Journal on Satisfiability, Boolean Modeling and Computation 9 (2016) 187-206

QBF Gallery 2014: The QBF Competition at the FLoC 2014 Olympic Games

Mikoláš Janota mikolas.janota@gmail.com
SAT Group
INESC-ID Lisboa
Portugal

Charles Jordan skip@ist.hokudai.ac.jp
Graduate School of Information Science and Technology
Hokkaido University
Japan

Will Klieber wklieber@cs.cmu.edu
Software Engineering Institute
Carnegie Mellon University
United States

Florian Lonsing florian.lonsing@tuwien.ac.at
Institute of Information Systems
TU Vienna
Austria

Martina Seidl martina.seidl@jku.at
Institute for Formal Models and Verification
JKU Linz
Austria

Allen Van Gelder http://www.cse.ucsc.edu/~avg
Jack Baskin School of Engineering
University of California at Santa Cruz
United States

Abstract

The QBF Gallery 2014 was a competitive evaluation for QBF solvers organized as part of the FLoC 2014 Olympic Games during the Vienna Summer of Logic. The QBF Gallery 2014 featured three different tracks on formulas in prenex conjunctive normal form (PCNF) including more than 1200 formulas to be solved. Gold, silver, and bronze track medals were awarded to the solvers that solved the most formulas in each of the three tracks. Additionally, the three participants that were most successful over the complete benchmark set were awarded with Kurt Gödel medals, the official prizes of the FLoC 2014 Olympic Games.

In this paper, we give an overview of the setup and rules of the competition, briefly review the participating solvers, and finally report on the results of the QBF Gallery 2014.

KEYWORDS: *Quantified Boolean formulas, QBF solving, QBF competition*

Submitted December 2014; revised October 2015; published January 2016

1. Introduction

Quantified Boolean formulas (QBF) are an extension of propositional logic with explicit existential and universal quantification over the propositional variables [19]. QBF satisfiability checking is PSPACE-complete, and, therefore, many problems from application domains such as model checking, formal verification, planning, or synthesis can be efficiently encoded in QBF [1]. This motivates the effort to improve QBF solving technology, with the hope that QBF solvers become general purpose PSPACE solvers which can be used as blackboxes for solving PSPACE reasoning tasks. In recent years, considerable progress has been made in QBF solving by many different research groups resulting in a variety of available tools.

As in other research communities which build tools for automated reasoning, the QBF community has a long tradition of organizing competitive events. The goal is to obtain an objective evaluation of the available tools and to help potential users identify the QBF tools that are most suitable for their applications. Further, the competitive nature of such evaluations is an important driving force for the solver developers. This is because competitions acknowledge the important but often-overlooked work spent on implementing, improving and tuning tools. In addition, competitions can produce standard benchmark sets which then serve as a uniform basis for the empirical evaluation of novel approaches presented in future papers. That is, competitions are uniquely valuable in several respects.

In this paper, we report on the organization and results of the QBF Gallery 2014. Since the QBF Gallery 2014 is an evolution of previous competitions, we begin by tracing the history of these events in Section 2. Then, we introduce the participating solvers in Section 3 and give an overview of the considered benchmarks in Section 4, before focusing on the various kinds of experiments and results in Section 5. Finally, we conclude with an outlook on future work.

2. Short History of QBF Competitions

For almost ten years, the QBF competition called QBF Eval[1] was organized either annually or biannually. The details and rules of these events were decided by the organizing committee together with an expert jury, a common process that does not provide for community participation.

The first QBF Gallery was organized in 2013 as a different kind of event. The overall goal, to assess the state of QBF tools (especially solvers), was similar to the previous QBF Eval events. However, the QBF Gallery 2013 was organized as a non-competitive, community-driven event, where tool developers and users made decisions on the organization. They were not only invited to submit their tools and benchmarks, but also to decide which experiments would be performed. The results of these experiments were communicated immediately to the participants, who could then submit updated versions of their tools. See the QBF Gallery 2013 technical report [28] for further details on this event.

The 2014 edition of the QBF Gallery was organized as a part of the FLoC 2014 Olympic Games[2] in the style of a traditional competition. In 2014, there was a strict submission

1. http://www.qbflib.org/qbfeval/
2. http://vsl2014.at/olympics/

deadline and the results were publicly announced at the QBF workshop, the SAT conference and the FLoC 2014 Olympic Games Award ceremony. No intermediate feedback was given. The QBF Gallery 2014 consisted of three tracks and for each track, gold, silver and bronze medals were awarded to the three solvers which successfully solved the most formulas within a given time frame. The three solvers with the best solving performance over all three tracks were also awarded with Kurt Gödel medals, the official prizes of the FLoC 2014 Olympic Games.

3. Participating Solvers

The field of participants of the QBF Gallery 2014 consisted of six solvers in 14 different configurations. In general, different configurations of a given solver differ in preprocessing techniques. Often the solvers use external preprocessors like Bloqqer,[3] QxBF,[4] or the QBF variant of Coprocessor[5] (called Qprocessor) before passing the formula to the actual solver. Four of the six solvers were newly submitted to the QBF Gallery 2014 and are described later in this section.

The developers of the other two solvers (QuBE7.2 and ooq) kindly allowed us to use the versions submitted to the 2013 edition of the QBF Gallery for reference purposes. The solver QuBE7.2 [9] in configuration sqube is a search-based QBF solver employing the preprocessor SqueezeBF [10]. The solver ooq[6] is a search-based solver which recovers structure from the CNF in order to perform dual propagation [13]. It participated in the configurations ooq13, where no structure reconstruction and dual propagation is performed, dual_ooq13, where dual propagation is performed after structure reconstruction, and pre_dual_ooq13, which additionally applies the preprocessor SqueezeBF.

All solvers accept formulas in prenex conjunctive normal form in the QDIMACS format as input. A formula is in *prenex conjunctive normal form* (PCNF) if it is of the form $Q_1x_1Q_2x_2\ldots Q_nx_n.\phi$ for $Q_i \in \{\exists, \forall\}$ and ϕ in conjunctive normal form (CNF). In the rest of this section, we briefly describe the solvers submitted to the QBF Gallery 2014.

DepQBF The search-based QBF solver DepQBF[7] [25] implements conflict-driven clause learning and solution-driven cube learning [38, 24, 11]. Prior to solving, DepQBF analyzes the structure of the input formula and computes the standard dependency scheme D^{std} [33]. During the solving process, the standard dependency scheme D^{std} is used to exploit potential independence of variables. Two variables x and y are independent in a PCNF ψ if their positions in the quantifier prefix can safely be swapped without changing the truth value of ψ. Solving may benefit from independence of variables because the linear ordering of the quantifier prefix of ψ is relaxed.

