Resolution-Based Certificate Extraction for QBF (Tool Presentation)*

Aina Niemetz, Mathias Preiner,
Florian Lonsing, Martina Seidl, and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

Abstract. A certificate of (un)satisfiability for a quantified Boolean formula (QBF) represents concrete assignments to the variables, which act as witnesses for its truth value. Certificates are highly requested for practical applications of QBF like formal verification and model checking. We present an integrated set of tools realizing resolution-based certificate extraction for QBF in prenex conjunctive normal form. Starting from resolution proofs produced by the solver DepQBF, we describe the workflow consisting of proof checking, certificate extraction, and certificate checking. We implemented the steps of that workflow in stand-alone tools and carried out comprehensive experiments. Our results demonstrate the practical applicability of resolution-based certificate extraction.

1 Introduction

Over the last 10 years, several approaches and tools supporting the generation of certificates for *quantified Boolean formulae* (QBF) have been presented. An overview of the status quo of 2009 is given in [6]. Initially, the main goal was to use certificates for validating the results of QBF solvers instead of relying on majority votes (only). Therefore, independent verifiers to check the output of a QBF solver have been developed. Such solver outputs are either clause/cube resolution proofs, or functions representing variable assignments. In case of so-called *Skolem/Herbrand* functions as output, we gain further information on the solution of a solved problem, e.g., the path to a bad state in case of model checking. Their extraction, however, is not directly applicable if the successful variant of DPLL style procedures for QBF is employed. In any case, most of these tools and solvers are not maintained anymore.

More recently, the circuit solver CirQit [4] has been extended to produce Q-refutations for true and false QBFs based on its dual propagation mechanism. Due to the circuit representation of a formula, solving the negated formula does not involve any expensive transformations. Furthermore, (partial) solution

* This work was partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018 and by the Austrian Science Fund (FWF) under NFN Grant S11408-N23 (RiSE).

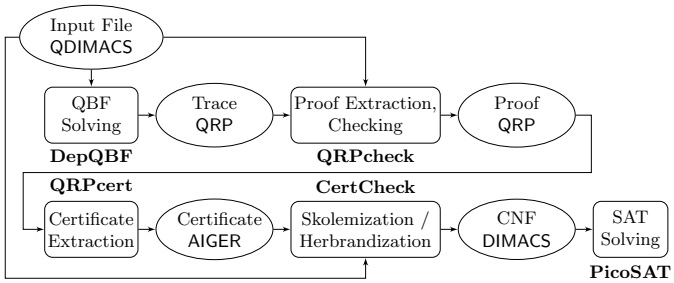


Fig. 1: Certification workflow.

strategies can be extracted. However, CirQit cannot exploit its full strength on formulae in prenex conjunctive normal form (PCNF).

Based on the resolution generation tools discussed above, recently the prototype ResQu [2], which implements an approach to extract Skolem/Herbrand functions from resolution proofs, was presented. Given a resolution proof for true or false QBF, such functions can be extracted in linear time.

In this paper, we present a complete framework for generating QBF certificates from resolution proofs obtained by the state-of-the-art solver DepQBF [5]. We further performed a profound empirical study on the applicability of resolution-based extraction of QBF certificates and discuss strengths and limitations of this approach in detail.

2 The Certification Framework at a Glance

We provide a complete framework to certify and validate the results of state-of-the-art QBF solver DepQBF [5], a dependency-aware search-based QBF solver for QBFs in PCNF. The certification workflow is shown in Figure 1. The framework consists of a chain of loosely coupled stand-alone tools on top of DepQBF, which support both proof extraction and checking (QRPcheck) as well as certificate extraction and validation (QRPcert, CertCheck, and PicoSAT). The components of our framework are described as follows.

Trace Extraction. We instrumented DepQBF to output traces in our novel, text-based QRP format.[1] A trace represents the set of all resolution sequences involved in generating learnt constraints in a search-based QBF solver. Our approach of tracing is similar to [7], except that each single resolution step has *exactly* two antecedents. This way, exponential worst-case behaviour during reconstruction of resolvents from unordered lists of antecedents is avoided [3]. Alternatively, the QIR format [1] used in [4] allows multiple antecedents but predefines the ordering in which resolvents should be reconstructed. As far as resolution is concerned, QRP proofs can be checked in deterministic log space, a desirable property of proof formats suggested in [3]. Syntactically, QRP is a lightweight format as it does not distinguish between (resolution steps over) clauses and cubes explicitly.

