, cf. [7].

References

[1] Abdelwaheb Ayari and David A. Basin. QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers. In *FMCAD*, **2517** of *LNCS*, pages 187–201. Springer, 2002.

[2] Olaf Beyersdorff, Leroy Chew, and Mikolas Janota. Extension Variables in QBF Resolution. In *Beyond NP Workshop*, **WS-16-05** of *AAAI Workshops*. AAAI Press, 2016.

[3] Armin Biere. Resolve and Expand. In *SAT*, **3542** of *LNCS*, pages 59–70. Springer, 2004.

[4] Armin Biere. PicoSAT Essentials. *JSAT*, **4**(2-4):75–97, 2008.

[5] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked Clause Elimination for QBF. In *CADE*, **6803** of *LNCS*, pages 101–115. Springer, 2011.

[6] Joshua Blinkhorn and Olaf Beyersdorff. Shortening QBF Proofs with Dependency Schemes. In *SAT*, **10491** of *LNCS*, pages 263–280. Springer, 2017.

[7] Katalin Fazekas, Marijn Heule, Martina Seidl, and Armin Biere. Skolem Function Continuation for Quantified Boolean Formulas. In *TAP*, **10375** of *LNCS*, pages 129–138. Springer, 2017.

[8] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with Quantified Boolean Formulas. In *Handbook of Satisfiability*, **185** of *FAIA*, pages 761–780. IOS Press, 2009.

[9] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. *JAIR*, **26**:371–416, 2006.

[10] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause Elimination for SAT and QSAT. *JAIR*, **53**:127–168, 2015.

[11] Marijn Heule, Martina Seidl, and Armin Biere. Solution Validation and Extraction for QBF Preprocessing. *J. Autom. Reasoning*, **58**(1):97–125, 2017.

[12] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, **234**:1–25, 2016.

[13] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing Rules. In *IJCAR*, **7364** of *LNCS*, pages 355–370. Springer, 2012.

[14] Benjamin Kiesl, Marijn Heule, and Martina Seidl. A Little Blocked Literal Goes a Long Way. In *SAT*, **10491** of *LNCS*, pages 281–297. Springer, 2017.

[15] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Super-Blocked Clauses. In *IJCAR*, **9706** of *LNCS*, pages 45–61. Springer, 2016.

[16] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Blockedness in Propositional Logic: Are You Satisfied With Your Neighborhood? In *IJCAI*, pages 4884–4888. ijcai.org, 2017.

[17] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for Quantified Boolean Formulas. *Inf. Comput.*, **117**(1):12–18, 1995.

[18] Reinhold Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *TABLEAUX*, **2381** of *LNCS*, pages 160–175. Springer, 2002.

[19] Florian Lonsing, Fahiem Bacchus, Armin Biere, Uwe Egly, and Martina Seidl. Enhancing Search-Based QBF Solving by Dynamic Blocked Clause Elimination. In *LPAR*, **9450** of *LNCS*, pages 418–433. Springer, 2015.

[20] Florian Lonsing and Armin Biere. Nenofex: Expanding NNF for QBF Solving. In *SAT*, **4996** of *LNCS*, pages 196–210. Springer, 2008.

[21] Florian Lonsing and Armin Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, **7**(2-3):71–76, 2010.

[22] Florian Lonsing and Armin Biere. Failed Literal Detection for QBF. In *SAT*, **6695** of *LNCS*, pages 259–272. Springer, 2011.

[23] Florian Lonsing and Uwe Egly. DepQBF 6.0: A Search-Based QBF Solver Beyond Traditional QCDCL. In *CADE*, **10395** of *LNCS*, pages 371–384. Springer, 2017.

[24] Florian Lonsing and Uwe Egly. QRAT^{+}: Generalizing QRAT by a More Powerful QBF Redundancy Property. In *IJCAR*, **10900** of *LNCS*, pages 161–177. Springer, 2018.

[25] Florian Lonsing and Uwe Egly. QRATPre+: Effective QBF Preprocessing via Strong Redundancy Properties. In *SAT*, **11628** of *LNCS*, pages 203–210. Springer, 2019.

[26] Florian Lonsing, Uwe Egly, and Martina Seidl. Q-Resolution with Generalized Axioms. In *SAT*, **9710** of *LNCS*, pages 435–452. Springer, 2016.

[27] Florian Lonsing, Martina Seidl, and Allen Van Gelder. The QBF Gallery: Behind the scenes. *Artif. Intell.*, **237**:92–114, 2016.

[28] Paolo Marin, Massimo Narizzano, Luca Pulina, Armando Tacchella, and Enrico Giunchiglia. Twelve Years of QBF Evaluations: QSAT Is PSPACE-Hard and It Shows. *Fundam. Inform.*, **149**(1-2):133–158, 2016.

[29] Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency Learning for QBF. In *SAT*, **10491** of *LNCS*, pages 298–313. Springer, 2017.

[30] Markus N. Rabe and Sanjit A. Seshia. Incremental Determinization. In *SAT*, **9710** of *LNCS*, pages 375–392. Springer, 2016.

[31] Markus N. Rabe, Leander Tentrup, Cameron Rasmussen, and Sanjit A. Seshia. Understanding and Extending Incremental Determinization for 2QBF. In *CAV*, **10982** of *LNCS*, pages 256–274. Springer, 2018.

[32] Olivier Roussel. Controlling a Solver Execution with the runsolver Tool. *JSAT*, **7**(4):139–144, 2011.

[33] Marko Samer and Stefan Szeider. Backdoor Sets of Quantified Boolean Formulas. *JAR*, **42**(1):77–97, 2009.

[34] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, **185** of *FAIA*, pages 131–153. IOS Press, 2009.

[35] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic*, 1968.

[36] Christoph Weidenbach. Do Portfolio Solvers Harm? In *ARCADE*, **51** of *EPiC Series in Computing*, pages 76–81. EasyChair, 2017.

[37] Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. HQSpre - An Effective Preprocessor for QBF and DQBF. In *TACAS*, **10205** of *LNCS*, pages 373–390. Springer, 2017.

[38] Lintao Zhang and Sharad Malik. Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. In *ICCAD*, pages 442–449. ACM / IEEE Computer Society, 2002.

[39] Lintao Zhang and Sharad Malik. Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In *CP*, **2470** of *LNCS*, pages 200–215. Springer, 2002.