DepQBF has been equipped with clause learning by long-distance resolution [8, 37]. However, the version submitted to the QBF Gallery 2014 only learns non-tautological clauses by traditional Q-resolution [4] based on lazy QBF pseudo unit propagation (QPUP) [27]. Addi-

3. http://fmv.jku.at/bloqqer/
4. http://fmv.jku.at/qxbf/
5. http://tools.computational-logic.org/content/riss3g.php
6. http://www.cs.utoronto.ca/~alexia/ooq/
7. http://lonsing.github.io/depqbf/

tionally, it comes with an algorithmic optimization of the implementation of cube learning. DepQBF was submitted in the following configurations to the QBF Gallery 2014:

- cbdepqbf: A formula is first simplified by Bloqqer (version 31) [3], before it is forwarded to the preprocessor Qprocessor, and finally solved by DepQBF. Qprocessor, implemented by Norbert Manthey, is a QBF preprocessing tool based on the SAT preprocessor Coprocessor [29].

- xbdepqbf: QxBF [26] (version 1.2) is a preprocessor which implements failed literal detection for QBF. It calls the SAT solver PicoSAT[8] [2] (version 951) as a library. The formula preprocessed by QxBF is forwarded to Bloqqer, and finally handed over to DepQBF.

- depqbf: the plain solver DepQBF as described above without any preprocessing.

GhostQ The solver GhostQ[9] is a DPLL-based solver that uses *ghost variables* to achieve a version of the Tseitin transformation that is symmetric with respect to the existential and universal quantifiers, as described in [22] and [21]. A *dual propagation* technique similar to ghost variables was independently and contemporaneously developed in [12]. GhostQ tries to reverse-engineer its CNF input file into circuit form, from which it obtains a set of clauses and cubes with ghost variables.

GhostQ also has a capability to perform a limited version of the CEGAR learning used by RAReQS (described below). A detailed treatment of this capability can be found in Section 5.4 of [20]. To illustrate the basic idea, consider a QBF of the form $\forall X \exists Y.\phi$, where ϕ is a propositional formula. Let π_{cand} be an assignment to the variables in X such that in the solver's clause/cube database, no clauses are falsified and no cubes are satisfied (even under Boolean constraint propagation (BCP)). Let π_{cex} be a *counterexample* to π_{cand}; i.e., let π_{cex} be an assignment to the variables in Y such that ϕ evaluates to `true` under $\pi_{\text{cand}} \cup \pi_{\text{cex}}$. CEGAR learning produces a set of clauses and cubes such that if they are added to the clause/cube database, then for every assignment π'_{cand} to X for which π_{cex} is a counterexample, π'_{cand} will satisfy a cube (under BCP). GhostQ was submitted in three configurations:

- cghostq: CEGAR learning is enabled.

- bcghostq: Same as cghostq except that first Bloqqer [3] is run on the input file. If Bloqqer sufficiently simplifies the problem, then its output is used. Otherwise its output is discarded and the solver proceeds with the original input file, since the reverse engineering code employed by GhostQ currently cannot handle the output of Bloqqer which removes structural information for simplifying the formula.

- ghostq: Plain solver without CEGAR learning.

8. http://fmv.jku.at/picosat/
9. http://www.cs.cmu.edu/~wklieber/ghostq/

hiqqer The QBF solver hiqqer[10] consists of a `csh` script that invokes two preprocessors, plodder and eqxbf, then passes the resulting file to the complete solver stepqbf. Two versions were entered in the QBF Gallery:

- hiqqer1: This version invokes plodder, then eqxbf, then plodder again, then stepqbf.
- hiqqer3: This version invokes (plodder, eqxbf) three times, then plodder again, then stepqbf. The reason for following eqxbf by plodder is that eqxbf cannot detect that a QBF formula is easily true, whereas plodder has this capability. If preprocessing solves the instance, the script exits with the appropriate return code.

The authors of hiqqer made modifications of publicly available tools. The names were changed slightly to avoid confusion with the official versions of those programs. The solver stepqbf is mostly a version of DepQBF dating from about 2012. This version predates the QPUP [27] and long-distance resolution strategies [8]. The preprocessor eqxbf is an extensive modification of qxbf [26] based on the ideas in [36]. The submitted version has several enhancements that are not yet reported.

The preprocessor plodder is a moderate modification of Bloqqer [3] (flip the first five letters of Bloqqer and it becomes plodder). The modifications attempt to improve the performance of Bloqqer by reducing operational count limits in the public version. To this end, the count limits are tailored based on instance statistics, such as numbers of variables and clauses, as well as a possible user-supplied time budget.

RAReQS The solver RAReQS[11] (Recursive Abstraction Refinement QBF Solver) is a QBF solver that implements the ideas of *counterexample guided abstraction refinement* (CEGAR) for QBF [14]. The initial idea comes from the AReQS algorithm, which solves 2-level QBFs [15, 16] (with prefixes of type $\exists\forall$ and $\forall\exists$). AReQS gradually expands the given formula into its *abstraction*. This is done by choosing assignments to the inner quantifier; these assignments are selected by the counterexample scheme. RAReQS generalizes this algorithm to an arbitrary number of quantification levels by calling AReQS *recursively* (hence the "R" at the beginning of the name). RAReQS is implemented in C++ and a recursive call corresponds to a creation of a new object of the solver class. Each object maintains the abstraction in a child object. It creates another object for verifying that the solution of the abstraction (called the *candidate*) is indeed a solution. The exception to this pattern are the leaf objects, which invoke a SAT solver. The underlying SAT solver used in the implementation is minisat 2.2 [7].

The implementation has several optimizations. In order to avoid repetition of counterexamples, abstractions are maintained from one recursive sub-call to another, similar to the way SAT solvers provide *incremental* interfaces. This incremental approach, however, tends to lead to unwieldy memory consumption and, therefore, it is used only when a formula with less than 4 quantification blocks is to be solved.