[1] See also Sec. A in the appendix

Proof Extraction and Checking. QRPcheck is a proof checker for resolution-based traces and proofs of (un)satisfiability in QRP format. Starting with the empty constraint, QRPcheck extracts the proof from a given trace on-the-fly while checking each proof step incrementally. This way, all parts of the trace irrelevant for deriving the empty constraint are omitted. In case of a proof of satisfiability, QRPcheck further provides the possibility to check that each initial (input) cube, which was generated during constraint learning, can be extended to satisfy the matrix. For this propositional check we are using the SAT solver PicoSAT.[2] In order to handle very large traces and proofs we map the input file to memory using virtual memory mechanisms ("mmap").

Certificate Extraction. QRPcert is a tool for extracting Skolem/Herbrand function-based QBF certificates from resolution proofs in QRP format. It implements the algorithm presented in [2], but traverses the resolution proof in the reverse topological order starting from the empty constraint. Hence, all irrelevant parts of the proof are omitted similarly to proof extraction by QRPcheck. The extracted certificates are represented as AIGs[3], which are simplified by common basic simplification techniques, including structural hashing and constant propagation.

SAT-Based Certificate Checking. The tool CertCheck transforms the given input formula into an AIG and merges the result with the certificate by substituting each existentially (universally) quantified input variable with its Skolem (Herbrand) function. The resulting AIG is first translated into CNF via Tseitin transformation and then checked for being tautological (unsatisfiable). We validate the correctness of the certificate by checking the resulting CNF with the SAT solver PicoSAT.

3 Experiments

We applied our framework on the benchmark sets of the QBF competitions 2008 and 2010 consisting of 3326 and 568 formulas, respectively[4]. We considered only those 1229 and 362 formulas solved by DepQBF within 900 seconds. Instead of advanced dependency schemes, we used the orderings of quantifier prefixes in DepQBF. All experiments[5] were performed on 2.83 GHz Intel Core 2 Quad machines with 8 GB of memory running Ubuntu 9.04. Time and memory limits for the whole certification workflow were set to 1800 seconds and 7 GB, respectively.

Out of 362 solved instances of the QBFEVAL'10 benchmark set, our framework was able to check 348 proofs and extract 337 certificates, of which 275 were validated successfully. DepQBF required almost 5000 seconds for solving and tracing the 275 instances that were validated by PicoSAT, whereas the certification of those instances needed about 5600 seconds. On 14 instances, QRPcheck

[2] `http://www.fmv.jku.at/picosat`
[3] ASCII AIGER format: `http://fmv.jku.at/aiger/FORMAT.aiger`
[4] Available at `http://www.qbflib.org/index_eval.php`
[5] Log files and binaries are available from `http://fmv.jku.at/cdepqbf/`.

ran out of memory, as the file size of the traces produced by DepQBF were 16 GB on average with a maximum of 27 GB. QRPcert ran out of memory on 11 proofs with an average file size of 3.6 GB and a maximum of 5.9 GB.

The largest number of instances (62) were lost during the validation process. PicoSAT timed out on 17 instances and ran out of memory on 45 instances. From 62 instances that were not validated by PicoSAT, 51 instances are part of the 'mqm' family, which consists of a total of 128 formulae with 70 instances being unsatisfiable and 58 satisfiable. PicoSAT was able to validate all 70 unsatisfiable instances, but did not succeed in validating even one satisfiable instance. This is due to the fact that proofs of satisfiability tend to grow much larger than proofs of unsatisfiability mostly because of the size of the initial cubes. For example, the proofs of the 51 instances that were not validated by PicoSAT have 40000 intial cubes on average, where each cube has an average size of 970 literals.

We evaluated the runtime of each component of the framework w.r.t. the 275 certified instances. First, we compared the time required by DepQBF for solving and tracing to the aggregated time needed by QRPcheck, QRPcert, CertCheck, and PicoSAT for certification. Figure 2a shows that the whole certification process requires marginally more time than DepQBF on average. It also shows that only a few instances are responsible for the certification process being slower in total runtime than DepQBF. In fact, three instances require more than 59% of the total certification runtime, in contrast to 34% of total solving time.

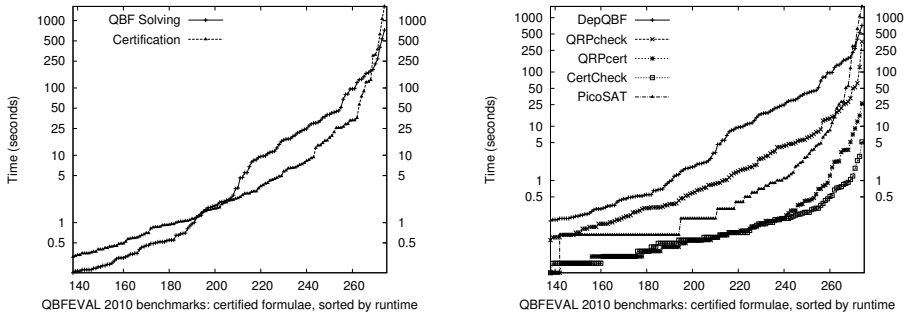
We further compared the runtime of each component of the framework, which is depicted in Figure 2b. More than 77% of the certification time is required for validating the certificates with PicoSAT, where three instances require over 58% of the total certification time. Extracting and checking proofs with QRPcheck requires about 20% of the total certification time, which typically involves heavy I/O operations in case proof extraction is enabled. Considering all checked instances, disabling proof extraction saves about 54% of the runtime of QRPcheck. The extraction of certificates and the CNF conversion takes a small fraction of the total certification time, which is approximately 2% and 1%, respectively.

Table 1[6] summarizes the results. Certification heavily depends on whether an instance is satisfiable or unsatisfiable, especially for certificate validation. On average over 91% of the solved unsatisfiable instances were certified in about 61% of the solving time. For 74% of the solved satisfiable instances, the certification took over four times the solving time.

Certificate validation requires most of the time. Particularly vaildating satisfiable instances is time-consuming. Given the QBFEVAL'10 set, traces of satisfiable instances are on average 2-3 times larger than traces of unsatisfiable instances and further contain in the worst case 13 million steps with 1.4 billion literals and 18 million steps with 1.3 billion literals, respectively. The difference between proofs of satisfiability and unsatisfiability is even larger by a factor of eight on average.

An interesting property of the generated AIG certificates is the number of and-gates involved, where certificates of satisfiability are on average (and in the

[6] See also Tables 2 and 3 in the appendix for detailed statistics.



(a) Solving vs. certification (b) Tool comparison

Fig. 2: Runtime comparison, all instances with solving time $\geq$0.2s considered.

median) over 100 times larger than certificates of unsatisfiability. The maximum number of and-gates generated for AIG certificates of satisfiability and unsatisfiability are 147 million resp. 10 million and-gates. Compared to certificates of unsatisfiability, CNFs generated for validating certificates of satisfiability are on average up to 70 times larger and contain in the worst-case over 10 times more clauses with a maximum of 440 million clauses.[7] On certain instances, the file size of traces were enourmous with almost 52 GB and 27 GB in the worst-case in the QBFEVAL'08 and QBFEVAL'10 benchmark sets, respectively.

Finally, we investigated the 14 instances (4 sat., 10 unsat.) of the QBFEVAL'10 benchmark set that were not checked by QRPcheck due to given memory constraints. The corresponding traces had an average file size of 16 GB (with 27 GB as a maximum) and 17 million steps with 3.3 billion literals on average. For these 14 instances, we lifted the previous memory limit of 7 GB and rerun the experiments on a machine with 96 GB and a time limit of 3600 seconds. As a consequence, we were able to certify 12 out of 14 instances. On two instances, PicoSAT timed out while validating CNFs with 3 and 30 million clauses, respectively. Certification of the other 12 instances took less than 4600 seconds in total, whereas DepQBF required over 7700 seconds for solving and tracing altogether.

The average (median) time for the whole workflow on all 14 instances was 1412 (1014) seconds. File sizes of the extracted proofs were ranging from 85% to 0.0001% w.r.t. the trace size with a maximum of 14.6 GB and a minimum of 13 kB. The average (median) number of steps was 18 (10) million in the traces, and 4 (0.1) million in the extracted proofs. The ratio of proof size over trace size in the number of steps for each of the 14 instances was 0.23 on average. As an extreme case, the certified satisfiable instance `blocks_enc_2_b3_ser--opt-9_-shuffled.qdimacs` resulted in a trace of more than 50 million steps and 18 GB file size for which a proof of only 38 (!) steps and 47 kB file size was extracted. The average (median) memory usage for the whole workflow was 19 (18) GB with a maximum of 28 GB.