Apart from the standard refinement, which strengthens the abstraction, a clause is generated that *blocks* the last failed candidate. *Pure literals* and *unit propagation* are used

10. Scripts and binaries of hiqqer are available at http://www.cse.ucsc.edu/~avg/EFL/. Source files may be obtained for research purposes from its authors.
11. http://sat.inesc-id.pt/~mikolas/sw/areqs/

Table 1. Formula characteristics of the different benchmark sets: number of formulas (#formulas), average variable number (vars), average clause number (clauses), average number of quantifier alternations (alt.), average number of universals/existentials (∃/∀)

name	#formulas	vars*	clauses*	alt.*	∃*	∀*
QBFLib	276	28702	67461	12	13459	789
Preprocessing	243	11481	42778	6	6305	814
bomb	132	3644	238191	3	3629	14
dungeon	107	44509	350115	3	44504	5
reduction finding	104	2336	10895	2	2281	54
qbf-hardness	114	2893	10834	27	2611	123
planning-CTE	147	3297	612209	5	3295	2
sauer-reimer	131	15087	44332	3	14815	271

to simplify the generated subformulas; this must be done carefully to avoid losing possible solutions to abstractions. RAReQS was submitted in two configurations:

- brareqs: First the formula is given to the preprocessor Bloqqer and then it is handed over to RAReQS.
- rareqs: the plain solver RAReQS as described above without any preprocessing.

4. Benchmark Sets

The QBF Gallery 2014 featured three different tracks: (1) the QBFLib track, (2) the Preprocessing track, and (3) the Application track. In total, 1254 different formulas were considered. Table 1 contains the aggregated characteristics of the formulas (for the Application track a more fine-grained characterization w.r.t. formula families is given). 18 formulas have more than 50 quantifier alternations, 5 formulas (all from the Preprocessing track) have no quantifier alternation. The number of variables ranges from 70 to 770K, and the number of clauses ranges from 200 to 5000K. A superset of the formulas has been extensively used in the QBF Gallery 2013 and was available to the solver developers. We decided against removing formulas previously solved by all solvers participating in the QBF Gallery 2013, because these formulas are important for checking the correctness of the solvers. In the following, the benchmarks of the different tracks are discussed.

QBFLib Track The benchmark set of the QBFLib track contains 276 formulas. The formulas originate from a set of 345 formulas extracted from the QBFLib[12.] in the context of the QBF Competition 2012r2[13.], which was an unofficial repetition of the QBF Eval 2012. In contrast to previous benchmark sets, the formulas of this benchmark set are selected in such a manner that (1) no formula family is overrepresented, and (2) formulas which have been solved by most solvers in previous competitions are removed. The resulting sample

12. http://www.qbflib.org
13. http://fmv.jku.at/seidl/qbfeval12r2/

was evaluated extensively in the first edition of the QBF Gallery (see [28] for a technical report) and the experiments showed that this sample is representative for the entire formula collection. As several solvers make use of the preprocessor Bloqqer, we removed the 69 formulas that were directly solved by Bloqqer and used the remaining 276 formulas for the QBFLib track of the QBF Gallery 2014. We decided to remove the formulas solved by Bloqqer because we employed the count of solved formulas as the ranking scheme. If we had kept these formulas then the solvers employing Bloqqer would have been awarded points which were earned by Bloqqer. Note that we did not preprocess these formulas. This benchmark set is rather heterogeneous and covers most formula families collected in the QBFLib—each family contributes less than 10 formulas.

Preprocessing Track The goal of the Preprocessing track is to evaluate the behavior of solvers on extensively preprocessed formulas where much structural information available in a PCNF formula was already exploited by a preprocessor to perform simplifications. The 2013 edition of the QBF Gallery included an experiment where we repeatedly applied the preprocessors Bloqqer, variants of hiqqer without calling a complete solver, and SqueezeBF on the formula set $\mathcal{S}$ of the QBF Competition 2012r2 in two different execution strategies. This results in two different formula sets—we refer to them as $\mathcal{S}'$ and $\mathcal{S}''$. For each formula $f \in \mathcal{S}$, we selected the preprocessed variant randomly either from $\mathcal{S}'$ or $\mathcal{S}''$. Then, with probability one half, we also applied the preprocessor Qprocessor, which was submitted as a new preprocessor to the QBF Gallery 2014. Qprocessor applies a QBF-specific variant of bounded variable addition (BVA) [30] and extended resolution to reduce the number of clauses in a formula. To this end, fresh existentially quantified variables are introduced and added to the rightmost existential quantifier block in the quantifier prefix.

The resulting 243 formulas that were not solved by Bloqqer were then included in the benchmark set of the QBF Gallery 2014 Preprocessing track. The preprocessor Qprocessor was applied on 130 formulas.

Application Track The Application track consists of 735 formulas stemming from recent QBF applications that were submitted to the 2013 edition of the QBF Gallery. These applications include

- reduction finding: instances encoding searches for complexity-theoretic reductions between various decision problems in complexity class NL (nondeterministic logspace), contributed by Charles Jordan and Łukasz Kaiser [6, 17, 18];
- bomb and dungeon: formulas from a planning domain with uncertainty in the initial state, contributed by Martin Kronegger, Andreas Pfandler, and Reinhard Pichler [23];
- planning-CTE: encodings of planning problems, contributed by Michael Cashmore [5];
- sauer-reimer: QBF-based test generation instances, contributed by Paolo Marin [34];
- qbf-hardness: instances from bounded model checking of incomplete designs, contributed by Paolo Marin [31].

5. Results of the QBF Gallery 2014

All experiments of the QBF Gallery 2014 were performed on the StarExec[14.] cluster. It provided not only a reliable infrastructure for running the experiments but also an archive for storing the results of this competition as a reference for future experiments.

In the following, we report the details of the conducted solver evaluations. First, we recapitulate the rules of the competition, then we discuss the individual tracks in detail, and finally we summarize the overall results. All reported runtimes are wall-clock time and the newly submitted solvers did not produce any contradicting results. Only dual_ooq13 reported "unsat" on the formula `p20-5.pddl_planlen=41` (Application track, family bomb) which was decided to be "sat" by 9 other solvers.

Rules Each of the three tracks was evaluated and analyzed individually. In each track, the three solvers which solved the most formulas were awarded a medal (gold, silver, bronze). For each formula a time limit of 900 seconds (wall-clock time) and a memory limit of 7GB was set. No parallel solvers participated.

Each solver could receive at most one medal in each track, i.e., if multiple configurations of the same solver were ranked among the first three solvers, the developer was awarded the higher-valued medal. The other medal was given to the developer of the next best solver. This is because most solver configurations differed only by using external tools (i.e., using a third-party preprocessor). The use of such tools is allowed, however it seems unfair to award multiple prizes to a developer for the same solver core. This also avoids the possibility of rewarding developers multiple times per track for multiple identical submissions (although there were no such submissions). The overall best three solvers were additionally awarded Kurt Gödel medals of the FLoC 2014 Olympic Games. Therefore, we accumulated the results of the three tracks discussed above weighted by their numbers of formulas such that the results of each track contribute to the same extent.