[7] See also Tables 4 and 5 in the appendix for detailed statistics.

Table 1: Runtime comparison, runtime considers validated instances only.

		Instances				**Total Time [s]**			
		sv	ch	ex	va	DepQBF	QRPcheck	QRPcert	PicoSAT
2008	sat	494	476	464	397	3502.9	911.6	95.3	13874.1
	unsat	735	690	685	673	9863.7	2938.1	831.8	2639.8
	total	1229	1166	1149	1070	13366.6	3849.7	927.1	16513.9
2010	sat	157	153	143	86	701.8	80.1	30.9	3247.0
	unsat	205	195	194	189	4241.9	1011.5	86.8	1090.0
	total	362	348	337	275	4943.7	1091.7	117.6	4337.0

4 Discussion

In this paper, we presented the first framework for complete and robust certification of QBF using the state-of-the-art QBF solver DepQBF. We presented tools for proof extraction, proof checking, certificate extraction and certificate validation. We further performed an extensive evaluation on recent benchmark sets, which shows that our framework is able to extract certificates for over 90% of solved instances. Further, we were able to validate over 80% of extracted certificates, which all were proved correct.

In future work, we consider to extend DepQBF to maintain proofs internally in order to extract certificates directly from the solver. We also plan to extend QRPcert to support advanced dependency schemes as applied in DepQBF. Further, we want to improve the process of certificate validation as it is considered to be a bottleneck in the current framework. We believe that this technology will finally actually enable the application of QBF solving in practice, both in already proposed as well as new applications.

References

1. QIR Proof Format (Version 1.0). Website. `http://users.soe.ucsc.edu/~avg/ProofChecker/qir-proof-grammar.txt`.
2. V. Balabanov and J. R. Jiang. Resolution Proofs and Skolem Functions in QBF Evaluation and Applications. In *CAV'11*, volume 6806 of *LNCS*. Springer, 2011.
3. A. Van Gelder. Verifying Propositional Unsatisfiability: Pitfalls to Avoid. In J. Marques-Silva and K. A. Sakallah, editors, *Proc. of SAT'07*, volume 4501 of *LNCS*, pages 328–333. Springer, 2007.
4. A. Goultiaeva, A. Van Gelder, and F. Bacchus. A Uniform Approach for Generating Proofs and Strategies for Both True and False QBF Formulas. In *IJCAI'11*, pages 546–553. IJCAI/AAAI, 2011.
5. F. Lonsing and A. Biere. Integrating Dependency Schemes in Search-Based QBF Solvers. In *SAT*, LNCS, pages 158–171. Springer, 2010.
6. M. Narizzano, C. Peschiera, L. Pulina, and A. Tacchella. Evaluating and Certifying QBFs: A Comparison of State-of-the-Art Tools. *AI Commun.*, 22(4):191–210, 2009.
7. Y. Yu and S. Malik. Validating the Result of a Quantified Boolean Formula (QBF) Solver: Theory and Practice. In *ASP-DAC*, pages 1047–1051. ACM Press, 2005.

Appendix

A QRP Format

```
trace       =  preamble { quant_set } { step } result EOF.

preamble    =  { comment } header.
comment     =  "#" text EOL.
header      =  "p qrp" pnum pnum EOL.

quant_set   =  quantifier { var } "0".
quantifier  =  "a" | "e".
var         =  pnum.

step        =  idx literals antecedents.
idx         =  pnum.
literals    =  { lit } "0".
lit         =  ["-"] var.
antecedents =  [idx [idx]] "0".

result      =  "r " sat EOL.
sat         =  "sat" | "unsat".

text        =  ? a sequence of non-special ASCII chars ?.
pnum        =  ? a 32-bit signed integer > 0 ?.
EOL         =  ? end-of-line marker ?.
EOF         =  ? end-of-file marker ?.
```

Fig. 3: The QRP format in Extended Backus-Naur Form (EBNF).

B QRPcheck

```
1  function check (Step s)
2  {
3    if (is_visited (s) or is_initial (s))
4      return OK
5
6    c ← resolve_and_reduce (s)
7
8    if (c = INVALID)
9      return ERROR
10
11   if (check_constraint (s, c) = ERROR)
12     return ERROR
13
14   if (check (get_first_antecedent (s)) = ERROR)
15     return ERROR
16   else if (is_resolution_step (s))
17     return check (get_second_antecedent (s))
18
19   return OK
20 }
```

Fig. 4: Top-level view of the proof checking algorithm in QRPcheck.

Table 2: QBFEVAL'10 family overview of certification workflow.

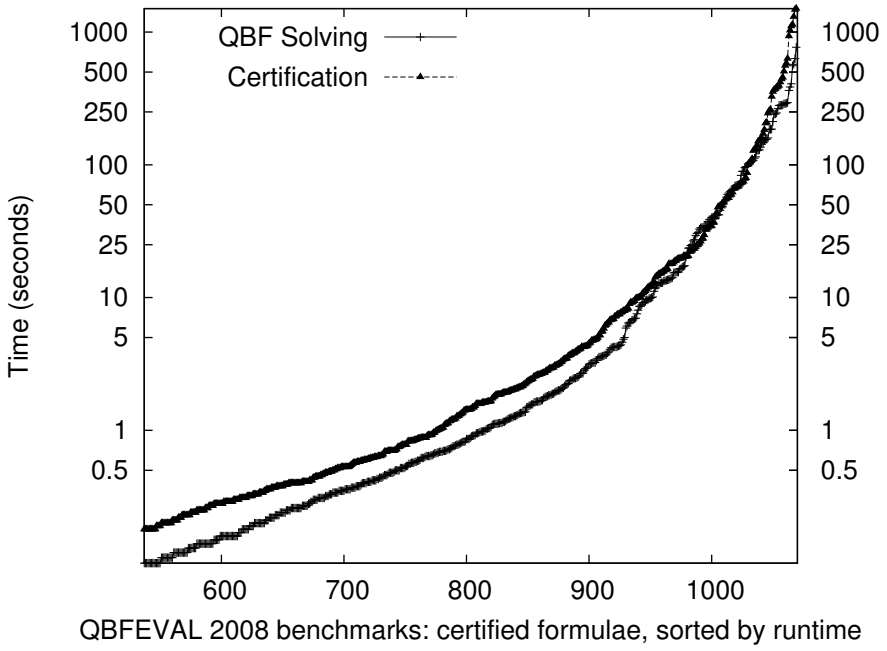
Family	Instances				Time Solv. [s]			Time Cert. [s]		
	sv	ch	ex	va	total	avg	med	total	avg	med
Abduction	48	48	48	48	48.3	1.0	0.0	21.4	0.4	0.1
Adder	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
blackbox-01X-QBF	43	36	36	36	531.3	14.8	0.2	226.4	6.3	0.3
blackbox_design	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
Blocks	4	3	3	3	11.4	3.8	0.1	310.5	103.5	0.1
BMC	12	12	12	12	34.4	2.9	0.5	165.1	13.8	3.4
Chain	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
circuits	2	2	2	2	30.2	15.1	15.1	4.4	2.2	2.2
conformant	5	3	3	3	24.7	8.2	0.1	121.5	40.5	0.2
Connect4	8	8	8	8	40.8	5.1	0.1	13.8	1.7	1.5
Counter	2	2	2	2	229.6	114.8	114.8	20.3	10.2	10.2
Debug	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
evader-pursuer	10	9	9	9	719.5	79.9	0.7	164.8	18.3	2.1
FPGA_*_FAST	2	2	2	2	0.2	0.1	0.1	0.5	0.3	0.3
FPGA_*_SLOW	1	1	1	0	0.0	0.0	0.0	0.0	0.0	0.0
Impl	1	1	1	1	0.0	0.0	0.0	0.0	0.0	0.0
jmc_quant	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
mqm	128	128	121	70	1227.0	17.5	0.6	84.0	1.2	0.4
pan	24	24	21	14	1785.3	127.5	9.0	3103.8	221.7	14.6
Rintanen	1	1	1	1	12.8	12.8	12.8	3.5	3.5	3.5
Sakallah	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
Scholl-Becker	11	10	10	10	30.4	3.0	0.2	30.9	3.1	0.5
SortingNet	6	5	5	4	187.4	46.9	3.7	112.8	28.2	18.9
SzymanskiP	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
tipdiam	3	2	1	1	0.0	0.0	0.0	0.0	0.0	0.0
tipfixpoint	9	9	9	9	2.2	0.2	0.1	6.1	0.7	0.4
Toilet	40	40	40	38	23.2	0.6	0.0	686.0	18.1	0.0
VonNeumann	2	2	2	2	4.9	2.5	2.5	506.0	253.0	253.0
Total	362	348	337	275	4943.7	18.0	0.2	5581.7	20.3	0.3