QBFLib Track Figure 1 summarizes the results of the QBFLib track. Details are shown in Table 4 in the Appendix. Overall, 67 formulas were solved by no solver, 32 formulas were solved by all solvers. Further, 13 formulas were solved by only one solver (brareqs (3), GhostQ (2), hiqqer1 (1), hiqqer3 (1), pre_dual_ooq13 (2), rareqs (2), sqube (2)). The solver GhostQ solved the most formulas (143 of 276) in the configurations cghostq and bcghostq, where the latter uses Bloqqer if the formula shrinks to at least a certain size. The runtimes of these two configurations are almost identical, indicating that in most cases, bcghostq solves the formula which has not been preprocessed. In both cases the formula is passed to the cghostq configuration of GhostQ. From 158 formulas with two quantifier alternations, cghostq solved 96. In second place was DepQBF in configuration xbdepqbf (138 formulas) and in third place was RAReQS in configuration brareqs (134 formulas). From the formulas with two quantifier alternations, the solvers xbdepqbf and brareqs solved 70 and 88 formulas, respectively. Figure 2 compares the runtimes of these three solvers in a pairwise manner. Recall that 67 formulas could not be solved by any solver. Further, 92 formulas could not be solved by either xbdepqbf or cghostq, 115 formulas could not be solved by brareqs or cghostq, and 124 could not be solved by brareqs or xbdepqbf. With an average runtime of 20 seconds, brareqs performs especially well on satisfiable formulas, whereas cghostq has

14. https://www.starexec.org/

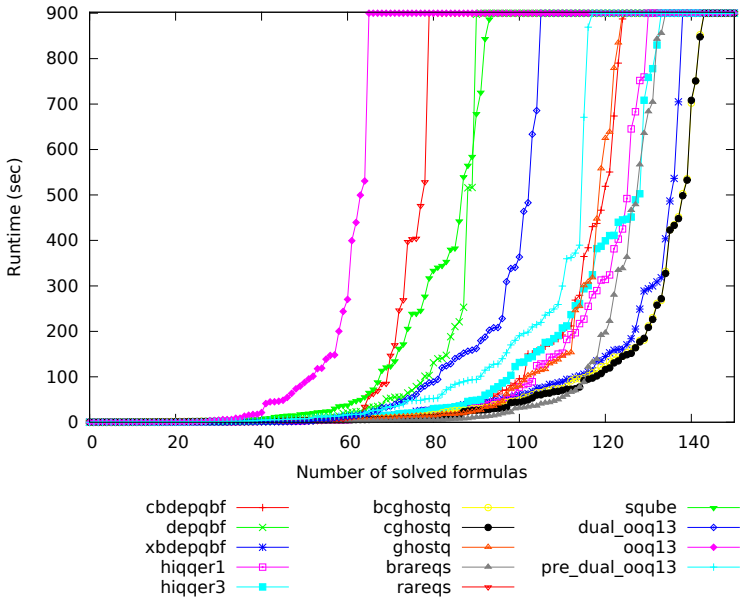

Figure 1. Runtimes (sec) of QBFLib track

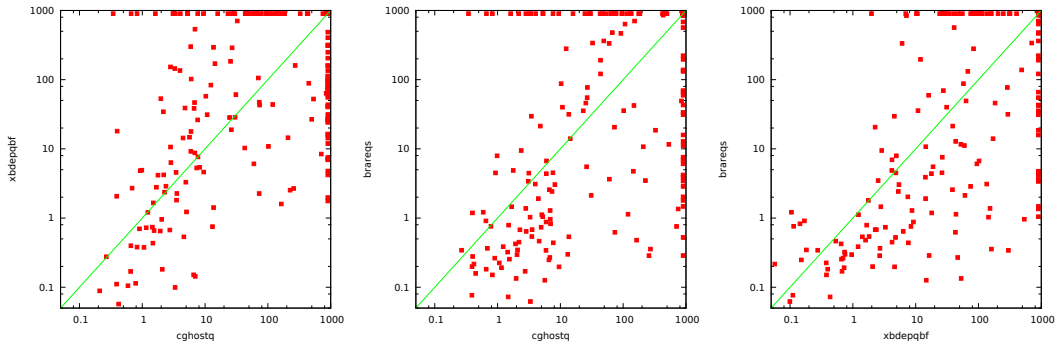

Figure 2. Pairwise comparison of the top-ranked solvers of the QBFLib track (runtimes in seconds)

an average runtime of 84 seconds and xbdepqbf has an average runtime of 57 seconds. On unsatisfiable instances brareqs uses approximately two/three times the average running time of GhostQ and xbdepqbf. Both rareqs and DepQBF perform much worse if no preprocessing is applied, indicating (also confirmed through the success of GhostQ) that the structural information of the formulas could be used to improve solving.

Preprocessing Track The Preprocessing track was similar to the QBFLib track discussed above. The formulas used are almost the same, with the difference that they have been extensively preprocessed. Therefore, most of the structural information available in the formulas is used for performing the simplifications. This leads to results different to the results of the QBFLib track.

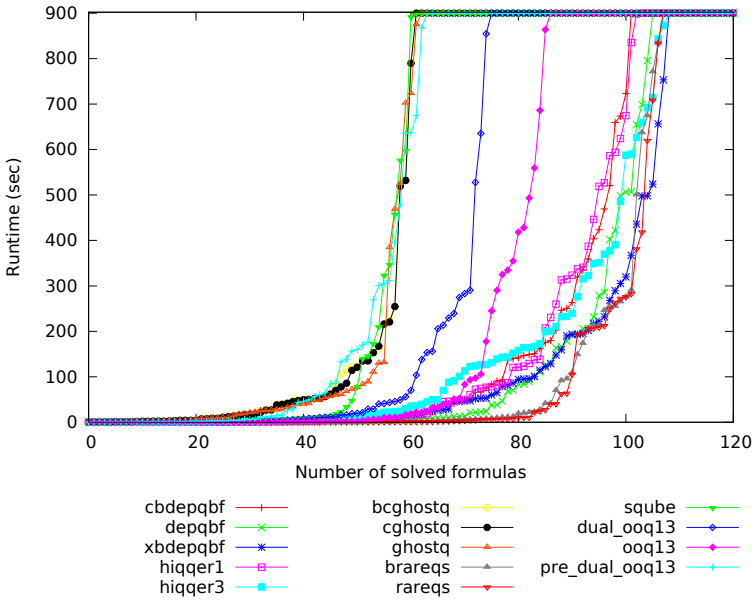

Figure 3. Runtimes (sec) of Preprocessing track

Figure 3 summarizes the results of the experiments, the details are shown in Table 5. In the Preprocessing track, the best three solvers (hiqqer3 (109), xbdepqbf (108), and brareqs (107)) are very close w.r.t. the number of solved formulas and differ on only one formula. Now, 91 formulas were solved by no solver and 29 formulas were solved by all solvers. Three formulas were solved by one solver only (DepQBF, ooq13, and sqube).

Not surprisingly, the preprocessing employed by the different solver configurations shows less effect or even negatively impacts the runtimes. For example, rareqs is faster than brareqs, DepQBF is faster than cbdepqbf, and ooq13 is faster than dual_ooq13, because little structural information is available for recovering the dual representation of the formula. The configurations of hiqqer and also xbdepqbf apply not only Bloqqer (or variants thereof) but also use failed literal probing. This seems to still be effective.

In the pairwise comparison of the three top-ranked solvers shown in Figure 4 it is clear that hiqqer and DepQBF have a common code base and implement similar techniques. The solver RAReQS which implements a CEGAR-based approach, performs better on most formulas which were solved by xbdepqbf or cghostq.

Application Track Figure 5 summarizes the results of the Application track. The details are shown in Table 6. Of the entire set, 16 formulas were only solved by brareqs, 8 formulas were only solved by rareqs, 2 formulas were only solved by hiqqer1. Further, xbdepqbf, hiqqer3, and GhostQ each solved one formula that could not be solved by any other solver. 83 formulas were solved by no solver and 112 formulas were solved by all solvers. By far the most successful solver was brareqs (544 formulas), which solved 93 more formulas than hiqqer3 (431 formulas). In third place was hiqqer1 (422 formulas) followed by xbdepqbf (418 formulas).

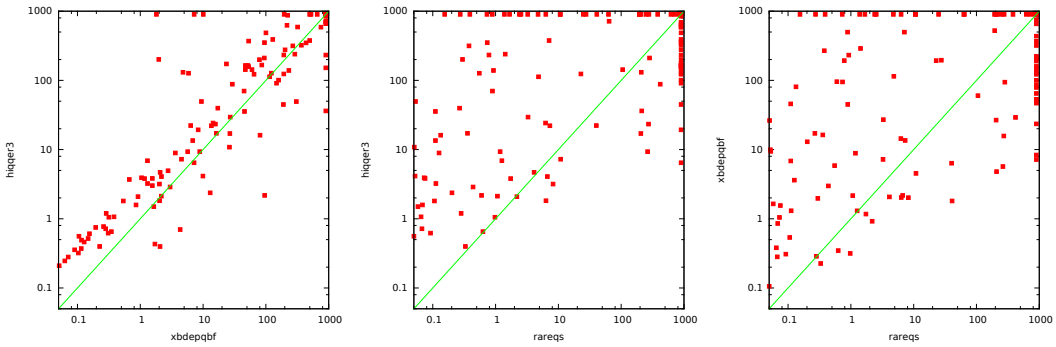

Figure 4. Pairwise comparison of the top-ranked solvers of the Preprocessing track (runtimes in seconds)

If analyzed family-wise, then for family bomb, dual_ooq13 was the most successful solver, solving 100 formulas. Most of the other solvers solved roughly 80 formulas of this set, and only DepQBF, ooq13, and squbе solved fewer than 70 formulas. This family has two quantifier alternations.

The dungeon family also contains formulas with two quantifier alternations. This set was best handled by xbdepqbf and both versions of hiqqer followed by brareqs. For the sauer-reimer family (also two quantifier alternations), cghostq solved 14 more formulas than the basic version GhostQ followed by the solvers employing preprocessing with Bloqqer.

In the reduction finding family (two or three quantifier alternations), brareqs solves 15 more formulas than rareqs and about 35 more formulas than the majority of the solvers. The qbf-hardness family is very suited to the solvers based on DepQBF. Here both variants of hiqqer (87 formulas) and the two versions of DepQBF solved the most formulas (80 formulas). The solver brareqs could solve 69 formulas. Formulas of this set have between 10 and 60 quantifier alternations, indicating that search-based QBF solving as implemented in DepQBF can effectively handle many quantifier alternations.

Finally, brareqs and rareqs could solve almost all formulas (145 and 146 out of 147 formulas) of the planning-CTE family, whereas the next best solver DepQBF (without preprocessing) could solve less than half of the formulas. These formulas have eight quantifier alternations.

The detailed comparison of the top-ranked solvers are shown in Figure 6. As hiqqer1 and hiqqer3 are two configurations of hiqqer, we also included the next best solver xbdepqbf. The configurations of hiqqer and xbdepqbf show very similar behavior, while brareqs solves the majority of the formulas faster underpinning its dominance in this track.

Overall Results Of the 1254 formulas, approx. 18% were not solved by any solver and 14% were solved by all solvers. In total, approximately 2500 hours of computing time was needed to finish the experiments. The winners of the three Kurt Gödel medals were determined as follows. We accumulated the results of the three tracks discussed above weighted by their numbers of formulas. In the official award ceremony, the medals were given to RAReQS with configuration brareqs (2.81 points), hiqqer with configuration hiqqer3 (2.35 points), and DepQBF with configuration xbdepqbf (2.30 points). Note that these three

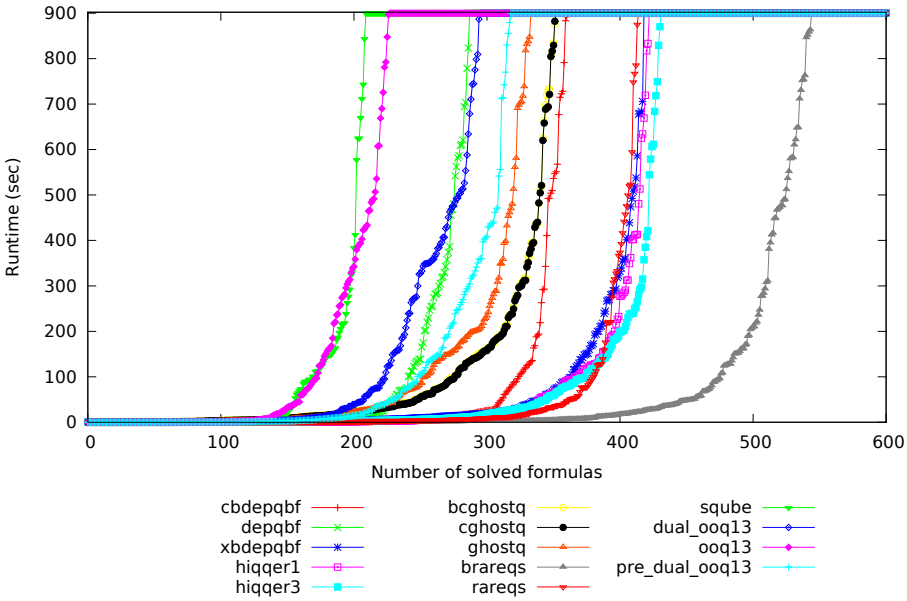

Figure 5. Runtimes (sec) of Application track

Table 2. Numbers of solved formulas for the three tracks QBFLib, Preprocessing (Prepro.) and Applications (App.), the sum of solved instances over the three tracks (sum) and the sum weighted w.r.t. formulas contained in the benchmark set of each track (weighted)