C Evaluation Details

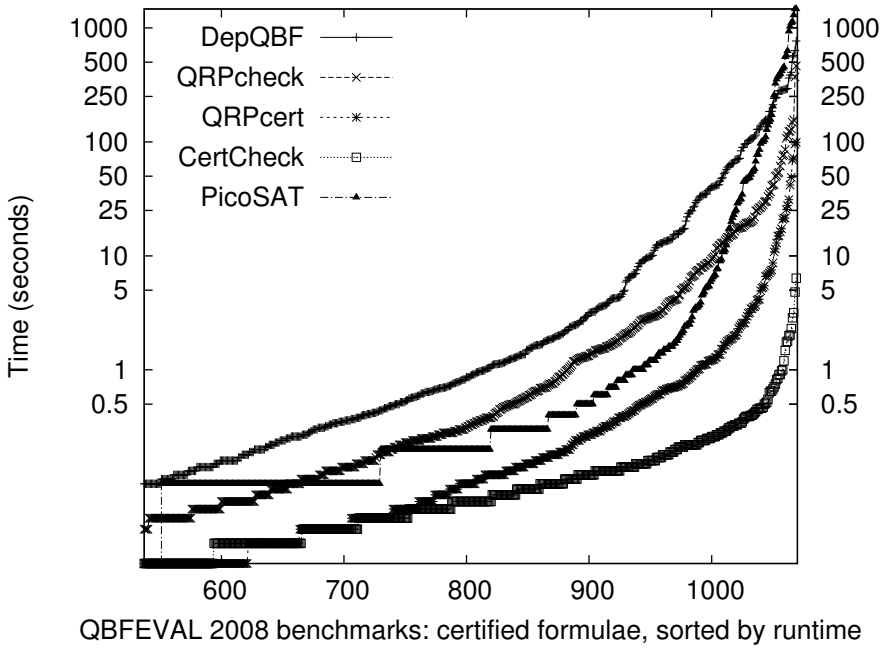
Table 2 and Table 3 show the aggregated results of the QBFEVAL'10 and QBFEVAL'08 benchmarks set grouped by family. The first column contains the number of instances solved by DepQBF (sv), number of proofs checked by QRPcheck (ch), number of certifices extracted by QRPcert (ex) and validated by PicoSAT (va). We omit the number of CNFs generated by CertCheck as it is equal to the number of certificates extracted. The second and third columns indicate the runtime required for solving and certifying all instances that were successfully validated by PicoSAT under given time and memory constraints. Note that the time required for solving includes tracing (DepQBF), whereas the time required for certification includes proof extraction and checking (QRPcheck), certificate extraction (QRPcert), and certificate validation (CertCheck and PicoSAT).

Table 3: QBFEVAL'08 family overview of certification workflow.