	QBFLib	Prepro.	App.	sum	weighted
brareqs	134	107	544	785	2.81
hiqqer3	133	109	431	673	2.35
xbdepqbf	138	108	418	664	2.30
hiqqer1	130	103	422	655	2.29
rareqs	79	107	414	600	2.20
cbdepqbf	125	101	361	587	2.02
bcghostq	143	61	352	556	1.86
cghostq	143	61	352	556	1.86
ghostq	124	62	333	519	1.76
pre_dual_ooq13*	117	63	318	498	1.70
depqbf	91	105	287	483	1.69
dual_ooq13*	105	75	296	476	1.63
ooq13*	65	86	227	378	1.33
sqube*	94	61	209	364	1.21

*hors concours

solvers were also the solvers which solved the most formulas overall. As we have seen above, however, the ranking of the individual tracks partly gives a different picture.

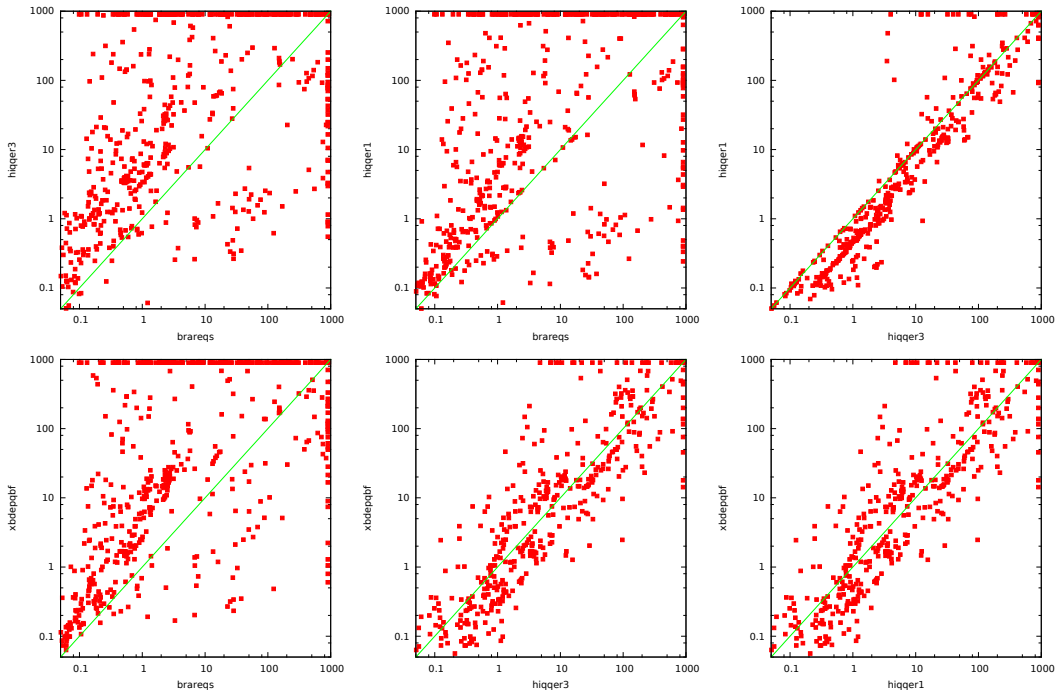

Figure 6. Pairwise comparison of the top-ranked solvers of the Application track

Table 3 shows the ranking under a different ranking scheme, namely the *State-of-the-Art-Contribution* (SOTAC) ranking scheme as used in the CADE ATP System Competition[15.] (CASC). SOTAC is calculated as follows: the SOTAC value of each formula is the inverse of the number of solvers which solved the formula. Then, the SOTAC value of each solver is the average SOTAC value of the problems it could solve [35].

In the SOTAC ranking, the two configurations of RAReQS, i.e., rareqs without any preprocessing and brareqs, the version with preprocessing enabled, are ranked top, while the plain version is ranked fifth in the solved instance count ranking. This indicates that the CEGAR-based approach of RAReQS can be very efficient when other solving techniques are not. As we have see above, RAReQS performs particularly well on the benchmarks of the Application track.

In a virtual experiment, a portfolio approach using all 14 participating solvers each running on its own core in parallel was evaluated. This *virtual best solver* (VBS) would solve 1013 formulas (454 satisfiable, 559 unsatisfiable) with a total running time about half shown by the best single solvers. The average runtime per formula (including timeouts) for the VBS solver is 251 seconds, the average runtime per formula (including timeouts) for brareqs, xbdepqbf, and hiqqer3 is 433 seconds, 476 seconds, and, respectively, 488 seconds. When considering the minimal runtime for each formula, rareqs contributed about 25% to the overall set of formulas solved by the VBS, brareqs contributed about 15%, and depqbf, ooq13, hiqqer1, and dual_ooq13 contributed around 10% each. The rest of the

15. http://www.cs.miami.edu/~tptp/CASC/

Table 3. State-of-the-Art-Contribution (SOTAC) ranking for the three tracks QBFLib, Preprocessing (Prepro.) and Applications (App.), the sum of the SOTAC for the three tracks (weighted), and the SOTAC for the complete benchmark set (overall)

	QBFLib	Prepro.	App.	weighted	overall
RAReQS	0.1451	0.1615	0.2153	0.5219	0.1759
brareqs	0.1517	0.1584	0.2030	0.5131	0.1665
dual_ooq13*	0.0961	0.2217	0.1808	0.4985	0.1363
bcghostq	0.1588	0.0951	0.1175	0.3714	0.1096
cghostq	0.1588	0.0951	0.1175	0.3714	0.1096
hiqqer3	0.1222	0.1251	0.1194	0.3667	0.1009
hiqqer1	0.1200	0.1187	0.1188	0.3575	0.1000
GhostQ	0.1555	0.0890	0.1053	0.3499	0.0989
xbdepqbf	0.1143	0.1136	0.1125	0.3405	0.0951
cbdepqbf	0.1102	0.1100	0.1024	0.3226	0.0874
depqbf	0.1000	0.1197	0.0961	0.3158	0.0824
sqube*	0.1281	0.1002	0.0838	0.3120	0.0820
pre_dual_ooq13*	0.1235	0.0857	0.0932	0.3024	0.0812
ooq13*	0.0834	0.1242	0.0915	0.2991	0.0787

*hors concours

solvers contributed between 1% and 5%. It is remarkable that all solvers contributed to the VBS. This indicates that the various solving techniques are complementary in their effectiveness and that portfolio solving, which has hardly been exploited for QBF solving (one approach is the solver AQME [32], which is—to the best of our knowledge—currently not being developed) could be a promising direction for future work.

6. Conclusion

The Olympic motto "*The most important thing is not to win but to take part*" also holds for competitions like the QBF Gallery. Winning a prize is a good personal incentive for the participating solver developers, but the overall contribution is much stronger. From competitions we can learn what challenges the community has to tackle by considering not the solved problems, but the problems that are currently out of scope for state-of-the-art solvers. Furthermore, we get a comparison of the current systems conducted in a homogeneous environment.

In this paper, we presented the results of the QBF Gallery 2014, a competitive evaluation of QBF solvers. The solvers competed in three different tracks for track medals and, additionally, the overall best three solvers were awarded with a Kurt Gödel medal. As is common in other competitions like the SAT competition, the ranking was based on the number of solved formulas within a certain time. Interestingly, each track was won by a different solver.

The benchmarks and logfiles of the competition are available at

http://qbf.satisfiability.org/gallery

In future editions of the QBF Gallery, more tracks should be considered. From a practical side, there is a strong interest in efficiently solving 2-QBF problems, i.e., formulas with only one quantifier alternation. Further, several recent works underpin the conjecture that the transformation to CNF is counterproductive for efficient QBF solving. Therefore, in the context of the QBF Gallery, several people formulated a new format which is now available at the QBF Gallery website[16]. This format is very simple to parse but also supports non-prenex, non-CNF formulas with structure sharing. This makes it very attractive for encoding application problems in QBF.

Acknowledgments

We would like to thank Aaron Stump and Cesare Tinelli for giving us resources on the StarExec cluster and for providing us technical support with the cluster while running the experiments. Further, we would like to thank our judges Daniel Le Berre, Horst Samulowitz, and Christoph Wintersteiger and the contributors of benchmarks. Special thanks to Norbert Manthey for providing Qprocessor. Martina Seidl was supported by the Austrian Science Fund (FWF) under grant S11408-N23 and the Vienna Science and Technology Fund (WWTF) under grant ICT10-018. Florian Lonsing was supported by the Austrian Science Fund (FWF) under grant S11409-N23. Charles Jordan was supported by the Japan Society for the Promotion of Science (JSPS) under grants 25106501 and 15H00847. Finally, we would like to thank the anonymous referees for their valuable feedback on an earlier version of this paper.

References

[1] Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *Journal on Satisfiability, Boolean Modeling and Computation*, **5**(1-4):133–191, 2008.

[2] Armin Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, **4**(2-4):75–97, 2008.

[3] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In *Proc. of the 23rd Int. Conference on Automated Deduction (CADE 2011)*, **6803** of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2011.

[4] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, **117**(1):12–18, 1995.

[5] Michael Cashmore, Maria Fox, and Enrico Giunchiglia. Planning as quantified Boolean formula. In *Proc. of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, **242** of *Frontiers in Artificial Intelligence and Applications*, pages 217–222. IOS Press, 2012.

[6] Michael Crouch, Neil Immerman, and J. Eliot B. Moss. Finding reductions automatically. In *Fields of Logic and Computation – Essays Dedicated to Yuri Gurevich on*

16. http://qbf.satisfiability.org/gallery/qcir-gallery14.pdf

the Occasion of His 70th Birthday, **6300** of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2010.

[7] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. of the 6th Int. Conference on Theory and Applications of Satisfiability Testing Conference (SAT 2003), Selected Revised Papers*, **2919** of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[8] Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Proc. of the 19th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2013)*, **8312** of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

[9] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Qube7.0. *Journal on Satisfiability, Boolean Modeling and Computation*, **7**(2-3):83–88, 2010.

[10] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. sQueezeBF: An Effective Preprocessor for QBFs Based on Equivalence Reasoning. In *Proc. of the 13th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, **6175** of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2010.

[11] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of quantified Boolean formulas. *Journal of Artificial Intelligence Research*, **26**:371–416, 2006.

[12] Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press, 2010.

[13] Alexandra Goultiaeva and Fahiem Bacchus. Recovering and utilizing partial duality in QBF. In *Proc. of the 16th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, **7962** of *Lecture Notes in Computer Science*, pages 83–99. Springer, 2013.

[14] Mikoláš Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. In *Proc. of the 15th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, **7317** of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2012.

[15] Mikoláš Janota, Radu Grigore, and Joao Marques-Silva. Counterexample guided abstraction refinement algorithm for propositional circumscription. In *Proc. of the 12th European Conference on Logics in Artificial Intelligence (JELIA 2010)*, **6341** of *Lecture Notes in Artificial Intelligence*, pages 195–207. Springer, 2010.

[16] Mikoláš Janota and Joao Marques-Silva. Abstraction-based algorithm for 2QBF. In *Proc. of the 14th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, **6695** of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2011.

[17] Charles Jordan and Łukasz Kaiser. Benchmarks from reduction finding. In *QBF Workshop*, 2013. http://fmv.jku.at/qbf2013/.

[18] Charles Jordan and Łukasz Kaiser. Experiments with reduction finding. In *Proc. of the 16th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, **7962** of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2013.

[19] Hans Kleine Büning and Uwe Bubeck. Theory of quantified Boolean formulas. In *Handbook of Satisfiability*, **185** of *Frontiers in Artificial Intelligence and Applications*, pages 735–760. IOS Press, 2009.

[20] William Klieber. *Formal Verification Using Quantified Boolean Formulas (QBF)*. PhD thesis, Carnegie Mellon University, available at http://reports-archive.adm.cs.cmu.edu/anon/2014/CMU-CS-14-117.pdf, 2014.

[21] William Klieber, Mikoláš Janota, Joao Marques-Silva, and Edmund Clarke. Solving QBF with free variables. In *Proc. of the 19th Int. Conference on Principles and Practice of Constraint Programming (CP 2013)*, **8124** of *Lecture Notes in Computer Science*, pages 415–431. Springer, 2013.

[22] William Klieber, Samir Sapra, Sicun Gao, and Edmund Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *Proc. of the 13th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, **6175** of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2010.

[23] Martin Kronegger, Andreas Pfandler, and Reinhard Pichler. Conformant planning as a benchmark for QBF-solvers. In *QBF Workshop*, 2013. http://fmv.jku.at/qbf2013/.