Family	**Instances**				**Time Solv. [s]**			**Time Cert. [s]**		
	sv	ch	ex	va	total	avg	med	total	avg	med
Abduction	284	283	283	283	562.4	2.0	0.1	1139.2	4.0	0.1
Adder	5	5	5	5	1.4	0.3	0.0	6.5	1.3	1.0
blackbox-01X-QBF	315	282	279	279	3377.7	12.1	0.1	1514.7	5.4	0.2
blackbox_design	1	1	1	1	0.4	0.4	0.4	0.7	0.7	0.7
Blocks	11	10	10	10	129.0	12.9	0.7	1548.9	154.9	0.1
BMC	81	80	80	80	3277.6	41.0	0.4	1121.0	14.0	0.7
Chain	10	8	6	3	6.9	2.3	1.9	1643.6	547.9	260.5
circuits	5	4	4	4	3.0	0.8	0.6	0.5	0.1	0.1
conformant	11	9	9	8	197.0	24.6	2.2	124.1	15.5	0.7
Counter	10	10	10	10	130.9	13.1	0.0	15.3	1.5	0.0
Debug	1	1	1	0	0.0	0.0	0.0	0.0	0.0	0.0
DFlipFlop	10	10	10	10	1.9	0.2	0.1	24.5	2.5	0.3
evader-pursuer	16	14	14	14	332.8	23.8	0.2	51.5	3.7	1.5
FPGA_*_FAST	5	5	5	5	0.7	0.1	0.1	2.9	0.6	0.1
FPGA_*_SLOW	3	3	3	1	9.1	9.1	9.1	0.5	0.5	0.5
Impl	10	10	10	10	0.0	0.0	0.0	0.0	0.0	0.0
irqlkeapclte	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
jmc_quant	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
MutexP	3	3	3	2	0.1	0.0	0.0	9.2	4.6	4.6
pan	162	157	152	114	3289.7	28.9	0.4	8080.4	70.9	0.3
Rintanen	2	2	2	2	39.3	19.6	19.6	23.4	11.7	11.7
Sakallah	1	1	1	1	0.1	0.1	0.1	1.5	1.5	1.5
Scholl-Becker	38	35	35	35	183.4	5.2	0.1	855.3	24.4	0.0
SortingNet	49	44	44	33	1069.8	32.4	1.8	1347.2	40.8	1.9
SzymanskiP	2	2	2	2	0.0	0.0	0.0	0.0	0.0	0.0
terminator	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
tipdiam	78	76	73	59	10.3	0.2	0.0	1853.2	31.4	0.0
tipfixpoint	73	70	69	68	707.6	10.4	0.1	724.6	10.7	0.4
Toilet	8	7	7	6	28.6	4.8	0.4	211.5	35.2	0.0
Tree	12	11	10	6	0.4	0.1	0.0	420.8	70.1	0.3
uclid	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0
VonNeumann	10	10	10	10	6.5	0.6	0.4	665.6	66.6	15.2
wmiforward	13	13	11	9	0.0	0.0	0.0	0.1	0.0	0.0
Total	1229	1166	1149	1070	13366.6	12.5	0.1	21386.4	20.0	0.2



(a) Solving vs. certification



(b) Individual tools

Fig. 5: Run time overview, all instances with solving time $\geq$0.1s considered.

Table 4: Comparison of traces, proofs, AIGs and CNFs. Avg. and med. values.

		Trace			**Proof**			**AIG**		**CNF**	
		$\bar{s}$	$\bar{l}$	$\tilde{l}$	$\bar{s}$	$\bar{l}$	$\tilde{l}$	$\bar{a}$	$\tilde{a}$	$\bar{c}$	$\tilde{c}$
2008	sat	324k	66M	242k	127k	33M	32k	2M	4k	6M	51k
	unsat	380k	60M	379k	155k	19M	58k	68k	58	409k	40k
	total	357k	62M	334k	144k	25M	43k	872k	197	3M	45k
2010	sat	785k	200M	17M	308k	117M	626k	20M	24k	59M	183k
	unsat	376k	70M	1M	135k	14M	146k	170k	193	846k	55k
	total	556k	127M	2M	211k	60M	175k	8M	369	25M	71k

Table 5: File size comparison in kilobyte (k), megabyte (M) and gigabyte (G).

		Trace		**Proof**		**AIG**		**CNF**	
		avg	med	avg	med	avg	med	avg	med
2008	sat	823.1M	1.6M	145.1M	154.4k	49.2M	68.5k	133.7M	779.0k
	unsat	1.4G	3.0M	89.8M	351.5k	1.5M	14.9k	7.4M	626.1k
	total	1.2G	2.3M	112.4M	272.5k	20.7M	26.5k	58.4M	675.0k
2010	sat	1.3G	78.7M	518.4M	2.8M	449.4M	378.9k	1.2G	2.8M
	unsat	1.1G	9.2M	66.7M	729.9k	3.6M	13.8k	15.5M	874.7k
	total	1.2G	11.2M	265.3M	1.0M	192.7M	23.6k	524.2M	1.1M

We also compared average and median size of the generated files in terms of number of Q-resolution and reduction steps (s), number of literals (l), number of and-gates (a) and number of clauses (c) as shown in Table 4. The results show that proofs are on average over 50% smaller than the traces produced by DepQBF. A comparison of the actual file sizes of the generated files is shown in Table 5.