[24] Reinhold Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *Proc. of the Int. Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2002)*, **2381** of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2002.

[25] Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *Journal on Satisfiability, Boolean Modeling and Computation*, **7**(2-3):71–76, 2010.

[26] Florian Lonsing and Armin Biere. Failed literal detection for QBF. In *Proc. of the 14th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, **6695** of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2011.

[27] Florian Lonsing, Uwe Egly, and Allen Van Gelder. Efficient clause learning for quantified Boolean formulas via QBF pseudo unit propagation. In *Proc. of the 16th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, **7962** of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2013.

[28] Florian Lonsing, Martina Seidl, and Allen Van Gelder. The QBF Gallery: Behind the Scenes. *Accepted subject to minor revisions by Artificial Intelligence*, 2015.

[29] Norbert Manthey. Coprocessor 2.0 - A flexible CNF simplifier. In *Proc. of the 15th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, **7317** of *Lecture Notes in Computer Science*, pages 436–441. Springer, 2012.

[30] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In *Proc. of the 8th Int. Haifa Verification Conference (HVC 2012)*, **7857** of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2012.

[31] Christian Miller, Stefan Kupferschmid, Matthew D. T. Lewis, and Bernd Becker. Encoding techniques, Craig interpolants and bounded model checking for incomplete designs. In *Proc. of the 13th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2010)*, **6175** of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2010.

[32] Luca Pulina and Armando Tacchella. AQME'10. *Journal on Satisfiability, Boolean Modeling and Computation*, **7**(2-3):65–70, 2010.

[33] Marko Samer and Stefan Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, **42**(1):77–97, 2009.

[34] Matthias Sauer, Sven Reimer, Ilia Polian, Tobias Schubert, and Bernd Becker. Provably optimal test cube generation using quantified Boolean formula solving. In *Proc. of the 18th Asia and South Pacific Design Automation Conference, (ASP-DAC 2013)*, pages 533–539. IEEE, 2013.

[35] Geoff Sutcliffe. Proceedings of the 6th IJCAR ATP system competition (CASC-J6). In *CASC-J6*, **11** of *EPiC Series*, pages 1–50. EasyChair, 2012.

[36] Allen Van Gelder, Samuel B. Wood, and Florian Lonsing. Extended failed-literal preprocessing for quantified Boolean formulas. In *Proc. of the 15th Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, **7317** of *Lecture Notes in Computer Science*, pages 86–99. Springer, 2012.

[37] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proc. of the 2002 IEEE/ACM Int. Conference on Computer-aided Design (CAD 2002)*, pages 442–449. ACM, 2002.

[38] Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In *Proc. of the 8th Int. Conference on Principles and Practice of Constraint Programming (CP 2002)*, **2470** of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2002.

Appendix A. Detailed Results

Table 4. Details on results from the QBFLib track: number of solved satisfiable formulas (#sat), number of solved unsatisfiable formulas (#unsat), total number of solved formulas (#total), average runtime for satisfiable formulas in seconds (avg sat (s)), average runtime for unsatisfiable formulas in seconds (avg unsat (s)), total runtime in seconds for all formulas

solver	#sat	#unsat	#total	avg sat (s)	avg unsat (s)	time total (s)
cghostq	80	63	143	84.80	44.61	129294.94
bcghostq	80	63	143	88.28	47.79	129773.90
xbdepqbf	70	68	138	57.26	63.15	132502.62
brareqs	66	68	134	19.23	108.85	136472.02
hiqqer3	71	62	133	85.47	113.51	141806.63
hiqqer1	66	64	130	56.25	92.79	141051.90
cbdepqbf	63	62	125	68.97	81.58	145303.48
ghostq	68	56	124	72.52	52.16	144652.62
pre_dual_ooq13	64	53	117	92.15	49.11	151601.22
dual_ooq13	60	45	105	81.06	51.09	161063.41
sqube	50	44	94	111.49	116.02	174479.85
depqbf	41	50	91	69.37	34.35	171062.03
rareqs	32	47	79	69.57	28.79	180879.88
ooq13	30	35	65	54.90	72.14	194072.36

Table 5. Details on results from the Preprocessing track: number of solved satisfiable formulas (#sat), number of solved unsatisfiable formulas (#unsat), total number of solved formulas (#total), average runtime for satisfiable formulas in seconds (avg sat (s)), average runtime for unsatisfiable formulas in seconds (avg unsat (s)), total runtime in seconds for all formulas

solver	#sat	#unsat	#total	avg sat (s)	avg unsat (s)	time total (s)
hiqqer3	63	46	109	138.93	116.85	134728.02
xbdepqbf	61	47	108	71.74	91.92	130197.09
rareqs	63	44	107	48.70	67.76	128450.44
brareqs	64	43	107	66.15	56.34	129056.86
depqbf	59	46	105	62.51	89.97	132027.15
hiqqer1	59	44	103	107.24	91.57	136356.71
cbdepqbf	59	42	101	79.24	86.34	136101.77
ooq13	50	36	86	57.24	87.18	147300.85
dual_ooq13	49	26	75	42.94	108.10	156114.85
pre_dual_ooq13	44	19	63	68.80	181.84	168482.55
ghostq	46	16	62	60.59	155.25	168171.32
cghostq	43	18	61	51.26	120.72	168177.70
bcghostq	43	18	61	53.82	121.37	168299.07
sqube	40	21	61	67.99	71.52	168021.73

Table 6. Details on results from the Application track: number of solved satisfiable formulas (#sat), number of solved unsatisfiable formulas (#unsat), total number of solved formulas (#total), average runtime for satisfiable formulas in seconds (avg sat (s)), average runtime for unsatisfiable formulas in seconds (avg unsat (s)), total runtime in seconds for all formulas

solver	#sat	#unsat	#total	avg sat (s)	avg unsat (s)	time total (s)
brareqs	232	312	544	21.13	78.98	201446.67
hiqqer3	207	224	431	57.67	37.39	293913.56
hiqqer1	197	225	422	42.29	44.97	300152.10
xbdepqbf	199	219	418	44.81	51.74	305550.61
rareqs	142	272	414	14.30	48.41	304099.67
cbdepqbf	157	204	361	19.55	62.37	352394.35
cghostq	184	168	352	57.17	98.63	371791.52
bcghostq	184	168	352	58.48	99.68	372209.25
ghostq	182	151	333	63.34	104.28	389076.30
pre_dual_ooq13	151	167	318	50.24	103.95	400247.34
dual_ooq13	87	209	296	34.61	131.90	425679.09
depqbf	157	130	287	61.49	60.55	420727.14
ooq13	92	135	227	37.15	138.42	479305.11
sqube	84	125	209	53.66	65.03	486036.72